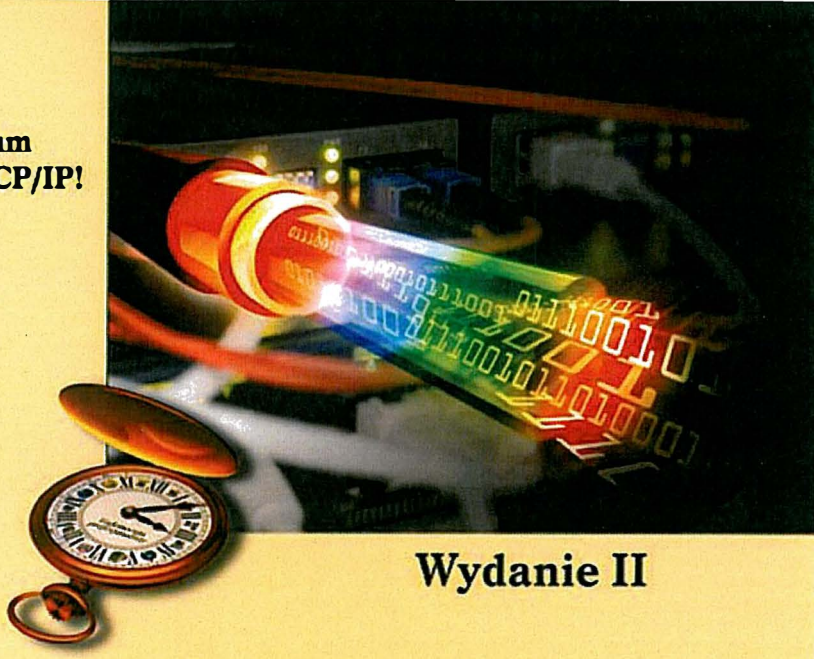
Kompendium wiedzy o TCP/IP!



Wydanie II

TCP/IP od środka Protokoły

Vademecum profesjonalisty

ADDISON-WESLEY

Kevin R. Fall
W. Richard Stevens

Helion

Opinie o pierwszym wydaniu TCP/IP od środka. Protokoły

„To niechybnie będzie biblia dla użytkowników TCP/IP i twórców aplikacji internetowych. Już po kilku minutach lektury natrafiłem na kilka scenariuszy, które w przeszłości spędzały sen z powiek mnie i moim kolegom. Steven odkrywa wiele tajemnic, skrywanych dotąd zazdrośnie przez mało rozmownych guru od sieci. W moim odczuciu — a jestem osobą zajmującą się od wielu lat implementowaniem protokołów TCP/IP — to bodaj najlepsza pozycja o tej tematyce wydana w ostatnich latach".

— Robert A. Ciampa, inżynier sieci, Synernetics, oddział 3COM

„To prawda, że wszystkie książki Stevena są doskonałe merytorycznie i znakomicie się je czyta, ta jednak naprawdę rzuca na kolana. Choć napisano wiele książek na temat protokołów TCP/IP, daleko im wszystkim do wnikliwości i szczegółowości Stevena. Czytelnik wprowadzony zostaje w tajniki mechanizmów poprzez wizualną prezentację ich funkcjonowania w rzeczywistym świecie".

— Steven Baker, felietonista „Unix Review"

„*TCP/IP od środka. Protokoły* to wspaniały materiał referencyjny dla programistów aplikacji sieciowych, administratorów sieci i w ogóle każdego, kto zainteresowany jest technologiami protokołów TCP/IP. Obszerne i szczegółowe ujęcie jest w stanie zadowolić nawet ekspertów, a jednocześnie dostarczyć niezbędnej wiedzy i komentarzy również nowicjuszom".

— Bob Williams, wiceprezes ds. marketingu NetManage, Inc.

„[...] W książce Stevena wyjątkowe jest połączenie przystępnej narracji ze szczegółową wizualną prezentacją. Proste i czytelne objaśnienia, ćwiczenia na zakończenie każdego rozdziału, szczegółowe diagramy, bajt po bajcie, nagłówków i innych struktur — nie ma lepszych narzędzi dydaktycznych".

—Walter Zintz, „UnixWorld"

„Zamiast czystej teorii — interesująca wycieczka po krainie TCP/IP, wraz z ilustracjami bazującymi na rzeczywistej sieci. Praktyczne przykłady ilustrują rzeczywisty obraz realizacji poszczególnych koncepcji, teoria staje się bardziej zrozumiała przez jej praktyczne wykorzystanie — to wszystko sprawia, że książka ta jest wysoce pouczająca, a przy tym napisana w bardzo czytelny sposób".

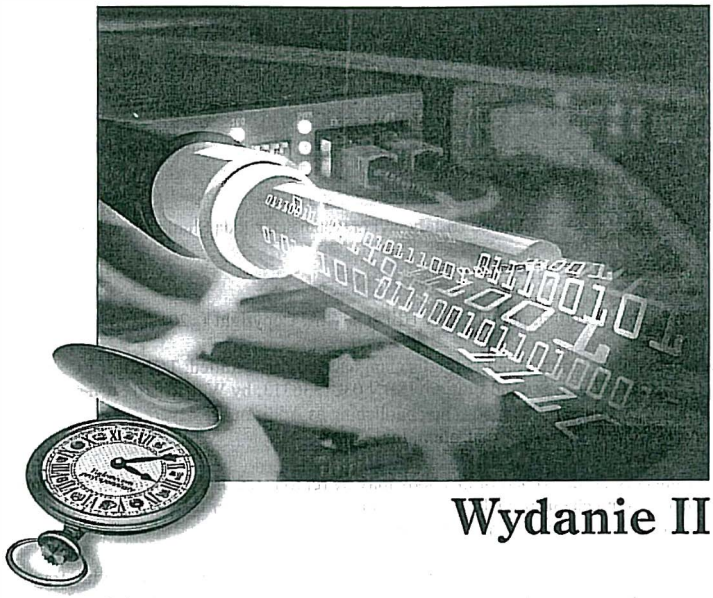
— Peter M. Haverlock, konsultant w dziale protokołów TCP/IP firmy IBM

"Wspaniałe, przejrzyste diagramy i czytelny styl narracji służą przystępnemu wyjaśnianiu prawdziwie skomplikowanych rzeczy. Książka ta naprawdę zasługuje na uwagę — przeczytaj ją i trzymaj na swej półce".

— Elizabeth Zinkann, administrator systemów

„W. Richard Stevens stworzył naprawdę piękny tekst i wspaniały materiał referencyjny. Znakomita organizacja, jasny styl i — zgodnie z tytułem — wiele czytelnych ilustracji przybliżają subtelne szczegóły logiki i funkcjonowania protokołów IP, TCP oraz innych wspomagających protokołów i aplikacji".

— Scott Bradner, konsultant Uniwersytet Harvarda, OIT/NSD



Wydanie II

TCP/IP od środka Protokoły

Vademecum profesjonalisty

ADDISON-WESLEY

Kevin R. Fall
W. Richard Stevens

Helion

Tytuł oryginału: TCP/IP Illustrated, Volume 1: The Protocols (2nd Edition)

Tłumaczenie: Andrzej Grażyński (wstęp, rozdz. 1 – 9),
Marek Pałczyński (rozdz. 10 – 11, 18), Grzegorz Pawłowski (rozdz. 12 – 17, dodatek)
Projekt okładki: Maciej Pasek

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

ISBN: 978-83-246-4815-3

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Spis treści

Słowo wstępne

Trudno napisać książkę na znany temat, obszerną i aktualną, a jednocześnie właściwie umiejscawiającą teraźniejszość na tle kontekstu historycznego. Wiarygodność przedstawienia tematu uwarunkowana jest jego rzeczowym, obiektywnym potraktowaniem, bez upiększeń — to właśnie najbardziej ujęło mnie w tej książce. Architektura TCP/IP była produktem na miarę czasów, w których została wymyślona, zaprojektowana i zrealizowana: jej (potwierdzona z upływem wielu lat) zdolność adaptacji do zmieniających się wymagań, wielokierunkowych, zwielokrotnionych o czynnik milion i więcej, nie wspominam już o ogromie aplikacji — to właśnie są fakty niezaprzeczalne i wolne od jakiegokolwiek koloryzowania. Właściwe zrozumienie, już u samych początków tej architektury, zakresu jej możliwości z jednej strony, a przyrodzonych jej ograniczeń z drugiej, stało się kluczowym czynnikiem, który umożliwił jej sukcesywną ewolucję, a w pewnym momencie nawet rewolucję.

Gdy kształtowały się zręby architektury Internetu, koncepcja „przedsiębiorstwa" — *enterprise* — daleka była jeszcze od należytego zrozumienia. Większość sieci komputerowych funkcjonowała w prywatnej przestrzeni globalnych adresów IP, eksponując owe adresy bezpośrednio na działanie globalnego systemu trasowania; z czasem jednak względy techniczne i ekonomiczne doprowadziły do ukształtowania się komercyjnych usług pośredników, zwanych fachowo „dostawcami Internetu" (ISP — *Internet Service Providers*), eksponujących bloki adresów internetowych w imieniu swych klientów. W rezultacie większość adresów IP przydzielana była w trybie „zależnym od dostawcy" (bezpośredni przydział globalnych przestrzeni adresowych był raczej rzadkością), co w krótkiej perspektywie zaowocowało powstaniem mechanizmu agregowania tras i przyhamowaniem tempa rozrostu globalnej tablicy trasowania. Mimo niewątpliwych korzyści, strategia ta okazała się jednak także problematyczna dla użytkowników korzystających z usług kilku dostawców („multihomingu"). W miarę upływu lat wielkimi krokami przybliżała się też perspektywa wyczerpania puli dostępnych adresów IP, choć ta — o liczebności przewyższającej 4 miliardy — początkowo wydawała się ogromna. Skutecznym środkiem odsunięcia nieszczęścia w czasie okazał się mechanizm translacji adresów sieciowych (NAT), który jednak w niczym nie łagodził problemu zależności od dostawcy ani komplikacji multihomingu w warunkach tej zależności.

Czytając kolejne rozdziały tej książki, nie sposób oprzeć się wrażeniu podziwu wynikającego z faktu, iż początkowo niewielki zbiór stosunkowo prostych koncepcji, wykorzystywanych w niewielu sieciach, wyewoluował do postaci dzisiejszego Internetu — po części jako rezultat spełniania coraz to nowych wymagań, dyktowanych nowymi warunkami i wyzwaniami, po części zaś lawinowo rosnącą skalą powszechnego zainteresowania.

Względy zapewnienia bezpieczeństwa „firmowym" użytkownikom doprowadziły do wynalezienia firewalli, chroniących sieć korporacyjną — na jej granicach — przed nieuprawnionym dostępem z zewnątrz. Mimo celowości i skuteczności takiej ochrony, nie można jednak zapominać, że zagrożenie wtargnąć może do wnętrza sieci również w sposób bardziej banalny, pod osłoną zawirusowanego laptopa, zainfekowanej płyty DVD czy spreparowanego pendrive'a.

Stało się jednocześnie oczywiste, iż nieuchronna perspektywa wyczerpania dostępnych adresów zmusza do opracowania i sukcesywnego wdrażania nowej wersji protokołu IP — IPv6 — ze 128-bitowymi adresami w liczbie praktycznie nieskończonej, bo wyrażającej się 39 cyframi dziesiętnymi. Rosnące zagrożenie rozmaitymi atakami zrodziło pilną potrzebę opracowania rozmaitych mechanizmów zabezpieczających, takich jak choćby Domain Name System Security Extension (DNSSEC).

Tym, co w mojej ocenie czyni tę książkę wyjątkową, jest udane połączenie właściwego stopnia szczegółowości ze spojrzeniem retrospektywnym, przybliżającym motywacje, warunki i racjonalizację opracowania poszczególnych rozwiązań w takiej, a nie innej formie. Z pewnością okaże się to przydatne inżynierom, cyzelującym operacje w swych sieciach, wypracowującym najlepsze mechanizmy ich zabezpieczania czy też poszukującym alternatywnych sposobów rozwiązywania uporczywych problemów. Autorzy zasługują na wyjątkowe uznanie za dogłębne przedstawienie technologii wykorzystywanych we współczesnym Internecie.

Vint Cerf
Woodhurst, w czerwcu 2011

Przedmowa do wydania drugiego

Witamy w drugim wydaniu *TCP/IP od środka. Protokoły*. Zadaniem tej książki jest dostarczenie aktualnego, szczegółowego obrazu zestawu protokołów TCP/IP. Zamiast werbalnego opisu ich funkcjonowania wyjaśniamy tu funkcjonowanie przy użyciu rozmaitych narzędzi analitycznych. Dzięki temu czytelnik powinien lepiej rozumieć decyzje kryjące się za projektami poszczególnych protokołów, a także detale współdziałania tychże protokołów — bez konieczności studiowania kodu źródłowego czy też budowania instalacji eksperymentalnych, choć te, oczywiście, dostarczyć mogą mnóstwo dodatkowej wiedzy.

Sieci komputerowe uległy w ciągu ostatnich 30 lat kolosalnym przeobrażeniom. Internet, stanowiący początkowo projekt badawczy i techniczną ciekawostkę dla postronnych obserwatorów, stał się dziś fabryką globalnej komunikacji, od której swe funkcjonowanie uzależniają już nie tylko instytucje rządowe i organizacje biznesowe, lecz także przeciętni użytkownicy komputerów. Architektura protokołów TCP/IP definiuje metody wymiany informacji za pośrednictwem Internetu, między dowolnymi niemal urządzeniami. Po kilkunastu latach względnej stagnacji Internet i same protokoły TCP/IP przechodzą najbardziej spektakularne przeobrażenie związane z wdrażaniem IPv6. W tej książce obie wersje protokołu — IPv4 i IPv6 — omówione zostaną łącznie, z wyjątkiem przypadków, gdy opisywane będą zasadnicze różnice między nimi. Ponieważ różnice te są znaczące na tyle, iż uniemożliwiają bezpośrednią współpracę wymienionych wersji, wspomniana ewolucja niesie ze sobą szereg poważnych wyzwań, którym projektanci starają się stawiać czoło przy użyciu stosownych mechanizmów.

Książkę tę adresujemy do wszystkich pragnących lepiej zrozumieć konstrukcję protokołów TCP/IP i ich funkcjonowanie: operatorów i administratorów sieci, programistów tworzących aplikacje sieciowe, studentów i wszystkich innych zainteresowanych użytkowników. Mamy nadzieję, że prezentowany przez nas materiał okaże się interesujący zarówno dla nowych czytelników, jak i tych, którzy mają za sobą lekturę pierwszego wydania.

Kilka słów o wydaniu pierwszym

Od publikacji pierwszego wydania tej książki minęło niemal 20 lat, w czasie których cieszyła się ona popularnością zarówno wśród studentów, jak i profesjonalistów poszukujących informacji na temat TCP/IP na poziomie szczegółowości trudno dostępnym w innych publikacjach z literatury przedmiotu. Jednak każda książka, zwłaszcza traktująca o tak

dynamicznie zmieniającej się dziedzinie wiedzy jak Internet, po pewnym czasie staje się nieaktualna. W tym wydaniu podjęliśmy zatem próbę starannego zaktualizowania treści zawartych w pionierskiej pracy dr. Stevensa, a jednocześnie staraliśmy się zachować jej wyjątkowo wysoki standard prezentacji w zakresie nowo dodanych treści.

W pierwszym wydaniu opisywany jest obszerny zestaw protokołów i ich operacji, począwszy od poziomu warstwy łącza danych, aż do poziomu aplikacji i zarządzania siecią. Minęły jednak prawie dwie dekady i zachowanie tej formy prezentacji w odniesieniu do tej tematyki w dzisiejszym stanie rzeczy skutkowałoby nadmiernym rozrostem objętości książki. Z tego względu skupiliśmy się na „rdzennych" protokołach — tych relatywnie niskopoziomowych, wykorzystywanych najczęściej w usługach związanych z konfiguracją, nazewnictwem, dostarczaniem danych i bezpieczeństwem Internetu; szczegółową dyskusję na temat aplikacji, trasowania, usług webowych i wielu innych interesujących tematów odkładamy do następnych tomów.

W czasie dzielącym to wydanie od poprzedniego dokonał się też znaczący postęp w zakresie polepszenia niezawodności implementacji protokołów grupy TCP/IP oraz ich zgodności z wzorcowymi specyfikacjami. Wiele eksponowanych w pierwszym wydaniu przykładów błędów implementacyjnych i niekompatybilnych zachowań to obecnie już historia — sporo tych błędów zostało skutecznie usuniętych, przynajmniej na gruncie IPv4, czemu trudno się dziwić wobec coraz większej popularności wspomnianych protokołów i ich implementacji w różnych systemach operacyjnych. Przejawy nieprawidłowego funkcjonowania tych implementacji, jakkolwiek spotykane, są jednak dziś rzadkością, co notabene stanowi świadectwo określonej dojrzałości całej grupy TCP/IP. Większość obecnych problemów w działaniu „rdzennych" protokołów sprowadza się do zamierzonego nadużywania rzadko wykorzystywanych funkcji — temu zagadnieniu, potraktowanemu dość pobieżnie w pierwszym wydaniu, tym razem poświęciliśmy należną uwagę.

Realia Internetu XXI wieku

Głównym motorem przeobrażeń, jakim w ostatnich dwudziestu latach podlegał Internet i wzorce jego używania, była postępująca komercjalizacja sieci WWW, rozpoczęta gdzieś na początku lat 90. ubiegłego wieku. Internet trafił pod strzechy, zagościł w życiu wielu ludzi o zróżnicowanych (często przeciwstawnych) motywacjach. Protokołom i systemom, oryginalnie opracowanym i zaimplementowanym w małej skali, na potrzeby współpracy akademickiej, przyszło zmierzyć się z problemem kurczącej się z dnia na dzień puli dostępnych adresów IP oraz wyzwaniami w zakresie bezpieczeństwa i integralności komunikacji.

W odpowiedzi na zagrożenie bezpieczeństwa administratorzy wyposażali swe sieci w specjalne elementy kontrolne — instalowanie *firewalli* na styku sieci lokalnej z Internetem jest dzisiaj powszechną praktyką zarówno w sieciach korporacyjnych, jak i małych sieciach domowych. Powstrzymywaniu tempa utylizacji dostępnych adresów IP służyć miał mechanizm translacji adresów sieciowych (NAT), implementowany w coraz większej gamie routerów nowej generacji i cieszący się wciąż niesłabnącą popularnością, umożliwiający obsługę wielu hostów w wewnętrznej sieci za pomocą małej puli globalnych adresów IP. Spełnił on bardzo dobrze pokładane w nim nadzieje, przyczyniając się jednak do spowolnienia — niestety — tempa wdrażania IPv6 i migracji na jego platformę, odsuwającą perspektywę wyczerpania adresów praktycznie w nieskończoność.

Skończyła się także epoka „osamotnionych" komputerów PC — mniej więcej od połowy lat 90. ubiegłego wieku połączenie z Internetem jest standardowym elementem ich funkcjonalnego wyposażenia. W związku z tym największy dostawca oprogramowania do pecetów — firma Microsoft — zarzucił politykę oferowania specyficznych, alternatywnych dla Internetu mechanizmów i zwrócił się w stronę kompatybilności większości produktów ze specyfikacjami TCP/IP. W rezultacie komputery osobiste sterowane różnymi wersjami systemu Windows stanowią dominującą część wszystkich pecetów łączących się z Internetem; wydaje się jednak, iż Windows traci powoli swą dominującą pozycję w tym względzie na rzecz różnych dystrybucji systemu Linux. Pozostałe systemy, na czele z Solarisem firmy Oracle i odmianami BSD, plasują się w tej kategorii jako zdecydowana mniejszość. Coraz bardziej zauważalne stają się także systemy linii Apple OS X, głównie wskutek ekspansji laptopów tego producenta — w 2003 roku sprzedaż nowych laptopów zdominowała rynek komputerów osobistych: użytkownicy pragną mobilności w połączeniu z szybkim dostępem do Internetu. Szacuje się, że rok 2012 będzie rokiem smartfonów i (prawdopodobnie) tabletów.

Obecnie prowadzące do Internetu sieci bezprzewodowe dostępne są niemal wszędzie — w restauracjach, portach lotniczych, kawiarenkach i innych miejscach publicznych. W różnych regionach świata popularność zdobywają technologie bezprzewodowego dostępu szerokopasmowego, wywodzące się z telefonii komórkowej: mowa tu m.in. o LTE, HSPA, UMTS i EV-DO, stanowiących atrakcyjną ofertę dla krajów rozwijających się, które ze względów ekonomicznych nie mogą pozwolić sobie na budowę przewodowej infrastruktury Internetu. Użytkownicy zarówno komputerów przenośnych, jak i urządzeń podręcznych żądają dostępu do Internetu w warunkach mobilności, co stwarza kilka poważnych wyzwań pod adresem architektury protokołów TCP/IP.

Po pierwsze, mobilność ma głęboki wpływ na strukturę trasowania i adresowania, przestaje bowiem obowiązywać zasada, że adresy IP przydzielane są hostom na podstawie tożsamości pobliskiego routera. Po drugie, łącza bezprzewodowe są znacznie bardziej zawodne w porównaniu z łączami kablowymi: zanik połączenia bezprzewodowego jest sytuacją normalną, podczas gdy w sieciach przewodowych gubienie danych zdarza się sporadycznie, przeważnie z powodu przeciążenia łącza.

Po trzecie wreszcie, na bazie Internetu, jako swoistej pożywki, pojawiły się aplikacje typu peer-to-peer, formujące „sieci nakładkowane". Aplikacje te — coraz popularniejsze jako alternatywa dla klasycznej architektury klient-serwer — nie działają w oparciu o centralny serwer, świadczący usługi na rzecz klientów, lecz każdy z komputerów wspomnianej sieci nakładkowanej może pełnić funkcje zarówno serwera, jak i klienta, w komunikacji ze swymi partnerami, w dziele realizacji określonego zadania. Koncepcja „nakładkowania" wywodzi się z faktu, że współpracujące ze sobą komputery partnerskie tworzą swoistą sieć, nałożoną niejako na klasyczną sieć TCP/IP (która, co łatwo zauważyć, również jest rodzajem nakładki na sieć, jaką tworzą fizyczne łącza). Tworzenie aplikacji peer-to-peer, mimo iż wielce interesujące z perspektywy przepływu ruchu czy handlu elektronicznego, nie miało i raczej nie ma bezpośredniego wpływu na kształt „rdzennych" protokołów opisywanych w tomie pierwszym; sama jednak koncepcja sieci nakładkowanej stała się jednym z generalnych zagadnień związanych z technologiami sieciowymi.

Zmiany i nowości merytoryczne w drugim wydaniu

Najbardziej istotną różnicą merytoryczną obecnego wydania w stosunku do poprzednika jest ogólna restrukturyzacja układu treści oraz dodanie sporej części materiału poświęconego bezpieczeństwu. Zamiast opisywania kompletu protokołów tworzących poszczególne warstwy modelu odniesienia, obecny tekst koncentruje się przede wszystkim na powszechnie używanych (teraz lub w najbliższej perspektywie) protokołach pomijających kontekst bezpieczeństwa — w tej grupie wymienić należy przede wszystkim Ethernet (802.3), Wi-Fi (802.11), PPP, ARP, IPv4, IPv6, UDP, TCP, DHCP i DNS. Z nimi właśnie najczęściej spotyka się każdy administrator sieci i większość użytkowników.

Tematyka bezpieczeństwa sieci zagościła na łamach drugiego wydania w sposób dwojaki. Po pierwsze, każdy z rozdziałów uwieńczony jest podrozdziałem traktującym o zagrożeniach znanymi atakami, przypuszczanymi w kontekście opisywanego w rozdziale protokołu lub mechanizmu; wyszczególnieniu istoty najważniejszych ataków towarzyszy opis znanych środków przeciwdziałania im. Czytelnicy nie powinni traktować tych opisów jako swoistej recepty — nie taka była nasza intencja — lecz raczej jako przejawy niedoskonałości określonych implementacji (w niektórych przypadkach — także specyfikacji) przedmiotowych protokołów. Należy zdawać sobie sprawę z tego, że w warunkach dzisiejszego Internetu niekompletna specyfikacja lub niedbałe praktyki implementacyjne stanowią zielone światło dla udanych ataków na systemy o krytycznym znaczeniu — ataków realizowanych nawet przy użyciu niewyszukanych środków.

Drugim obliczem ujęcia bezpieczeństwa w tej książce jest rozdział 18., w którym szczegółowo omawiane są techniki kryptograficzne i budowane na bazie tych technik mechanizmy zabezpieczeń. Czytelnicy znajdą tu m.in. opis protokołów IPsec, TLS, DNSSEC i DKIM — stanowią one podstawowy fundament implementacji dowolnej aplikacji lub usługi, od których oczekuje się zapewnienia integralności danych lub bezpieczeństwa wykonywanych operacji. Jest oczywiste, że wraz ze wzrostem komercyjnego znaczenia Internetu równie szybko pojawiają się pomysłowe narzędzia mające zapewnić swym autorom godziwe zyski w niegodziwy sposób; pomysłowość amatorów niecnych poczynań ściera się w (nierównej, niestety) walce z autorami jeszcze bardziej pomysłowych antidotów.

Mimo iż w pierwszym wydaniu nie znalazło się miejsce na opis protokołu IPv6, to jednak obecnie zyskał on należną rangę wobec tego, co zdarzyć się prędzej lub później musiało: w lutym 2011 roku reszta dostępnych jeszcze adresów IPv4 rozdysponowana została pomiędzy regionalne urzędy rejestracyjne. IPv6 z założenia wolny jest od syndromu wyczerpania adresów i choć obecnie jeszcze nie tak popularny jak IPv4, to z oczywistych względów dzierży palmę pierwszeństwa na większości obecnych i wszelkiego rodzaju przyszłych urządzeń mobilnych (telefonów komórkowych, urządzeń gospodarstwa domowego, czujników środowiskowych itp.) przyłączanych do Internetu. Uroczystości w rodzaju Światowego Dnia IPv6 (*World IPv6 Day* — 8 czerwca 2011)[1] są wyrazem tego, że Internet ma się całkiem dobrze, mimo iż jego podstawowe protokoły zostały rozszerzone i zmodyfikowane w znaczący sposób.

[1] Niespełna rok później, 6 czerwca 2012 roku, świętowano Światowe Uruchomienie IPv6 (*World IPv6 Launch*), pod hasłem *this time it's for real* — „tym razem na serio" — *przyp. tłum.*

Kolejnym zabiegiem strukturalnym w treści drugiego wydania było odsunięcie na dalszy plan omówienia protokołów, które obecnie używane są bardzo rzadko lub nieużywane w ogóle, oraz uaktualnienie opisu tych, które od czasu pierwszego wydania książki uległy znaczącym zmianom. Usunięto z treści rozdziały poświęcone protokołom RARP, BOOTP, NFS, SMTP i SNMP, zaś dyskusja na temat protokołu SLIP zastąpiona została obszernym omówieniem protokołów DHCP i PPP (włącznie z PPPoE). Funkcja forwardowania IP (której w wydaniu pierwszym poświęcono rozdział 9.), została w wydaniu drugim omówiona w kontekście ogólnego opisu IPv4 i IPv6, stanowiącego przedmiot rozdziału 5. Usunięto również omówienie protokołów dynamicznego trasowania: RIP, OSPF i BGP — dwa ostatnie z wymienionych zasługują w pełni na poświęcenie im odrębnych książek. Poczynając od protokołu ICMP, poprzez IP, TCP i UDP, omówiono łącznie te operacje IPv4 i IPv6, które w obu tych wersjach przebiegają identycznie, jednocześnie uwydatniono istniejące między nimi różnice. Nie ma w wydaniu drugim osobnego rozdziału poświęconego IPv6, omówiono go w kontekście wpływu na funkcjonowanie innych „rdzennych" protokołów. Rozdziały 15. oraz od 25. do 30. wydania pierwszego, poświęcone aplikacjom internetowym i obsługującym je protokołom, usunięto niemal w całości, pozostawiając jedynie wybrane fragmenty będące w ścisłym związku z rdzennymi protokołami.

Dodano — oczywiście — kilka nowych rozdziałów, zawierających materiał nieobecny w wydaniu pierwszym. Po wprowadzającym rozdziale 1. o raczej ogólnym charakterze następują rozdziały bardziej konkretne w treści — i tak np. w rozdziale 2. omawiamy wyczerpująco adresowanie wykorzystywane w architekturze Internetu, rozdział 6. poświęcamy konfigurowaniu hostów i generalnie temu, co dzieje się w momencie przyłączania hosta do sieci, natomiast treścią rozdziału 7. są firewalle oraz mechanizm translacji adresów sieciowych (NAT) i wykonywany przezeń podział przestrzeni adresowej na część trasowalną i nietrasowalną. Zestaw narzędzi wykorzystywanych w prezentacjach wydania pierwszego wzbogaciliśmy o program Wireshark — darmowe narzędzie GUI służące do wygodnego i czytelnego monitorowania ruchu w sieci.

Nie zmieniły się w niczym nasze zamiary i oczekiwania w odniesieniu do potencjalnych odbiorców książki: generalnie nie wymagamy jakiejś specjalnej wiedzy czy doświadczenia z zakresu sieci komputerowych, choć nawet doświadczeni w tej mierze czytelnicy z pewnością znajdą wiele szczegółów, o które będą chcieli swą wiedzę wzbogacić. Treści każdego z rozdziałów towarzyszy wykaz cytowanych źródeł o specjalistycznym charakterze.

Zmiany redakcyjne w drugim wydaniu

Generalnie układ materiału wydania drugiego jest podobny do wydania pierwszego. Po wprowadzających rozdziałach 1. i 2. prezentowane są poszczególne protokoły, w ujęciu wstępującym (*bottom-up*) ukazującym, w jaki sposób wpisują się one w cel nadrzędny — komunikację sieciową w ramach architektury Internetu. Podobnie jak w pierwszym wydaniu, funkcjonowanie poszczególnych protokołów ilustrowane jest bogato wynikami śledzenia przesyłanych pakietów; od pierwszego wydania upłynęło trochę czasu i obecnie inżynierowie oraz administratorzy mają do dyspozycji wiele nowoczesnych narzędzi GUI, bardziej wygodnych od starego dobrego programu `tcpdump` — wśród nich jest Wireshark, którego ekranami ilustrujemy często poszczególne etapy działania danego mechanizmu

lub protokołu. Nie pogardzamy jednak programem `tcpdump` w sytuacjach, gdy raporty tekstowe okazują się bardziej czytelne od ekranów GUI. Dla wygody i większej klarowności skracamy niektóre listingi, wśród nich także raporty `tcpdump`, usuwając z nich informacje nieistotne w danym kontekście.

Większość przykładów ilustrujących śledzenie pakietów odzwierciedla zachowanie się (części lub całości) przykładowej sieci przedstawionej na wewnętrznej stronie okładki — Internet łączy tu sieć domową, sieć korporacyjną i kawiarenkę internetową, czyli trzy najbardziej typowe instancje sieci składowych. Komputery znajdujące się w tych sieciach działają pod kontrolą powszechnie używanych systemów operacyjnych: Windows, Linuksa, Mac OS X i FreeBSD.

Zmodyfikowaliśmy nieznacznie strukturę poszczególnych rozdziałów. Każdy rozdział zaczyna się od wstępu, po czym następuje szczegółowy opis tematu, przeplatany niekiedy notatkami odzwierciedlającymi kontekst historyczny. Rozdział kończy się trzema podrozdziałami, z których pierwszy poświęcony jest zagadnieniu znanych ataków wykonywanych z użyciem opisywanych w rozdziale mechanizmów (lub ataków skierowanych przeciwko tymże mechanizmom), drugi ma charakter podsumowujący, trzeci natomiast stanowi wykaz literatury specjalistycznej cytowanej lub przywoływanej w treści rozdziału. Inaczej niż w wydaniu pierwszym, wykaz ten rozdzielony został pomiędzy odnośne rozdziały, które dzięki temu stały się bardziej kompletne, a czytelnik uwolniony został od kłopotliwego przeskakiwania między czytanym właśnie rozdziałem a częścią bibliograficzną. Wiele z pozycji cytowanej literatury dostępnych jest obecnie w wersji online, dlatego też zostały one w wykazie uzupełnione o odsyłacze URL. Nadaliśmy też bardziej zwartą formę identyfikatorom pozycji, przykładowo identyfikator [Cerf and Kahn 1974] z wydania pierwszego zyskał teraz bardziej zwartą formę [CK74], utworzoną z pierwszych liter nazwisk autorów i dwóch ostatnich cyfr roku ukazania się pozycji. Ujednolicono też sposób identyfikowania odsyłaczy prowadzących do dokumentów RFC, co spowodowało zgrupowanie tych odsyłaczy w ciągłym obszarze (nie było takiej ciągłości, gdy dokumenty te identyfikowane były przy użyciu nazwisk autorów).

Kevin R. Fall
Berkeley, Kalifornia
we wrześniu 2011

Przedmowa do wydania pierwszego

Wprowadzenie

Treścią tej książki jest opis zestawu protokołów TCP/IP, ukazanego jednak z innej perspektywy, niż zwykło się to czynić w większości publikacji o tej tematyce. Zamiast werbalnego wyjaśnienia rozmaitych szczegółów, czytelnicy otrzymują raporty „na żywo" z działania protokołów, sporządzone przy użyciu popularnych narzędzi diagnostycznych; to znacznie bardziej pouczające — i zarazem dostarczające wiedzy na temat wielu szczegółów implementacyjnych, bez konieczności zagłębiania się w lekturę kodu źródłowego.

Gdy w latach 1960–1980 konstruowano pierwsze protokoły sieciowe, inżynierowie nie mieli łatwego zadania: podglądnięcie pakietów „śmigających po drutach" wymagało po pierwsze, kosztownego, dedykowanego sprzętu, a po drugie — znajomości testowanego protokołu w stopniu ekstremalnym, by wyświetlaną informację prawidłowo interpretować. Funkcjonalność analizatorów sprzętowych ograniczona była do tej „zaszytej na sztywno" przez producentów.

Dziś sprawy zmieniły się radykalnie. Monitorowanie sieci lokalnych jest znakomicie ułatwione dzięki wszechobecnym stacjom roboczym ([Mogul 1990]): podłącza się stację do sieci, uruchamia dowolny z wielu publicznie dostępnych programów diagnostycznych — i już można delektować się pożądaną informacją, w dodatku sformatowaną w czytelnej konwencji. I choć takie narzędzia tworzone są głównie do celów *diagnostycznych*, to — oczywiście — równie dobrze spełniać mogą rolę *edukacyjną*, bo praktyczna obserwacja tego czy innego zjawiska pozwala lepiej rozumieć jego szczegóły, które w innych okolicznościach być może pozostawałyby wysoce zagadkowe. Ten właśnie fakt oddaje w pełni intencje, jakimi kierowałem się przy pisaniu tej książki.

Książki adresowanej do każdego, kogo interesują szczegóły działania protokołów z zestawu TCP/IP: programisty aplikacji sieciowych, administratora systemu odpowiedzialnego za prawidłowe funkcjonowanie systemów i sieci wykorzystujących rzeczone protokoły, a także użytkowników, którzy na co dzień stykają się z aplikacjami bazującymi na tych protokołach.

Konwencje typograficzne

W raportach i listingach informację wyświetlaną (wypisywaną) przez komputer przedstawiamy w postaci `czcionki o stałej szerokości`, natomiast informacja wprowadzana przez użytkownika ma formę **`pogrubionej czcionki o stałej szerokości`**. Komentarze wyróżniamy za pomocą *kursywy*:

```
bsdi % telnet svr4 discard      próba połączenia z serwerem discard
Trying 140.252.13.34...         ta linia i następne wyświetlane są przez program telnet
Connected to svr4.
```

Jak łatwo się zorientować, używamy znaków zachęty („prometów") odzwierciedlających używany system operacyjny (`bsdi` w powyższym przykładzie).

Okazjonalnie w treść rozdziałów wtrącane są notatki nawiązujące do historii omawianego tematu lub wyjaśniające niektóre jego szczegóły.

Odwołując się do poleceń systemowych, często stosujemy konwencję przyjętą w podręcznikach uniksowych, polegającą na dołączaniu do nazwy polecenia liczby zamkniętej w nawiasy — np. `ifconfig(8)`. Liczba ta to numer sekcji na „stronie" podręcznika poświęconej danemu programowi, czyli odsyłacz do bardziej szczegółowych informacji. Problem jednak w tym, że nie wszystkie systemy linii uniksowej organizują swe podręczniki w taki sam sposób; wykorzystywana przez nas numeracja zgodna jest z systemami grupy BSD (i systemami pochodnymi, m.in.. SunOS 4.1.3).

Podziękowania

Mimo iż nazwisko autora jest jedynym figurującym na okładce, każda książka jest owocem skoordynowanego wysiłku wielu osób. Przede wszystkim cierpliwości najbliższych w trakcie tych długich, dziwacznych godzin; Sally, Bill, Ellen i David — jeszcze raz serdecznie dziękuję.

Konsultant Brian Kernighan niewątpliwie należy do najlepszych w swoim fachu. To on jako pierwszy czytał szkice rękopisu, ozdabiając je nieskończonością czerwonych ornamentów. Jego wnikliwość, zwracanie uwagi zarówno na szczegóły, jak i na czytelność narracji oraz dogłębna weryfikacja kompletnego rękopisu to nieodmiennie skarb dla każdego autora.

Redaktorzy techniczni zaprezentowali nieco inny punkt widzenia, przywołując autora do porządku przez odnajdywanie i poprawianie błędów technicznych. Finalny produkt zyskał bardzo wiele dzięki ich komentarzom, sugestiom i (co najważniejsze) krytycznemu spojrzeniu. Dziękuję Stevowi Bellovinowi, Jonowi Crowcroftowi, Pete Haverlockowi i Dougowi Schmidtowi za komentarze dotyczące całości rękopisu. Równie serdecznie dziękuję Dave'owi Bormanowi za wnikliwy przegląd rozdziałów poświęconych protokołowi TCP oraz Bobowi Gilliganowi, który faktycznie jest współautorem dodatku E.

Jako autor, który nie jest w stanie pracować w odosobnieniu, chciałbym podziękować wielu osobom za okazaną mi wyrozumiałość, serdeczność i — oczywiście — wkład merytoryczny w postaci odpowiedzi na moje liczne e-maile. Do tej grupy zaliczają się m.in. Joe Godsil, Jim Hogue, Mike Karels, Paul Lucchina, Craig Partridge, Thomas Skibo i Jerry Toporek.

Treść książki opiera się w istocie na odpowiedziach na wiele pytań, na które trudno było mi znaleźć szybką i oczywistą odpowiedź. Przekonałem się osobiście, że najlepszym sposobem poszukiwania takich odpowiedzi jest samodzielne eksperymentowanie — wymusza się wystąpienie pewnych warunków, przeprowadza małe testy i z ciekawością oczekuje na rezultaty. Ogromne znaczenie ma w tym dziele umiejętność zadawania właściwych pytań — dużo w tym względzie nauczył mnie Peter Haverlock — oraz dobór właściwych narzędzi testowych, w szczególności oprogramowania, za co serdeczne dzięki dla Vana Jacobsona.

Książki traktującej o sieciach komputerowych i Internecie nie sposób tworzyć bez bieżącego dostępu do niebanalnej sieci przyłączonej do Internetu. Dziękuję w związku z tym National Optical Astronomy Observatories (NOAO), a szczególnie Disneyowi Wolffowi, Richardowi Wolffowi i Stevowi Grandiemu za udostępnienie sieci i hostów oraz założenie i utrzymywanie kont. Dziękuję też Keithowi Bosticowi i Kirkowi McKusickowi z grupy badawczej systemów komputerowych (CSRG) Uniwersytetu Kalifornijskiego w Berkeley za udostępnienie najnowszej wersji systemu 4.4BSD.

Wreszcie, rzecz oczywista — nie można zapominać o wydawcy, który dołożył wszelkich starań, by czytelnicy otrzymali jak najlepszy produkt. Wielkie dzięki dla redaktora Johna Waita, najlepszego z najlepszych — z przyjemnością wspominam współpracę z nim i zespołem profesjonalistów z Addison-Wesley.

Skład wykonany został przez autora, za pomocą nieśmiertelnego procesora tekstu Troff z wykorzystaniem pakietu Groff autorstwa Jamesa Clarka.

W. Richard Stevens
Tucson, Arizona
w październiku 1993

Rozdział 1. Wprowadzenie

Efektywność komunikacji uwarunkowana jest uzgodnieniem wspólnego języka; zasada ta odnosi się zarówno do komunikacji międzyludzkiej i porozumiewania się zwierząt, jak i do komunikacji między komputerami. Wspólny język w połączeniu ze wspólnymi wzorcami zachowania konstytuuje *protokół* — w słowniku *New Oxford American Dictionary* pojęcie to wyjaśniane jest następująco:

> Oficjalna procedura lub system reguł, dotyczące spraw wagi państwowej lub relacji dyplomatycznych.

Z protokołami spotykamy się — mniej lub bardziej świadomie — w życiu codziennym, zadając pytania lub odpowiadając na nie, negocjując kontrakty biznesowe, współpracując przy tworzeniu oprogramowania czy też przejeżdżając przez skrzyżowanie zgodnie ze wskazaniami sygnalizatorów. Komunikacja między komputerami także podporządkowana jest rozmaitym protokołom. Kolekcja powiązanych protokołów nazywana jest *zestawem protokołów* (*protocol suite*), zaś specyfikacja określająca sposób współdziałania elementów tego zestawu określana jest mianem architektury lub *modelu odniesienia* (*reference model*). TCP/IP to przykład zestawu protokołów implementujących architekturę Internetu, wywodzącą swe źródło z modelu odniesienia *ARPANET Reference Model* (ARM) [RFC0871]. Sam model ARM narodził się natomiast na bazie wczesnych prac nad sieciami komutacji pakietów, autorstwa m.in. Amerykanów Paula Barana [B64] i Leonarda Kleinrocka [K64], Brytyjczyka Donalda Daviesa [DBSW66] oraz Francuza Louisa Pouzina. Opracowano wiele innych architektur, takich jak *ISO protocol architecture* [Z80], XNS firmy Xerox [X85] oraz SNA firmy IBM [I96], zdecydowanie jednak to TCP/IP zyskał największą popularność. Napisano wiele interesujących książek na temat historii komunikowania się komputerów i genezy Internetu — w tym miejscu polecamy pozycje [P07] i [W02].

Godny uwagi jest fakt, że architektura TCP/IP ukształtowała się jako rezultat zabiegów zmierzających do zapewnienia połączeń między *różnymi* sieciami komutacji pakietów (patrz [CK74]). Połączenia te realizowano za pomocą *bram* (*gateways*), zwanych także *śluzami*, a później *routerami* — ich zadaniem było swoiste „tłumaczenie" funkcjonalności na styku dwóch lub więcej niekompatybilnych sieci. Taka „skonkatenowana" sieć, w rodzimym kręgu projektantów zwana *catenet*, zyskała sobie miano *międzysieci* (*internetwork*); ponieważ węzły takiej międzysieci, należące do różnych sieci składowych, mogły wymieniać się różnorodnymi usługami, stało się oczywiste, iż połączenie wielu sieci w międzysieć skutkuje znaczącą „wartością dodaną" — dodaną do sumarycznej funkcjonalności każdej sieci z osobna. Co ciekawe, wizja globalnej międzysieci pojawiła się

na wiele lat przed wyłonieniem się konkretnego kształtu architektury TCP/IP; przykładowo już w 1968 roku J.C.R. Licklider i Bob Taylor przedstawili koncepcję „nadspołeczności" (*supercommunity*) czerpiącej wielorakie korzyści z takowej międzysieci (patrz [LT68]):

> Współczesne społeczności online odseparowane są od siebie barierami wynikającymi bądź to z oddalenia geograficznego, bądź też z różnic w poszczególnych infrastrukturach. Każdy przedstawiciel określonej społeczności ma możliwość przetwarzania i magazynowania informacji w zakresie ograniczonym przez specyfikę infrastruktury, wokół której koncentruje się bytowanie owej społeczności. Nadszedł więc czas zintegrowania odrębnych społeczności w globalną — nazwijmy to — nadspołeczność, której każdy przedstawiciel zyskuje dostęp do wszystkich, uprzednio podzielonych, zasobów (danych i programów). [...] Wszystko to tworzy labilną sieć wszystkich sieci — nieustannie ewoluującą zarówno pod względem treści, jak i konfiguracji.

Jest więc oczywiste, że idea globalnej sieci, ucieleśniona najpierw pod postacią ARPANET-u, a później Internetu, zaprojektowana została z myślą o udostępnieniu wszystkich tych wspaniałości, którymi cieszą się dziś miliardy internautów na całym świecie. Droga od pomysłu do wykonania — sformułowanej koncepcji, prototypów, a ostatecznie produktów komercyjnych — nie była jednak ani prosta, ani też oczywista: osiągnięty sukces stanowi owoc starannego projektowania, pomysłowości i wizjonerstwa, wysiłku programistów i zaangażowania użytkowników z jednej strony, a dostępności wymaganych zasobów z drugiej.

W tym rozdziale przedstawiamy w ogólnym zarysie architekturę Internetu i protokołu TCP/IP, także w kontekście historycznym, stwarzając w ten sposób pewien fundament dla treści następnych rozdziałów. Projektowanie architektury — zarówno w sensie fizycznym, jak i w sensie wykorzystywanych protokołów — jest bardziej sztuką niż nauką, stanowi kwestię wielu kompromisów dotyczących wyboru elementów funkcjonalnych i ich logicznego umiejscowienia w implementacji. Kryteria tych wyborów zawsze obarczone są znaczącą dozą subiektywizmu, ponadto jako ściśle uwarunkowane stanem otaczającego świata w naturalny sposób podlegają różnorodnym przeobrażeniom wraz z upływem czasu. Czytelników szczególnie zainteresowanych architekturą sieci komputerowych odsyłamy do książki Daya [D08], jednej z nielicznych publikacji tego rodzaju.

1.1. Założenia architektoniczne

W XXI wieku trudno wyobrazić sobie codzienne życie, działalność biznesową, czy choćby rozrywkę, bez efektywnej, globalnej komunikacji. Protokoły grupy TCP/IP umożliwiają wzajemne komunikowanie się różnorodnym urządzeniom — komputerom, smartfonom i systemom wbudowanym, pochodzącym od różnych producentów i zarządzanym przez odmienne oprogramowanie. Protokoły te składają się na prawdziwie *otwarty system* — ich specyfikacje oraz większość implementacji dostępne są za darmo lub po umiarkowanej cenie. Protokoły te tworzą osnowę tego, co nazywamy *globalnym Internetem* (lub, po prostu, Internetem), a co fizycznie jest ogromną siecią rozległą (WAN) łączącą ponad dwa miliardy użytkowników (w roku 2010), czyli ok. 30% ziemskiej populacji. Mimo iż pojęcie „Internet" utożsamiane bywa ze skrótem WWW (od ang. *World Wide Web* — sieć ogólnoświatowa), to w rzeczywistości WWW jest aplikacją wykorzystującą

Internet jako medium komunikacyjne — aplikacją bezdyskusyjnie najpopularniejszą, napędzającą rozwój Internetu od ponad dwudziestu lat.

Jak pisze Clark [C88], najważniejszym celem, jaki postawili sobie projektanci architektury Internetu, było „opracowanie efektywnej techniki zwielokrotnionego wykorzystywania możliwości tkwiących w połączonych sieciach". Przymiotnik „zwielokrotniony" oznacza w tym kontekście *równoległe* uruchamianie wielu różnych usług, dostarczanych przez wspomniane sieci. Oprócz owego celu nadrzędnego, Clark wylicza też kilka celów drugoplanowych. Oto one.

- Komunikacja przez Internet musi trwać nieprzerwanie nawet w przypadku awarii sieci lub bram.
- Internet musi udostępniać wiele typów usług komunikacyjnych.
- Architektura Internetu musi być zgodna z architekturą wielu różnych sieci.
- Musi istnieć możliwość rozproszonego zarządzania zasobami architektury Internetu.
- Koszty związane z architekturą Internetu muszą mieścić się w rozsądnych granicach.
- Dołączanie nowych hostów do Internetu powinno odbywać się kosztem minimalnego wysiłku.
- Muszą istnieć mechanizmy kontroli i audytu wykorzystywania zasobów architektury Internetu.

Jest naturalne, że wymienione cele, jako w pewnym stopniu sprzeczne ze sobą, wymagały rozstrzygania licznych kompromisów, co powodowało podejmowanie decyzji projektowych cokolwiek różnych od tych, jakie ostatecznie ukształtowały architekturę Internetu. Dalej prześledzimy najważniejsze z tych decyzji i opiszemy konsekwencje przyjęcia określonych opcji projektowych.

1.1.1. Pakiety, połączenia i datagramy

Jeszcze w latach 60. ubiegłego wieku koncepcja „sieci" kojarzyła się przeważnie z siecią telefoniczną, w ramach której dwa aparaty telefoniczne łączone były ze sobą na czas trwania rozmowy. Owo „łączenie" polegało na zestawianiu obwodu (*circuit*) — początkowo fizycznego obwodu galwanicznego — istniejącego aż do zakończenia rozmowy i fizycznego rozłączenia. Na potrzeby rozliczeń (bilingów) rejestrowany był czas trwania każdej rozmowy i numery telefonów rozmówców. Wspomniany obwód udostępniał rozmówcom pewną *pojemność* lub *pasmo* częstotliwości, zwykle wystarczające dla komunikacji głosowej. Analogowe początkowo sieci telefoniczne wyewoluowały z czasem do postaci cyfrowej, co znacząco poprawiło ich niezawodność i wydajność. Dane „wpuszczane" do sieci na jednym końcu odwodu, po przebyciu pewnej ustanowionej a priori ścieżki, pojawiały się w przewidywalny sposób na drugim jego końcu po upływie pewnego czasu *opóźnienia* (*latency*), nieprzekraczającego predefiniowanego limitu. Obwód utrzymywany był przez cały czas trwania rozmowy, nawet jeśli jego możliwości nie były wykorzystywane; koncepcja ta przetrwała w telefonii do dziś — gdy rozmowa rozliczana jest w taryfie czasowej, minuta romantycznego milczenia kosztuje tyle samo, co minuta burzliwej dyskusji.

Jedną z istotnych koncepcji wspomnianego okresu, opisywaną m.in. w pracy [B64], jest *komutacja pakietów* (*packet switching*). Informacja w postaci cyfrowej dzielona jest na „kawałki" o określonej wielkości (w bajtach), zwane *pakietami*, które mogą przepływać przez sieć w dużym stopniu niezależnie od siebie. Pakiety pochodzące z różnych źródeł mogą być łączone w pojedynczy strumień — co nazywa się *multipleksowaniem* — i na powrót rozdzielane — ten zabieg nosi nazwę *demultipleksowania*. Droga przepływu pakietów nie jest ustalona z góry, lecz może się dynamicznie zmieniać. Daje to korzyści dwojakiego rodzaju: po pierwsze, sieć staje się bardziej odporna na zakłócenia i ataki (tak, projektanci brali pod uwagę możliwość przypuszczania fizycznych ataków na sieci), po drugie — zasoby sieci (łącza i przełączniki — *switches*) mogą być wykorzystywane bardziej efektywnie dzięki zastosowaniu *multipleksowania statystycznego*.

Pakiety docierające do przełącznika (komutatora) magazynowane są w *buforze pamięciowym* i formują *kolejkę* (*queue*), z której pobierane są zgodnie z zasadą „pierwszy nadszedł, pierwszy obsłużony" (*first-come-first-served* — FCFS), bardziej znaną pod akronimem FIFO (*first-in-first-out*). Algorytm FIFO w połączeniu z prognozowaniem zapotrzebowania na pasmo umożliwia łatwe zaimplementowanie multipleksowania statystycznego — podstawowej metody mieszania ruchu pochodzącego z różnych źródeł w Internecie. Jest to metoda prosta i skuteczna, bo jeżeli istnieje w sieci wolna „moc przerobowa" i ruch, który mógłby ją zagospodarować, to z pewnością zostanie wykorzystana — zasoby sieci są więc w pełni wykorzystywane stosownie do potrzeb. Podejście takie ma jednak podstawową wadę — ograniczoną przewidywalność: rzeczywista wydajność konkretnej aplikacji zależna jest od (statystycznego) zachowania innych aplikacji pracujących w tej samej sieci. Multipleksowanie statystyczne można porównać do autostrady, na której samochody mogą zmieniać pasy ruchu w celu wykorzystywania do maksimum każdej okazji do jak najszybszego przejazdu.

Techniki alternatywne — *multipleksowanie z podziałem czasu* (TDM — *time-division multiplexing*) i *multipleksowanie statyczne* (*static multiplexing*) rezerwują dla każdego połączenia pewien odcinek czasu (lub pewne porcje innych zasobów). Technika ta — stosowana w telefonii ze stałą szybkością transmisji (*bitrate*) — cechuje się znacznie lepszą przewidywalnością efektu, lecz prowadzić może do marnotrawienia zasobów sieci, bo nie zawsze są w pełni wykorzystywane.

Podczas gdy tradycyjne obwody mogą być z łatwością implementowane z użyciem TDM, technika transmisji pakietów w ramach połączeń umożliwia implementowanie *obwodów wirtualnych* (VC — *virtual circuits*), przejawiających zasadnicze cechy zachowania „prawdziwych" obwodów, lecz niezależnych od połączeń fizycznych. Obwody wirtualne stanowią podstawę protokołu o nazwie X.25, popularnego w ostatniej dekadzie ubiegłego wieku, wypartego później przez protokół Frame Relay, *cyfrowe linie abonenckie* (DSL — *digital subscriber lines*) i modemy kablowe (o czym dokładniej piszemy w rozdziale 3.).

Abstrakcja obwodu wirtualnego (i generalnie wszystkie sieci transmitujące pakiety w oparciu o połączenie, takie jak X.25) wymaga utrzymywania przez przełączniki pewnej informacji o *stanie* poszczególnych połączeń. Każdy pakiet niesie zaledwie szczątkową postać tej informacji, głównie w postaci indeksów do globalnej tablicy stanu. Przykładowo w pakiecie X.25 rolę tę pełnią 12-bitowe *identyfikator kanału logicznego* (LCI — *logical channel identifier*) i *numer kanału logicznego* (LCN — *logical channel number*): w porównaniu z informacją o *stanie przepływu* (*per-flow state*) jest ona wykorzystywana przez przełącznik do określenia następnego przełącznika na trasie pakietu. Informacja

o stanie przepływu tworzona jest przed rozpoczęciem wymiany danych przez obwód wirtualny, za pomocą protokołu sygnalizacyjnego odpowiedzialnego za nawiązywanie i rozłączanie połączenia oraz zarządzanie informacją o jego statusie. Sieci o takiej charakterystyce określane są powszechnie mianem sieci *zorientowanych na połączenie* (*connection-oriented*).

Sieci zorientowane na połączenie oparte zarówno na obwodach, jak i pakietach stanowiły przez długie lata znakomitą większość sieci, jednakże już u schyłku lat 60. ubiegłego wieku pojawiła się koncepcja konkurencyjna — *datagram* — urzeczywistniona po raz pierwszy w systemie CYCLADES (patrz [P73]). Datagram to pakiet zawierający *całość* informacji na temat źródła i przeznaczenia zamiast tylko indeksów do globalnych tablic przechowywanych w przełącznikach. Mimo iż z oczywistych względów obecność tej informacji przyczynia się do zwiększenia rozmiaru pakietu, to jednak pakiet staje się samodzielny i w konsekwencji zbędne jest nawiązywanie połączenia i angażowanie w związku z tym skomplikowanych protokołów sygnalizacyjnych. Możliwość budowania na tej podstawie sieci *bezpołączeniowych* (*connectionless*) była ochoczo wykorzystywana przez projektantów wczesnych wersji Internetu, co notabene wywarło głęboki wpływ na proces rozwoju protokołów grupy TCP/IP.

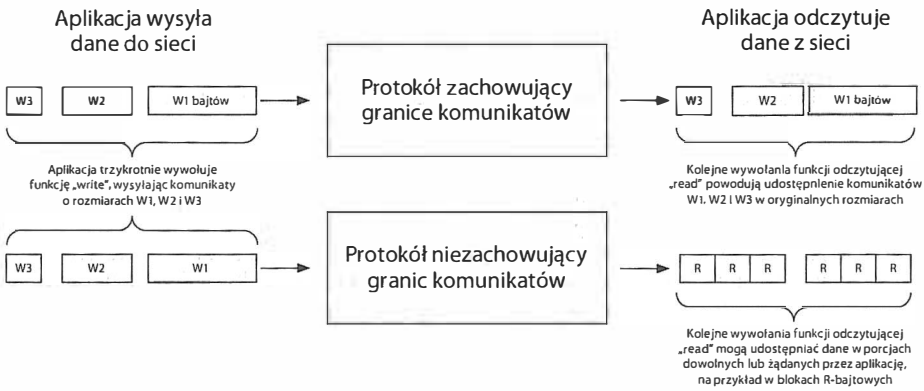
Istotnym zagadnieniem związanym z komunikowaniem się aplikacji przez sieć jest *zachowywanie* (albo niezachowywanie) *granic komunikatów*. Na rysunku 1.1 można zobaczyć, że aplikacja nadawca dzieli przesyłane dane na poszczególne porcje (komunikaty) i sposób tego podziału może, *lecz nie musi*, być zachowywany przez protokół komunikacyjny — w tym drugim przypadku wspomniane dane komasowane są przez ten protokół do postaci pojedynczego strumienia, z którego aplikacja docelowa może odczytywać dane w dowolnie wybranych porcjach. Większość protokołów opartych na datagramach honoruje rzeczony podział, co wynika bezpośrednio z wyraźnie zdefiniowanych granic poszczególnych datagramów; w przypadku obwodów (fizycznych lub wirtualnych) sprawy mogą jednak wyglądać inaczej — jeżeli informacja dotycząca podziału przesyłanych danych jest istotna dla komunikujących się aplikacji, muszą one we własnym zakresie zapewnić jej utrzymywanie.

1.1.2. Zasady „end-to-end argument" i „fate sharing"

Przy projektowaniu dużych systemów, takich jak systemy operacyjne czy zestawy protokołów, pojawia się nieodłącznie pytanie o umiejscowienie poszczególnych cech lub elementów funkcjonalnych w implementacji tychże systemów. Jedna z najważniejszych zasad, ułatwiająca udzielanie odpowiedzi na te pytania, nosi w oryginale nazwę *end-to-end argument* — jej istotę tak oto wyjaśniają autorzy publikacji [SRC84]:

> Odnośna funkcja może być zaimplementowana w sposób poprawny i kompletny jedynie w kontekście wykorzystujących ją aplikacji, komunikujących się ze sobą, wykonanie takiej implementacji nie jest więc możliwe na poziomie systemu komunikacyjnego. (System komunikacyjny może jednak dostarczać częściową implementację niektórych funkcji, jeśli może się to przyczynić do poprawy jego wydajności).

Wspomniana zasada może się w pierwszej chwili wydawać zgoła oczywista, mimo to wywarła istotny wpływ na sposób projektowania systemów komunikacyjnych. Zauważmy, iż akcentuje ona niemożliwość osiągnięcia poprawności i kompletności projektu w oderwaniu



Rysunek 1.1. *Aplikacja dostarcza kolejne komunikaty protokołowi komunikacyjnemu; protokół ten może utrzymywać informację związaną z tym podziałem, zapamiętując wielkości poszczególnych porcji (lub, co na jedno wychodzi, offsety miejsc podziału względem początku strumienia danych), i wówczas aplikacja docelowa ma możliwość odtworzenia tychże porcji. Protokoły strumieniowe, m.in. TCP, zwykle ignorują tę informację i transmitują dane w postaci skomasowanego strumienia; jeśli informacja dotycząca podziału ma zostać zachowana, muszą o to zadbać same aplikacje*

od aplikacji (lub użytkowników) stosujących system komunikacyjny, bo projektowanie wymagałoby wówczas „przewidywania nieprzewidywalnego", czyli próby trafnego odgadywania wszystkich potrzeb aplikacji, jakie kiedykolwiek będą z usług tego systemu korzystać. Wynika stąd ważny wniosek, iż istotne funkcje komunikacyjne (kontrola błędów, szyfrowanie, potwierdzanie dostarczenia) nie powinny być implementowane na niskich poziomach („warstwach" — patrz punkt 1.2.1) dużych systemów. Niższe warstwy mogą jednak dostarczać elementy wspomagające, ułatwiające implementowanie wspomnianych funkcji w warstwach wyższych i usprawniające ogólną wydajność systemu, z założenia jednak elementom tym będzie daleko do doskonałości (z perspektywy potrzeb komunikujących się aplikacji).

Zasada *end-to-end-argument* prowadzi więc do filozofii projektowej typu „głupia sieć, inteligentne końcówki", czyli łączenia inteligentnych systemów za pomocą sieci o prymitywnej raczej funkcjonalności. Wyraźny przejaw tej filozofii obserwujemy w zestawie protokołów TCP/IP, gdzie wiele istotnych funkcji — zapobieganie gubieniu danych, kontrolowanie tempa wysyłania danych przez aplikację nadawcę itp. — realizują hosty, na których uruchomione są komunikujące się aplikacje.

Wybór zestawu funkcji przeznaczonych do zaimplementowania w tym samym komputerze jest natomiast przedmiotem innej zasady projektowej, zwanej *fate sharing* (patrz [C88]). Zgodnie z tą zasadą, należy umiejscawiać w komunikujących się hostach implementacje wszystkich funkcji związanych z utrzymywaniem stanu aktualnego skojarzenia (np. połączenia wirtualnego), bo w tych warunkach zniszczenie wspomnianego skojarzenia może nastąpić jedynie w przypadku awarii któregoś ze skojarzonych hostów (co oznacza ewidentny koniec komunikacji). Dzięki takiemu rozwiązaniu możliwe jest utrzymywanie połączeń wirtualnych (np. implementowanych przez TCP) nawet w obliczu załamania się komunikacji w sieci (rzecz jasna na umiarkowany czas). Łatwo spostrzec, że zasada *fate sharing* zgodna jest z opisywaną wcześniej zasadą „głupiej" sieci i „inteligentnych"

końcówek. Obie zasady wyraźnie wyznaczają stosowane obecnie kryteria rozdziału funkcji na implementowane w systemach komunikacyjnych i implementowane w komunikujących się aplikacjach.

1.1.3. Kontrola błędów i sterowanie przepływem

Zdarza się, że dane transmitowane przez sieć ulegają zagubieniu lub zniekształceniu z różnych przyczyn: awarii sprzętu, zakłóceń elektromagnetycznych, utraty zasięgu w sieci bezprzewodowej itp. Ogół środków, których zadaniem jest radzenie sobie z takimi sytuacjami, nosi nazwę *kontroli błędów* (*error control*). Procedury kontroli błędów mogą być implementowane w systemach końcowych, w sieci łączącej te systemy bądź też z wykorzystaniem obu tych możliwości; oczywiście, zasady *end-to-end argument* i *fate-sharing* przemawiają za pierwszą z wymienionych ewentualności.

Gdy przekłamaniu ulega niewielka liczba bitów, wykrycie tego faktu i odtworzenie poprawnych wartości możliwe jest przy użyciu różnych kodów korekcyjnych (patrz [LC04]) i zadanie to wykonywane jest zazwyczaj w ramach sieci. Gdy jednak pakiet ulega poważniejszemu uszkodzeniu, jest przesyłany ponownie, czyli *retransmitowany*. W sieciach opartych na obwodach (fizycznych lub wirtualnych) — np. w sieciach X.25 — retransmisja jest funkcją automatycznie zapewnianą przez sieć. Stwarza to komfortowe środowisko dla aplikacji bezwzględnie wymagających niezawodnego dostarczenia wszystkich kolejno wysłanych pakietów, dla wielu innych aplikacji wiąże się jednak z niepotrzebnym narzutem — te potrafią radzić sobie z przypadkami gubienia pakietów i ich przybywania w kolejności innej niż oryginalna kolejność wysyłania; fatyga związana z nawiązywaniem połączenia oraz opóźnienia powiązane z retransmisją okazują się zbędnym balastem.

W związku z powyższym alternatywą dla zapewnianego przez sieć uporządkowanego i niezawodnego transportu jest koncepcja usług innego rodzaju, zwana potocznie *ruchem niegwarantowanym* (*best-effort delivery*), zastosowana m.in. w projekcie Frame Relay i protokołu IP. W sieciach opartych na tej koncepcji dostarczenie danych bez ich utraty i zniekształceń jest — oczywiście — podstawowym celem, ale nie celem, który osiągnąć należy za wszelką cenę: kontrola poprawności ogranicza się zwykle do poziomu podstawowego, czyli *sum kontrolnych*, szczególnie w odniesieniu do danych decydujących o trasie pakietu, a pakiety uznane za błędne są po prostu odrzucane, bez żadnych dodatkowych zabiegów.

Nawet przy założeniu, że pakiety transmitowane są przez sieć bez utraty i zniekształceń, stajemy przed kolejnym problemem: aplikacja nadawca może mianowicie produkować dane w tempie szybszym niż możliwości konsumowania tychże danych przez aplikację docelową. Wtedy w sieciach IP wymagane spowolnienie tempa osiągane jest przez mechanizm *sterowania przepływem* (*flow control*). Mechanizm ten implementowany jest poza systemem komunikacyjnym — jak zobaczymy w rozdziałach 15. i 16., kontrola przepływu spoczywa w gestii protokołu TCP. Takie rozwiązanie zgodne jest z zasadą *end-to-end argument*, bo implementacje protokołu TCP rezydują w komunikujących się hostach, uzgadniających na bieżąco szybkość wymiany danych. Rozwiązanie to jest także zgodne z zasadą *fate-sharing*: awaria w ramach infrastruktury sieciowej nie musi powodować zerwania połączenia między hostami, jeżeli możliwe jest ustanowienie między nimi jakiejkolwiek operatywnej trasy dla pakietów.

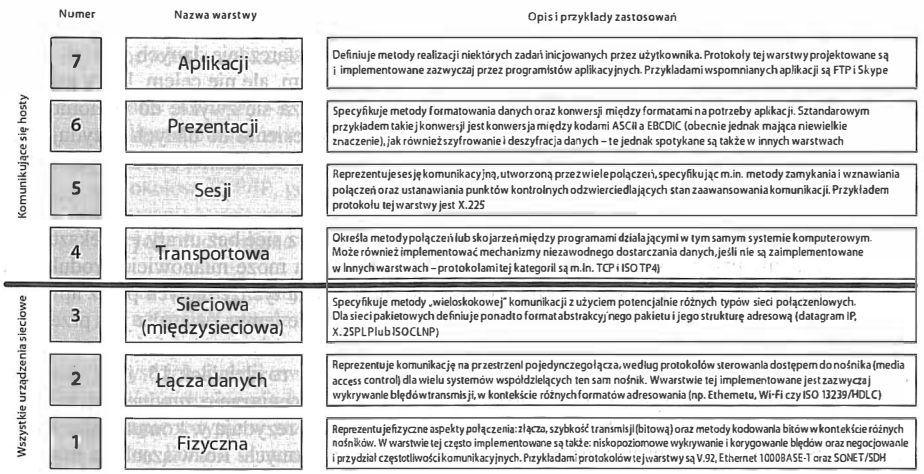
1.2. Projekt i implementacje

Mimo iż konkretna architektura protokołu może sugerować określone podejście do jego implementacji, zwykle jednak czyni się wyraźne rozróżnienie między architekturą protokołu a *architekturą implementacji* — czyli definicją sposobu urzeczywistnienia poszczególnych koncepcji, zwykle w postaci oprogramowania.

Wielu projektantów odpowiedzialnych za implementację protokołów dla sieci ARPANET to programiści systemowi, a „wpływowe" publikacje opisujące wieloprogramowy system operacyjny „THE" rekomendowały wykorzystywanie struktury hierarchicznej jako środka weryfikującego poprawność i solidność dużych implementacji programistycznych; wynikające stąd hierarchiczne podejście do projektowania i implementowania protokołów sieciowych ukształtowało w konsekwencji filozofię spojrzenia na te protokoły jak na *strukturę warstwową* (*layering*), przyjętą ostatecznie jako standard projektowania zestawów protokołów.

1.2.1. Architektura warstwowa

W architekturze warstwowej każda warstwa odpowiedzialna jest za inny aspekt komunikacji. Jest to zjawisko ze wszech miar korzystne, ponieważ umożliwia opracowywanie wielu elementów systemu niezależnie od siebie, często przez osobne zespoły charakteryzujące się doświadczeniem w różnych dziedzinach. Najbardziej znaną i najczęściej przywoływaną w publikacjach architekturą warstwową jest ta zdefiniowana przez organizację ISO (*International Organization for Standardization*) i nazywana powszechnie *modelem odniesienia OSI* (*Open System Interconnection Reference Model* — patrz [Z80]). Na rysunku 1.2 przedstawiamy układ jej warstw wraz z numerami, nazwami i opisem funkcjonalnym.



Rysunek 1.2. *Standardowy, siedmiowarstwowy model odniesienia OSI. W wielu urządzeniach implementowane są tylko niektóre warstwy, niemniej jednak powszechnie używane jest oryginalne numerowanie warstw i ich nazewnictwo*

Mimo iż model OSI sugeruje podział architektury implementacyjnej na siedem warstw, to — jak zobaczymy w podrozdziale 1.3 — architektura zestawu protokołów TCP/IP rozpatrywana jest w kontekście cokolwiek prostszym, bo pięciowarstwowym. Początkowo fakt ten stanowił źródło licznych kontrowersji podnoszących zalety (relatywne) oraz wady oryginalnego modelu OSI (i jego poprzednika — modelu ARPANET) i choć ostatecznie zwyciężył model TCP/IP, to trzeba uczciwie przyznać, że przeniknęło doń wiele idei (a nawet kompletnych protokołów, m.in. IS-IS [RFC3787]) z gruntu prac standaryzacyjnych ISO nad modelem OSI.

Z opisów widocznych na rysunku 1.2 wynika zróżnicowanie ról spełnianych przez poszczególne warstwy. W warstwie najniższej — *fizycznej* — definiowane są metody przesyłania informacji cyfrowej przez nośnik komunikacyjny, którym może być np. linia telefoniczna lub światłowód. Wśród tych definicji znajduje się część standardów Ethernet i Wi-Fi (w tej książce potraktujemy je raczej marginalnie). W kolejnej warstwie — *łącza danych* — plasują się protokoły i metody nawiązywania połączeń z sąsiadami współdzielącymi ten sam nośnik. W niektórych sieciach (m.in. DSL) nośnik łączy dokładnie dwóch sąsiadów, lecz ogólnie może być współdzielony przez wiele węzłów — jest tak w przypadku Ethernetu i Wi-Fi; w takich sieciach, zwanych *wielodostępnymi* (*multi access networks*), konieczne jest użycie protokołów sterujących dostępem węzłów do wspólnego nośnika — powrócimy do tego zagadnienia w rozdziale 3.

Kolejna w hierarchii warstwa *sieciowa* (zwana również *międzysieciową*) — najbardziej interesująca z perspektywy tematyki tej książki — jest w sieciach pakietowych (takich jak TCP/IP) miejscem definiowania uniwersalnego formatu pakietu, zapewniającego współdziałanie warstwy sieciowej z wieloma typami warstwy łącza danych. W warstwie sieciowej definiuje się także schemat adresowania hostów oraz algorytmy trasowania wyznaczające kolejne hosty na drodze wędrówki pakietu.

Powyżej warstwy sieciowej (3.) plasują się protokoły, które — przynajmniej w teorii — implementowane są jedynie w komunikujących się hostach. I tak warstwa *transportowa* zapewnia niezawodny transfer danych między sesjami, co może być zadaniem dość skomplikowanym, szczególnie w sieciach pakietowych mogących gubić pakiety. Pod pojęciem *sesji* rozumiemy rozciągniętą w czasie interakcję między aplikacjami; materialnym przykładem sesji między przeglądarką WWW a serwerem są obiekty *cookies*. Protokoły warstwy sesji mogą dostarczać zaawansowane mechanizmy, służące m.in. do inicjowania i wznawiania połączeń oraz utrwalania rezultatów dotychczasowej interakcji w postaci *punktu kontrolnego* (*checkpoint*) — możliwe jest wówczas wznawianie przerwanych sesji niekoniecznie od początku, lecz od jednego z dostępnych punktów kontrolnych. W warstwie *prezentacji* dokonuje się różnego rodzaju formatowanie danych oraz ich konwertowanie między formatami; jak wkrótce zobaczymy, protokoły internetowe nie definiują w sposób formalny warstw sesji i prezentacji, jeśli więc implementacja taka jest niezbędna z punktu widzenia współdziałających aplikacji, muszą one zatroszczyć się o nią we własnym zakresie.

Najwyższa warstwa — warstwa *aplikacji* — to zestaw różnorodnych protokołów specyficznych dla konkretnych aplikacji — protokołów tych jest obecnie multum i programiści wymyślają wciąż nowe. Sprawia to, że warstwa aplikacji jest z jednej strony najbardziej widoczna dla użytkownika, z drugiej natomiast stanowi największe pole do popisu dla pomysłowości w zakresie nowatorskich rozwiązań.

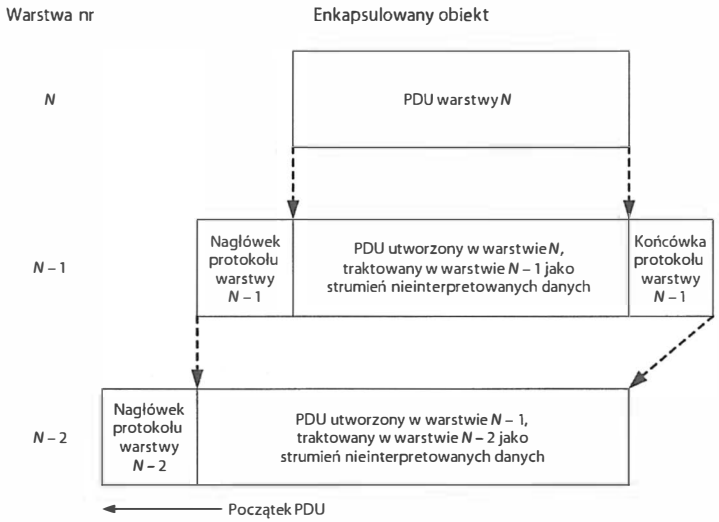
1.2.2. Multipleksowanie, demultipleksowanie i enkapsulacja w implementacjach warstwowych

Jedną z głównych zalet architektury warstwowej jest naturalna zdolność do realizacji *multipleksowania protokołów*. Multipleksowanie to technika umożliwiająca różnym protokołom współegzystowanie w ramach tej samej infrastruktury, pozwala ona także na uruchamianie wielu instancji danego obiektu związanego z protokołem (np. kilku równoczesnych połączeń) i ich funkcjonowanie bez wzajemnych interferencji.

Multipleksowanie może być realizowane w różnych warstwach, w każdej zaś warstwie *identyfikator* określonego rodzaju wykorzystywany jest do rozróżniania udziałów, jakie do multipleksowanej informacji wnoszą poszczególne protokoły; przykładowo większość technologii związanych z warstwą łącza danych (takich jak Ethernet i Wi-Fi) wykorzystuje pakiety posiadające pole identyfikujące protokół, na potrzeby którego transmitowane są ramki łącza (jednym z takich protokołów jest IP). Generalnie, gdy obiekt (pakiet, komunikat itp.) pewnej warstwy, zwany *jednostką danych protokołu* (PDU — *Protocol Data Unit*), transmitowany jest za pomocą mechanizmów warstwy niższej, mówimy o *hermetyzacji* lub *enkapsulacji* tegoż obiektu przez tę niższą warstwę. Istotą tej enkapsulacji jest traktowanie wspomnianego obiektu jako danych nieokreślonych, czyli niepodlegających interpretowaniu. Wynikiem enkapsulacji, jakiej w warstwie N–1 podlegają dane otrzymane od warstwy N, jest jednostka protokołu warstwy N–1, która z kolei enkapsulowana jest przez warstwę N–2. „Narzutem" wnoszonym przez każdy akt enkapsulacji jest identyfikator umożliwiający „rozhermetyzowanie" pakietu przez odnośną warstwę w ramach hosta docelowego.

Technikę tę wyjaśniamy poglądowo na rysunku 1.3. Każda warstwa charakteryzuje się własną koncepcją tworzonych przez siebie obiektów; przykładowo pakiet utworzony w warstwie 4. (transportowej) zwany jest jednostką danych protokołu transportowego lub krótko *transportowym* PDU (TPDU). Warstwa przejmująca PDU z warstwy wyższej „zobowiązuje się" niejako do traktowania tego PDU w sposób obojętny, czyli funkcjonowania w sposób abstrahujący od jego zawartości — owa „nieprzezroczystość" pakietu stanowi właśnie istotę enkapsulacji. Materialnym przejawem enkapsulowania pakietu jest poprzedzenie go odpowiednim nagłówkiem (*header*) i (niekiedy) dodanie odpowiedniej „końcówki" (*trailer*) — w protokołach grupy TCP/IP stosuje się wyłącznie nagłówki. Nagłówki te dodawane są więc w procesie multipleksowania protokołów i wykorzystywane w procesie odwrotnym — demultipleksowaniu — w oparciu o zawarte w nich identyfikatory. W sieciach TCP/IP identyfikatory te są najczęściej adresami sprzętowymi, adresami IP lub numerami portów. Nagłówki enkapsulujące mogą zawierać także dodatkowe informacje, np. parametry ustanowionego obwodu wirtualnego — tak czy inaczej poprzedzenie otrzymanego PDU stosownym nagłówkiem (i ewentualnie dodanie końcówki) daje w rezultacie nowe PDU warstwy niższej.

Z rysunku 1.2 wynika ponadto inna ważna cecha architektury warstwowej: nie w każdym urządzeniu sieciowym implementowane są wszystkie warstwy: jeżeli dane urządzenie spełniać ma niewielki zestaw funkcji, wystarczająca okazuje się implementacja jedynie wybranych warstw. Na rysunku 1.4 przedstawiamy (cokolwiek wyidealizowaną) sieć łączącą dwa systemy, zawierającą jeden przełącznik i jeden router. Liczby z lewej strony odpowiadają numerowaniu warstw w modelu odniesienia OSI, zaś literami oznaczono poszczególne implementacje protokołów. I tak w hoście z lewej strony rysunku widzimy

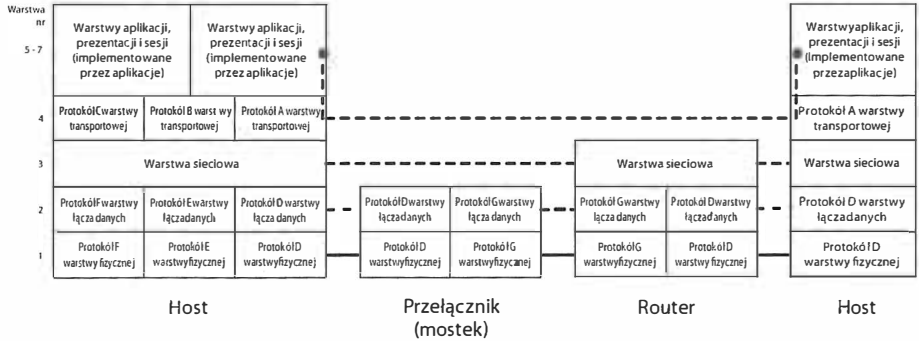


Rysunek 1.3. *Zazwyczaj enkapsulacja powiązana jest ściśle z architekturą warstwową: PDU utworzony w danej warstwie traktowany jest przez warstwę bezpośrednio niższą jako strumień nieinterpretowanych danych. W urządzaniu nadawczym strumień ten jest enkapsulowany, zaś utworzony w ten sposób PDU po dotarciu do urządzenia docelowego poddawany jest operacji odwrotnej* — dehermetyzacji *(*dekapsulacji*). W większości protokołów enkapsulowanie PDU polega na poprzedzeniu go odpowiednim nagłówkiem, niektóre protokoły dodają też specyficzną końcówkę*

trzy protokoły (D, E i F) w warstwie łącza danych (2.) i odpowiadające im protokoły w warstwie fizycznej (1.). Trzy różne protokoły (A, B i C) w warstwie transportowej (4.) współpracują z pojedynczym protokołem warstwy sieciowej (3.). W końcowych hostach implementowane są protokoły wszystkich warstw, podczas gdy przełącznik implementuje tylko dwie najniższe warstwy, zaś router — trzy najniższe. Aby router zdolny był do połączenia sieci działających w oparciu o różne mechanizmy warstwy łącza danych, musi implementować w warstwie 2. protokoły odpowiadające każdemu z tych mechanizmów.

Konfiguracja przedstawiona na rysunku 1.4 rozmija się nieco z rzeczywistością, bo współczesne routery i przełączniki oferują zazwyczaj funkcje wykraczające poza proste forwardowanie danych: funkcje te obejmują m.in. zdalne zarządzanie konfiguracją, które nierozłącznie wiąże się z realizacją zdalnego logowania. Wymaga to zaimplementowania protokołów warstwy transportowej i aplikacyjnej — i faktycznie wiele przełączników oraz routerów implementację taką posiada.

Na rysunku 1.4 przedstawiono tylko dwa komunikujące się hosty, jednak na sieciach reprezentowanych przez protokoły oznaczone D i G można posadowić dowolną liczbę hostów, z których każda para może komunikować się ze sobą, pod warunkiem odpowiedniego zaimplementowania protokołów warstw wyższych. Komunikujące się hosty nazywać będziemy *systemami końcowymi* (*end systems*), zaś urządzenia realizujące komunikację między nimi określać będziemy mianem *systemu pośredniczącego* (*intermediate system*) — na rysunku 1.4 system ten składa się z pojedynczego routera. Przełączniki i mostki



Rysunek 1.4. *Różne urządzenia sieciowe implementują odmienne podzbiory zestawu protokołów. W końcowych hostach implementowane są zazwyczaj wszystkie warstwy, routery zwykle implementują protokoły plasujące się poniżej warstwy transportowej, zaś w przełącznikach implementowane są jedynie dwie najniższe warstwy. Ten schemat jest jednak uproszczony o tyle, że routery i przełączniki często funkcjonować muszą jako hosty, by można było zdalnie zarządzać ich konfiguracjami; muszą więc implementować protokoły wszystkich warstw, nawet jeśli protokoły warstw wyższych wykorzystywane są sporadycznie*

nie są zwykle traktowane jako elementy systemów pośredniczących, ponieważ używane przez nie formaty adresów (i sposoby adresowania) różnią się wyraźnie od stosowanych w warstwie sieciowej i są w dużym stopniu „przezroczyste" zarówno dla tej warstwy (routerów), jak i komunikujących się systemów końcowych.

Warstwy najwyższe (te powyżej warstwy sieciowej) używają protokołów *end-to-end* (*end-to-end protocols*) — na rysunku 1.4 protokoły te wymagane są jedynie w systemach końcowych, jednakże w warstwie sieciowej implementowane są protokoły typu „skok po skoku" (*hop-by-hop*), które muszą być implementowane i w systemach końcowych, i w każdym systemie pośredniczącym.

Każdy router z definicji posiadać musi co najmniej dwa interfejsy sieciowe; system z większą liczbą interfejsów nazywany jest systemem *multihomed*. Systemem *multihomed* może być również „zwykły" host, o ile jednak nie realizuje on przekazywania pakietów między interfejsami, nie kwalifikujemy go jako routera. I vice versa: urządzenie funkcjonujące jako router niekoniecznie musi być prostym „pudełkiem" do forwardowania pakietów w Internecie — większość implementacji TCP/IP umożliwia pełnienie funkcji routera przez dowolny host *multihomed*, pod warunkiem jego właściwego skonfigurowania. Kwalifikacja takiego systemu zależna jest wówczas od kontekstu funkcjonowania: jeśli realizuje on konkretną aplikację (np. transfer plików zgodnie z protokołem FTP — patrz [RFC0959]), jest systemem końcowym (czyli po prostu „hostem"), jeśli natomiast wykonuje przekazywanie pakietów między sieciami, traktowany jest jako router.

Jednym z podstawowych celów projektowania Internetu jest ukrycie przed aplikacjami szczegółów zarówno topologii sieci, jak i niejednorodności (heterogeniczności) protokołów warstw niższych. Mimo iż nie wynika to w sposób oczywisty z rysunku 1.4, dla warstwy aplikacji jest zgoła obojętny fakt, że chociaż każdy host przyłączony jest do sieci przy użyciu protokołu D warstwy łącza danych (np. Ethernetu), komunikujące się hosty oddzielone są od siebie routerem i przełącznikiem, które wykorzystują protokół G. I gdyby na drodze między hostami pojawiło się dwadzieścia dodatkowych routerów, wy-

korzystujących inne jeszcze rodzaje połączeń fizycznych, aplikacje w systemach końcowych funkcjonowałyby tak, jak dotychczas, zupełnie obojętne na tę modyfikację (prawdę mówiąc, mogłaby się zmienić wydajność tego funkcjonowania). Abstrahowanie od szczegółów w taki oto sposób jest właśnie tym czynnikiem, dzięki któremu Internet jest tak funkcjonalny i użyteczny.

1.3. Architektura i protokoły zestawu TCP/IP

Dotychczas traktowaliśmy pojęcia architektury, protokołu, zestawu protokołów i implementacji w kategoriach abstrakcyjnych, w tym podrozdziale odniesiemy je natomiast do protokołów składających się na *zestaw protokołów TCP/IP* (*TCP/IP protocol suite*) — jakkolwiek zestaw ten jednoznacznie kojarzony jest z Internetem, to w Internecie znajduje zastosowanie wiele protokołów niewchodzących w jego skład. Rozpoczniemy ten podrozdział od przeanalizowania modelu odniesienia ARPANET, który stał się fundamentem dla konstrukcji protokołów TCP/IP i który różni się nieco od omawianego wcześniej modelu odniesienia OSI.

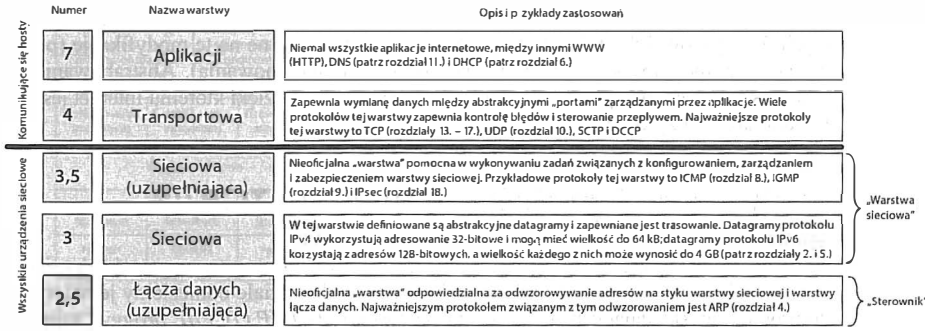
1.3.1. Model odniesienia ARPANET

Na rysunku 1.5 widoczny jest schemat warstwowy inspirowany modelem odniesienia ARPANET (oznaczanym skrótem ARM, od *Arpanet Reference Model*). Schemat ten jest prostszy od tego z modelu odniesienia OSI, obejmuje ponadto kilka specjalizowanych protokołów, które nie wpisują się w prosty sposób w żadną warstwę modelu OSI.

I tak na najniższym poziomie mamy „nieoficjalną" warstwę oznaczoną numerem 2,5. W warstwie tej funkcjonuje kilka protokołów, z których najstarszym i bodaj najważniejszym jest *ARP* (*Address Resolution Protocol* — protokół odwzorowania adresów). Zadaniem tego specjalizowanego protokołu, używanego przez protokół IP w wersji 4. (IPv4) w sieciach wielodostępnych (m.in. Ethernecie i Wi-Fi), jest konwersja między adresami IP a adresami używanymi w warstwie łącza danych. (Protokołem ARP zajmiemy się szczegółowo w rozdziale 4. W wersji 6. protokołu IP (IPv6) funkcja mapowania adresów jest częścią protokołu ICMPv6, którym zajmiemy się dokładniej w rozdziale 8.).

W warstwie 3. funkcjonuje protokół IP — podstawowy protokół sieciowy zestawu TCP/IP (opisujemy go z detalami w rozdziale 5.). Jednostki tego protokołu (PDU) wysyłane do warstwy łącza danych nazywane są *datagramami IP*; maksymalna wielkość takiego datagramu wynosi 64 kB (2^{16} bajtów) w wersji IPv4 i 4 GB (2^{32} bajtów) w wersji IPv6. W wielu przypadkach datagramy IP nazywane są po prostu *pakietami*, o ile w danym kontekście nie prowadzi to do nieporozumień. Podział dużych pakietów na mniejsze jednostki PDU warstwy łącza danych, zwane *ramkami* (*frames*), nosi nazwę *fragmentacji*; fragmentacja wykonywana jest przez hosty IP i (w razie potrzeby) przez niektóre routery. Poszczególne fragmenty pakietu, po dotarciu do celu, składane są z powrotem w jedną całość — proces ten nosi nazwę *reasemblacji* (fragmentację i reasemblację opisujemy dokładniej w rozdziale 10.).

W tej książce używać będziemy nazwy IP na określenie obu wersji protokołu: IPv4 oraz IPv6. Z perspektywy samej architektury protokołów różnice między tymi wersjami mają niewielkie znaczenie, okazują się natomiast istotne w kontekście adresowania i funkcji konfiguracyjnych (czym zajmiemy się szczegółowo w rozdziałach 2. i 6.).



Rysunek 1.5. *Warstwowa architektura modelu ARM i zestawu protokołów TCP/IP. W architekturze tej nie występują w sposób jawny warstwy sesji i prezentacji, ponadto wiele protokołów, wykonujących krytyczne nieraz funkcje pomocnicze na rzecz innych protokołów, nie wpisuje się w klasyczną strukturę warstwową modelu odniesienia OSI. Niektóre z tych protokołów — takie jak IGMP i ARP — nie są wykorzystywane w wersji IPv6*

Ponieważ pakiety IP są datagramami, każdy z nich zawiera adresy (w znaczeniu warstwy 3.) nadawcy i odbiorcy. Adresy te nazywane są *adresami IP*; w wersji IPv4 są one 32-bitowe, w wersji IPv6 — 128-bitowe (zajmiemy się nimi w rozdziale 2.). Adres odbiorcy zawarty w datagramie wykorzystywany jest przez router do określenia następnego routera na trasie datagramu; proces przesyłania datagramu między kolejnymi routerami („skokami") nazywamy *forwardowaniem*. Proces ten wykonywany jest głównie przez routery, choć mogą go realizować również hosty.

Rozróżniamy trzy typy adresów IP, zależnie od poszczególnych wariantów forwardowania. Adres typu *unicast* określa konkretny, pojedynczy host; adres typu *broadcast* reprezentuje wszystkie hosty danej sieci, natomiast adres *multicast* określa wszystkie hosty należące do pewnej grupy. Omówieniem wymienionych typów zajmiemy się w rozdziale 2.

Rolę pomocniczą w stosunku do protokołu IP pełni protokół *ICMP* (*Internet Control Message Protocol* internetowy protokół komunikatów kontrolnych), w modelu z rysunku 1.5 plasujący się w warstwie oznaczonej numerem 3,5. Jest on wykorzystywany do wymiany komunikatów o błędach i innych żywotnych informacji pomiędzy warstwami sieciowymi (3) komunikujących się hostów lub routerów. Analogicznie do dwóch wersji protokołu IP istnieją dwie wersje protokołu ICMP: ICMPv4 oraz ICMPv6; ta druga jest znacząco bardziej skomplikowana i obejmuje zaawansowane funkcje w rodzaju autokonfiguracji i odnajdywania sąsiednich routerów (*neighbor discovery*) — w sieciach IPv4 funkcje te spełniane były przez inne protokoły, m.in. ARP. Mimo iż protokół ICMP zaprojektowany został jako służebny w stosunku do IP, to możliwe jest jego bezpośrednie wykorzystywanie przez aplikacje, co czynią m.in. popularne narzędzia `ping` i `traceroute`. Komunikaty ICMP enkapsulowane są w datagramach IP tak samo jak jednostki protokołów warstwy transportowej.

Innym protokołem pomocniczym dla IPv4 jest *IGMP* (*Internet Group Management Protocol* — internetowy protokół zarządzania grupami), używany w kontekście adresów *multicast* do określenia, które z hostów należą do *grupy multicast*, czyli zainteresowane są komunikatami wysyłanymi pod określony adres. W rozdziale 9. zajmiemy się ogólnymi

właściwościami broadcastingu i multicastingu, a także protokołem IGMP i (wykorzystywanym przez IPv4) protokołem MLD (*Multicast Listener Discovery* — wykrywanie obserwatorów multicastu).

W warstwie 4. ulokowane są dwa najbardziej znane (i jednocześnie znacząco różne) protokoły transportowe zestawu TCP/IP. *Protokół sterowania transmisją* (TCP — *Transmission Control Protocol*) zapewnia radzenie sobie z takimi problemami transmisji jak utrata, powielanie i zmiana kolejności pakietów (o ile problemy te nie zostały naprawione w warstwie IP). Protokół ten funkcjonuje w oparciu o obwód wirtualny (VC) ustanawiany na bazie połączenia; nie zachowuje granic komunikatów (patrz rysunek 1.1). Dla odmiany, protokół *datagramów użytkownika* (UDP — *User Datagram Protocol*) oferuje niewiele ponad funkcjonalność udostępnianą przez IP. Za pomocą tego protokołu aplikacje mogą wysyłać datagramy z zachowaniem granic komunikatów, lecz bez możliwości kontrolowania błędów ani regulowania tempa wysyłania.

TCP zapewnia niezawodny przepływ danych między dwoma hostami przy użyciu takich mechanizmów jak dzielenie danych na porcje o wielkości odpowiedniej dla warstwy sieciowej (i scalanie tychże porcji), potwierdzanie otrzymywania pakietów i definiowanie limitów czasowych oczekiwania na owe potwierdzenia; odciąża to aplikacje od konieczności zajmowania się tymi detalami, pozwalając twórcom tych aplikacji skupić się na meritum ich funkcjonalności. Jednostka danych (PDU) protokołu TCP nazywana jest *segmentem TCP*.

Usługi świadczone dla aplikacji przez protokół UDP są daleko mniej zaawansowane: aplikacje mogą przesyłać między sobą datagramy, jednakże bez gwarancji, że wysyłany datagram w ogóle dotrze do miejsca przeznaczenia. Kontrola niezawodności i poprawności transmisji spada więc na barki samych aplikacji, protokół UDP ogranicza się w tym względzie do weryfikacji integralności danych za pomocą sum kontrolnych (*checksums*) oraz numerów portów używanych w procesie multipleksowania i demultipleksowania. Tak oto egzystują we wspólnej warstwie dwa drastycznie różniące się protokoły, choć — oczywiście — każdy z nich okazuje się lepszy od konkurenta w konkretnym zastosowaniu (niebawem rozwiniemy tę kwestię).

Oprócz protokołów TCP i UDP w warstwie transportowej istnieją jeszcze dwa godne uwagi protokoły — stosunkowo nowe i spotykane tylko w niektórych systemach. Pierwszy z nich — *DCCP* (*Datagram Congestion Control Protocol* — protokół kontroli przeciążenia [transmisji] datagramów) — opisany w dokumencie [RFC4340] zapewnia usługi pośrednie między TCP i UDP: datagramy wymieniane są w ramach połączenia, bez weryfikacji poprawności, lecz z kontrolowaniem przeciążeń, czyli ograniczaniem produktywności nadawcy stosownie do bieżącej przepustowości sieci. Zajmiemy się tą kwestią w rozdziale 16., przy okazji szczegółowego omawiania protokołu TCP.

Drugi ze wspomnianych protokołów — *SCTP* (*Stream Control Transmission Protocol* — protokół sterowania transmisją strumieni), zdefiniowany w dokumencie [RFC4960], zapewnia niezawodną transmisję na wzór TCP, jednakże bez konieczności ścisłego zachowywania sekwencyjności danych: w ramach tego samego połączenia możliwe jest równoległe przesyłanie kilku logicznych strumieni danych. Protokół ten zaprojektowany został na potrzeby przesyłania komunikatów sygnalizacyjnych w sieciach IP, analogicznie do sieci telefonicznych.

Dzięki usługom oferowanym przez warstwę transportową, zlokalizowana najwyżej w hierarchii warstwa aplikacji może ograniczyć się do merytorycznych aspektów samych aplikacji, w oderwaniu od sposobu przesyłania wymienianych danych przez sieć. I vice versa: trzy najniższe warstwy zajmują się wyłącznie detalami komunikacji, w oderwaniu od wymienianych treści.

1.3.2. Multipleksowanie, demultipleksowanie i enkapsulacja w protokołach TCP/IP

Jak już wyjaśnialiśmy, w procesie multipleksowania i demultipleksowania istotną rolę odgrywają identyfikatory protokołów działających w poszczególnych warstwach; zazwyczaj każda warstwa umieszcza w tworzonym PDU informacje adresowe, wykorzystywane później do zapewnienia, że owo PDU zostało dostarczone do właściwego miejsca przeznaczenia.

Na rysunku 1.6 pokazujemy schematycznie, jak przebiegać może demultipleksowanie wewnątrz hipotetycznego hosta. Rozpocznijmy od warstwy łącza danych (tej klasycznej, opatrzonej numerem 2 w modelu OSI; oczywiście, nie ma jej w schemacie TCP/IP) i załóżmy, że do hosta dociera ramka Ethernetu. Zawiera ona 48-bitowy adres docelowy (zwany potocznie *adresem MAC*) i 16-bitowe pole o nazwie *Ethernet Type*, identyfikujące typ przenoszonego przez ramkę datagramu — wartość 0x0800 w tym polu oznacza, że mamy do czynienia z datagramem IPv4, wartości 0x0806 i 0x86DD oznaczają datagramy protokołów (odpowiednio) ARP i IPv6. Przy założeniu, że zawarty w ramce adres docelowy istotnie jest adresem MAC naszego hosta i ramka została zweryfikowana jako wolna od błędów, zostaje ona pozbawiona nagłówka i końcówki wnoszonej przez Ethernet, a otrzymany w ten sposób *ładunek użyteczny* (*payload*) przekazany zostaje do przetworzenia przez protokół reprezentowany we wspomnianym polu *Ethernet Type*.

Jeżeli pole to zawiera wartość[1] 0x0800 lub 0x86DD, czyli ładunek użyteczny jest pakietem IP, protokół IP dokonuje sprawdzenia szeregu zawartych w nim informacji, m.in. adresu docelowego. Jeśli ten jest identyczny z adresem IP naszego hosta i nagłówek pakietu jest bezbłędny (IP nie weryfikuje zawartości ładunku użytecznego pakietu), następuje sprawdzenie 8-bitowego pola *Protokół* (w wersji IPv6 pole to nosi nazwę *Następny nagłówek*) w celu określenia, jakiego typu pakiet transportowy enkapsulowany jest w pakiecie IP. Najczęściej spotykanymi w tym polu wartościami są:

1 — identyfikująca pakiet ICMP,

2 — identyfikująca pakiet IGMP,

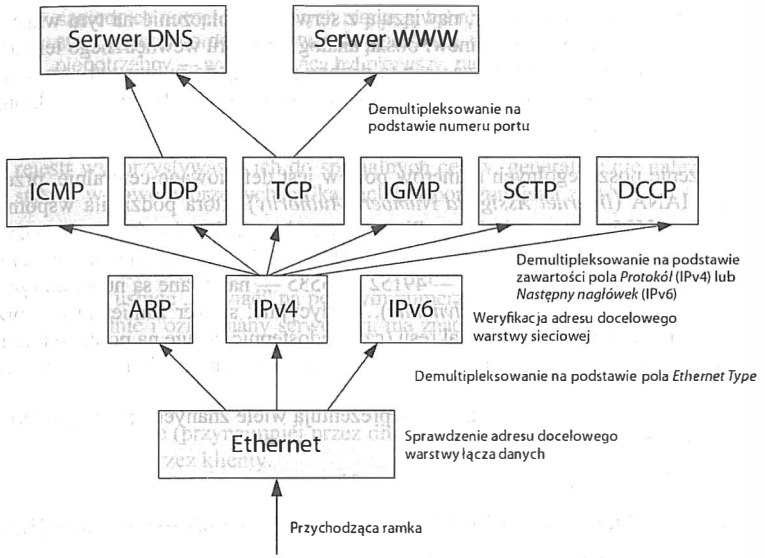
4 — identyfikująca pakiet IPv4,

41 — identyfikująca pakiet IPv6,

6 — identyfikująca segment TCP,

17 — identyfikująca datagram UDP.

[1] Przedrostek 0x oznacza liczbę w zapisie szesnastkowym — *przyp. tłum.*



Rysunek 1.6. *W zestawie protokołów TCP/IP wykorzystywana jest kombinacja identyfikatorów protokołów i informacji adresowych w celu weryfikacji bezbłędnego dostarczenia datagramu i przekazania go właściwemu protokołowi do dalszej obróbki. W niektórych warstwach stosowane są sumy kontrolne służące do weryfikowania integralności przesyłanych treści*

Wartości (cokolwiek zagadkowe) 4 i 41 oznaczają, że w pakiecie IP enkapsulowany jest inny pakiet IP; jakkolwiek stanowi to odstępstwo od ścisłej hierarchii architektury warstwowej (wynikowe PDU należy mianowicie do tej samej warstwy, co jego ładunek użyteczny) jest podstawą niezwykle użytecznej techniki zwanej *tunelowaniem* (którą szczegółowo omawiamy w rozdziale 3.).

Gdy warstwa sieciowa dokona pozytywnej weryfikacji otrzymanego datagramu i pomyślnego określenia protokołu warstwy transportowej, datagram ten (po ewentualnej reasemblacji, czyli scaleniu poszczególnych fragmentów) przekazywany jest do wspomnianego protokołu.

Większość protokołów warstwy transportowej (m.in. TCP i UDP) wykorzystuje *numery portów* do identyfikowania aplikacji, którym należy przekazywać ładunek użyteczny demultipleksowanych pakietów.

1.3.3. Numery portów

Numer portu to nic innego jak liczba całkowita z zakresu 0 – 65535; słowo „port" ma tu znaczenie abstrakcyjne i nie odnosi się do żadnego fizycznego obiektu. Liczba ta towarzyszy adresom IP w kontekście większości protokołów transportowych, a jej zadaniem jest rozróżnianie pomiędzy wieloma aplikacjami działającymi w tym samym hoście. Przykładowo w aplikacjach typu klient-serwer (piszemy o nich w punkcie 1.5.1) aplikacja serwera najpierw „przywiązuje się" do określonego numeru portu, po czym aplikacje

klienckie, znające ów numer, nawiązują z serwerem połączenie na tym właśnie porcie. W tym ujęciu numer portu stanowi bliską analogię numeru wewnętrznego telefonu w sieci telefonicznej przedsiębiorstwa — z tą drobną różnicą, że numer ten przydzielany jest w sposób sformalizowany, niejako poza kompetencjami przedsiębiorstwa, które stanowi tu — oczywiście — analogię serwera.

Znaczenie poszczególnych numerów portów jest definiowane centralnie, przez organizację IANA (*Internet Assigned Numbers Authority*), która podzieliła wspomniany zakres 0 – 65535 na trzy podzakresy. Pierwszy z tych podzakresów, 0 – 1023, to tzw. numery *dobrze określone* (*well known*); drugi — 1024 – 49151 — obejmuje numery *rejestrowane* (*registered*), pozostałe numery — 49152 – 65535 — nazywane są numerami *prywatnymi* (*private*) lub *dynamicznymi* (*dynamic*). Tradycyjnie, serwer zamierzający „przywiązać się" do numeru z pierwszego zakresu (czyli udostępnić usługę na porcie o tym numerze) może to zrobić tylko w kontekście przywileju administratora lub użytkownika „root".

Numery z tego pierwszego zakresu reprezentują wiele znanych aplikacji internetowych, że wymienimy tylko niektóre:

- SSH (*Secure Shell Protocol*) — port 22.,
- Protokół transferu plików FTP (*File Transfer Protocol*) — porty 20. i 21.,
- Protokół zdalnego terminala Telnet — port 23.,
- Protokół przesyłania poczty elektronicznej SMTP (*Simple Mail Transfer Protocol*) — port 25.,
- System DNS — port 53.,
- Protokoły hipertekstowe HTTP i HTTPS — porty (odpowiednio) 80. i 443.,
- Protokoły interaktywnego dostępu do poczty IMAP (*Interactive Mail Access Protocol*) i IMAPS — porty (odpowiednio) 143. i 993.,
- Protokół „urzędu pocztowego" POP3 (*Post Office Protocol*) — port 110.,
- Protokół zarządzania siecią SNMP (*Simple Network Management Protocol*) — porty 161. i 162.,
- „Lekki protokół usług katalogowych" LDAP (*Lightweight Directory Access Protocol*) — port 389.

Aplikacje posługujące się kilkoma portami wykorzystują każdy z nich do innego celu — przykładem jest protokół FTP, wykorzystujący oddzielne porty do przesyłania zawartości plików i przesyłania informacji sterujących. Z kolei wybór między protokołem HTTP i HTTPS zależny jest od tego, czy w protokole aplikacji wykorzystywany jest mechanizm TLS (*Transport Layer Security* — zabezpieczenie warstwy transportowej, patrz rozdział 18.).

Należy zauważyć, że większość numerów portów używanych przez najpopularniejsze aplikacje (Telnet, FTP, SMTP itd.) to numery nieparzyste. Nie jest to przypadek, lecz historyczne dziedzictwo protokołu NCP (*Network Control Protocol* — protokół sterowania siecią), wywodzącego się z sieci ARPANET protoplasty protokołu TCP. Był to protokół jednokierunkowy (simpleksowy), każda aplikacja wymagała więc do swego działania *dwóch* numerów portów, po jednym dla każdego kierunku; przydzielano jej zatem parę

sąsiednich numerów, z których pierwszy był nieparzysty. Gdy protokoły TCP i UDP stały się standardem warstwy transportowej, drugi z tych numerów okazywał się być niepotrzebny — wystarczający był pierwszy, nieparzysty.

Zarejestrowane numery portów, czyli te z przedziału 1024 – 49151, dostępne są dla klientów i serwerów ze specjalnymi przywilejami, ponieważ jednak IANA prowadzi rejestr wykorzystywania ich do specjalnych celów, generalnie nie należy ich wykorzystywać w nowo tworzonych aplikacjach bez porozumienia z IANA.

Numery dynamiczne portów nie podlegają zasadniczo żadnej regulacji, bo są używane jako tymczasowe („ulotne" — *ephemeral*). Aplikacja kliencka korzystająca z usługi serwera musi tę usługę „przyjąć" na pewnym numerze portu; numer ten wybierany jest przez nią arbitralnie i oznajmiany serwerowi, ma znaczenie jedynie dla komunikującej się pary aplikacji i odchodzi w niebyt po zakończeniu połączenia. Wybór portu przez aplikację kliencką może być dowolny, bo to ona odszukuje żądaną usługę serwerową, a nie odwrotnie. Dla odmiany, numery portów używane przez aplikacje serwerowe muszą pozostawać niezmienne (przynajmniej przez dłuższy okres), by aplikacje te mogły być w ogóle odnajdywane przez klienty.

1.3.4. Nazwy, adresy i usługa DNS

W zestawie protokołów TCP/IP każdy interfejs warstwy łącza danych w każdym komputerze (również w routerach) skojarzony jest z przynajmniej jednym adresem IP. Adresy IP są całkowicie wystarczające do identyfikowania hostów przez oprogramowanie, z ludzkiego punktu widzenia są jednak niewygodne do zapamiętywania i niezbyt poręczne w zarządzaniu zwłaszcza wtedy, gdy są to „długie" adresy protokołu IPv6. W związku z tym w świecie TCP/IP adresom IP przyporządkowuje się (wygodniejsze w użyciu) *nazwy symboliczne*, a szczegóły tego przyporządkowania przechowywane są w rozproszonej bazie DNS (*Domain Name System* — zajmiemy się tym szczegółowo w rozdziale 11.). Wspomniane nazwy zorganizowane są hierarchicznie, w podziale na *domeny*; ostatnim członem każdej nazwy jest domena najwyższego poziomu, przykładowo `.com`, `.org`, `.gov`, `.in`, `.edu`, `.uk` czy `.pl`. Co ciekawe, DNS jest protokołem warstwy aplikacji, jego funkcjonowanie uzależnione jest więc od wielu protokołów warstw niższych. Mimo iż dla większości protokołów TCP/IP nazwy DNS są nieistotne, to dla typowego użytkownika Internetu mają one znaczenie pierwszorzędne, ponieważ zapamiętuje on np. witryny WWW przez ich nazwy symboliczne, nie znając w ogóle ich adresów IP, zatem niedostępność DNS oznacza dla niego praktycznie niemożność korzystania z Internetu.

Aplikacje operujące nazwami DNS mają do dyspozycji standardową funkcję API „przeliczającą" podaną nazwę symboliczną hosta na jego adres IP (patrz punkt 1.5.3), istnieje także funkcja wykonująca poszukiwania w kierunku odwrotnym. Wiele aplikacji dopuszcza komunikację z użytkownikami w obu kategoriach — adresów IP i nazw DNS: koronnym przykładem są tu przeglądarki WWW, dopuszczające lokalizatory URL (*Uniform Resource Locators*) zarówno w postaci typu `http://131.243.2.201/index.html` lub `http://[2001:400:610:102::c9]/index.html`, jak i w postaci `http://ee.lbl.gov/index.html` (gdy piszemy te słowa, wszystkie trzy adresy identyfikują ten sam host, drugi z podanych adresów do nawiązania połączenia wymaga protokołu IPv6).

1.4. Internety, intranety i ekstranety

Jak już wyjaśnialiśmy, łącząc ze sobą dwie lub więcej sieci przy użyciu zestawu protokołów TCP/IP, otrzymujemy twór zwany międzysiecią lub (z angielska) internetem — pisanym z małej litery, jako rzeczownik pospolity. Można więc samodzielnie stworzyć sobie mały internet, łącząc za pomocą routera dwie sieci lokalne. Przeciętny użytkownik komputera, słysząc słowo „internet", wyobraża sobie jednak coś zupełnie innego — ową globalną „sieć sieci", łączącą ze sobą miliardy hostów rozproszonych po całym świecie. Ów gigantyczny, globalny internet wyróżniać będziemy spośród innych internetów i pisać przez duże „I"[2].

Jednym z powodów fenomenalnego rozwoju sieci komputerowych w latach 80. ubiegłego wieku był oczywisty poniekąd fakt, że użyteczność komputera znacznie wzrasta, gdy połączy się go z innymi komputerami. Niespełna dekadę później ta sama idea przeniesiona została na wyższy poziom: spostrzeżenie, że połączenie ze sobą wielu sieci komputerowych wnosi nową wartość, stało się podstawą prawa Metcalfe'a, zgodnie z którym użyteczność sieci komputerowej rośnie proporcjonalnie do pierwiastka kwadratowego z liczby jej węzłów — użytkowników i urządzeń. Dzięki idei Internetu i wspierających tę ideę protokołów stało się możliwe łączenie sieci o różnej architekturze; prosta pozornie koncepcja okazała się mieć ogromne konsekwencje.

Najprostszym sposobem zbudowania internetu jest połączenie dwóch (lub więcej) sieci za pomocą specjalizowanego urządzenia zwanego *routerem IP*. Wspaniałą cechą routerów jest możliwość łączenia sieci o różnej strukturze fizycznej — Ethernetu, Wi-Fi, połączeń „punkt-punkt", DSL, Internetu kablowego itp.

Routery IP, nazywane po prostu „routerami", niegdyś określane były jako *bramy* lub *śluzy* (*gateways*) i określenia te wciąż spotyka się w literaturze przedmiotu. Obecnie jednak termin „brama" zarezerwowany jest dla tzw. *bram aplikacyjnych* (ALG — *Application Layer Gateways*), czyli procesów zapewniających współdziałanie dwóch różnych zestawów protokołów (np. TCP/IP oraz IBM-owskiej architektury SNA[3]) na potrzeby określonej aplikacji (najczęściej poczty elektronicznej lub transferu plików).

Na przestrzeni ostatnich dwudziestu lat pojawiły się także różne terminy na określenie różnych konfiguracji połączeń sieci za pomocą protokołów grupy TCP/IP. I tak prywatną międzysieć, działającą zwykle na potrzeby firmy lub organizacji, nazywa się popularnie *intranetem*. Zasoby intranetu udostępniane są tylko określonym osobom (zazwyczaj pracownikom firmy), a dostęp do tych zasobów odbywa się za pośrednictwem *prywatnych sieci wirtualnych* (VPN — *Virtual Private Networks*) wykorzystujących mechanizm tunelowania (o którym wspomnieliśmy w punkcie 1.3.2). (Sieciom VPN poświęcamy rozdział 7.).

Niekiedy przedsiębiorstwa decydują się na udostępnienie swych zasobów wybranym partnerom (lub innym zaufanym podmiotom) za pośrednictwem Internetu. Udostępnienie to odbywa się również przy użyciu mechanizmów VPN wspomaganych firmowym fi-

[2] Przymiotnik „internetowy" piszemy jednak w obu znaczeniach z małej litery, zgodnie z regułami języka polskiego — *przyp. tłum.*

[3] Patrz np. *http://pl.wikipedia.org/wiki/Systems_Network_Architecture — przyp. tłum.*

rewallem (patrz rozdział 7.), a udostępnianą sieć nazywa się *ekstranetem*. Technicznie ekstranet niewiele różni się od intranetu, znacząco różnią się natomiast przypadki ich użycia i polityka administracyjna.

1.5. Projektowanie aplikacji

W świetle opisanych dotychczas koncepcji sieć komputerowa (w szczególności: internet) jawi się nam jako maszyna do przesyłania bajtów między komputerami (i niekiedy wewnątrz komputera) i chociaż nie sposób zaprzeczyć takiemu spojrzeniu (patrz dokument [RFC6250]), to jednak prawdziwa wartość sieci objawia się wówczas, gdy ów prosty model usługi zaprzęgnięty zostanie do wykonywania użytecznych zadań. Rolę tę spełniają aplikacje sieciowe; ich struktura da się zazwyczaj sprowadzić do jednego z kilku wzorców projektowych — wzorcami najczęściej wykorzystywanymi są *klient-serwer* i *peer-to-peer*.

1.5.1. Architektura klient-serwer

Architektura klient-serwer to dwa współdziałające elementy aplikacji: element zwany *serwerem*, świadczący określone usługi (np. udostępnianie plików zmagazynowanych na dyskach hosta), i element zwany *klientem*, będący konsumentem tychże usług. Zależnie od sposobu udostępniania usług, serwery możemy podzielić na dwie kategorie.

Serwer *iteratywny* to realizacja następującego prostego scenariusza:

I1. Oczekiwanie na nadejście żądania ze strony klienta.

I2. Przetwarzanie otrzymanego żądania.

I3. Odesłanie klientowi odpowiedzi na żądanie.

I4. Powrót do kroku I1.

Podstawowy mankament powyższego scenariusza wynika z faktu, że realizacja kroku I2 może trwać bardzo długo i może upłynąć dużo czasu, zanim sterowanie powróci do kroku I1 i serwer będzie mógł przyjąć kolejne żądanie. Z perspektywy klienta sytuacja taka oznacza po prostu długotrwałą niedostępność serwera.

Wolny jest od tej przypadłości serwer współbieżny, działający zgodnie z następującym scenariuszem:

C1. Oczekiwanie przez główną instancję serwera na nadejście żądania ze strony klienta.

C2. Uruchomienie potomnej instancji serwera, z zadaniem obsłużenia otrzymanego żądania. Instancja główna nie interesuje się odtąd losami uruchomionej instancji potomnej, która po wykonaniu swego zadania samorzutnie przestanie istnieć; gdy instancja potomna rozpocznie swą pracę, instancja główna przechodzi do kroku C3.

C3. Powrót do kroku C1.

Uruchamiane instancje serwera mogą być procesami, zadaniami, wątkami itp., zależnie od mechanizmów tkwiących w systemie operacyjnym. Oczywiście, musi to być wielozadaniowy system operacyjny, co dla współczesnych systemów jest standardem — i z tej to przyczyny większość obecnych serwerów to serwery współbieżne.

W przeciwieństwie do serwera iteratywnego, realizacja żądania klienckiego przez serwer współbieżny nie wiąże się z czasową utratą dostępności tegoż dla klientów: instancja główna aktywnie nasłuchuje żądań w kroku C1, z jedynie minimalnymi przerwami potrzebnymi na powołanie do życia instancji potomnej w kroku C2. W rezultacie każdemu żądaniu przydzielona zostaje osobna instancja serwera, co z punktu widzenia klienta jest raczej nieistotne — realizacja odebranego żądania odbywa się identycznie w serwerach obu kategorii, dlatego też kategoryzowaliśmy serwery, a nie klienty.

Zauważmy przy tym, że mówiąc o „kliencie" i „serwerze" mieliśmy na myśli encję *programową* (zadanie, proces, wątek), a nie konkretne obiekty sprzętowe, na których encje te są uruchamiane. Trzeba jednak przyznać, że terminologia w tym względzie jest nieprecyzyjna i w niektórych kontekstach elementy aplikacji klient-serwer faktycznie postrzegane są w kategoriach konkretnych komputerów. Tak czy inaczej, zawsze można traktować serwer w sensie sprzętowym jako urządzenie realizujące jeden lub więcej serwerów w sensie programowym.

1.5.2. Architektura peer-to-peer

Aplikacje sieciowe można także tworzyć pod kątem konfiguracji rozproszonej, w której nie istnieje pojedynczy, wyróżniony serwer, lecz każda z aplikacji może pełnić rolę zarówno klienta, jak i serwera — często równocześnie, funkcjonując jako przekaźnik żądań. Tak właśnie działa wiele popularnych aplikacji, m.in. Skype ([SKYPE]) i BitTorrent ([BT]), co określane jest mianem „każdy z każdym", po angielsku *peer-to-peer*, w skrócie *p2p*. Aplikacja p2p po odebraniu żądania może decydować o tym, czy obsłużyć to żądanie samodzielnie, czy też przekazać je do innej, partnerskiej aplikacji. Tak oto konstytuuje się *sieć aplikacji*, zwana także *siecią nakładkową* (*overlay network*). Sieci takie są obecnie bardzo popularne: jeśli wierzyć wynikom badań statystycznych, Skype okazuje się największym przekaźnikiem w międzynarodowej komunikacji telefonicznej, a BitTorrent odpowiedzialny jest za generowanie ponad połowy całego ruchu w Internecie (dane z roku 2009 — patrz [IPIS]).

Podstawowym problemem sieci p2p jest *odnajdywanie* aplikacji partnerskich: w jaki mianowicie sposób konkretna aplikacja znaleźć ma inną aplikację, która dostarczy jej niezbędne dane? Problem ten rozwiązuje się zwykle za pomocą odpowiedniej procedury rozruchowej (*bootstrapping*), w ramach której każdy klient informowany jest o adresach i numerach portów niektórych, prawdopodobnie aktywnych, aplikacji. Po nawiązaniu połączenia z tymi aplikacjami uzyskuje on dodatkowe informacje, zależnie od konkretnego protokołu i rodzaju żądanych (oferowanych) danych.

1.5.3. Interfejsy programisty (API)

Aplikacja sieciowa — niezależnie od typu — zamierzająca skorzystać z usługi sieciowej musi jakoś swój zamiar wyrazić oraz odebrać odpowiedź zwrotną na swe żądanie. Funkcję tę zazwyczaj spełnia system operacyjny hosta za pośrednictwem odpowiedniego *interfejsu*

programisty (API — *Appllication Programmer Interface*). Do najbardziej znanych API usług sieciowych należy niewątpliwie *interfejs gniazd* (*sockets interface*), ze względu na swą genealogię (patrz [LJFK93]) często poprzedzany przymiotnikiem „berkeleyowski" (*Berkeley sockets interface*).

Książka ta nie jest podręcznikiem programowania, gdzieniegdzie będziemy jednak odwoływać się do interfejsu gniazd, w kontekście zaimplementowania (bądź niezaimplementowania) w nim poszczególnych mechanizmów związanych z protokołami TCP/IP. Czytelnicy bliżej zainteresowani programistycznymi aspektami tej grupy protokołów znajdą wyczerpujące omówienie tego tematu, wraz z licznymi przykładami, w książce [SFR04]. Niezbędne modyfikacje interfejsu gniazd związane z przystosowywaniem go do wykorzystywania w kontekście protokołu IPv6 opisane są w szeregu ogólnodostępnych online dokumentów RFC, m.in. [RFC3493], [RFC3542], [RFC3678], [RFC4584] oraz [RFC5014].

1.6. Procesy standaryzacyjne

Gdy weźmie się pod uwagę stopień złożoności zestawu protokołów TCP/IP, nie sposób nie zadać sobie pytania, jak udało się tę złożoność okiełznać i jaka jest strategia jej kontrolowania w związku z naturalnym rozwojem wspomnianych protokołów. Odpowiedź na to pytanie kryje się pod enigmatycznym określeniem „standaryzacja" i — w prostym przełożeniu — pod organizacjami, które są za tę standaryzację odpowiedzialne. Wśród tych organizacji na pierwszym miejscu wymienić należy IETF (*Internet Engineering Task Force* — patrz [RFC4677]). Przedmiotem odbywających się trzy razy do roku (w różnych częściach świata) zebrań plenarnych tej organizacji są propozycje, dyskusje i uzgodnienia dotyczące „rdzennych" (*core*) protokołów internetowych; sama kwestia uznania tego czy innego protokołu za „rdzenny" również bywa przedmiotem dyskusji, niewątpliwie jednak status taki posiadają protokoły IPv4, IPv6, TCP, UDP i DNS.

IETF jest organizacją otwartą dla wszystkich chętnych — zainteresowana osoba może zapisać się do którejś z grup roboczych i zabierać głos online; nie istnieje instytucja formalnego członkostwa (ani składek), członkowie pokrywają jedynie koszty swego uczestnictwa w zebraniach. IETF nie posiada formalnej władzy; nadzór nad jej działalnością sprawują grupy kierownicze IESG (*Internet Engineering Steering Group*) i IAB (*Internet Architecture Board*). IESG to grupa ekspertów, których zadaniem jest techniczna weryfikacja nowo opracowywanych standardów oraz modyfikacji wprowadzanych do standardów istniejących; w gestii IAB spoczywa obserwowanie pracy IETF jako całości, jak również wyznaczanie łączników do kontaktów z innymi organizacjami standaryzacyjnymi (SDO — *standards-defining organizations*). Konkretne, szczegółowe prace prowadzone są w grupach roboczych (*working groups*), których stanowiska komunikowane są IESG (w celu zaopiniowania i rozstrzygania ewentualnych sporów).

Z IETF współpracują ściśle dwie inne organizacje. IRTF (*Internet Research Task Force*) zajmuje się eksplorowaniem protokołów, architektur i procedur, które wydają się jeszcze niedojrzałe do standaryzacji. Przewodniczący IRTF wyznaczany jest przez IAB. Z kolei IAB ściśle współpracuje ze stowarzyszeniem ISOC (*Internet Society*) w celu promowania na całym świecie polityki i technologii związanych z Internetem.

1.6.1. Dokumenty RFC (Request for Comments)

Każdy oficjalny standard wypracowany przez społeczność Internetu publikowany jest w formie dokumentu RFC (*Request for Comments* — dosł. „zapotrzebowanie na komentarze"). Każdy nowy dokument, opatrzony unikalnym numerem (nowsze dokumenty mają większe numery), publikowany jest w witrynie *RFC Editor* (*http://www.rfc-editor.org*); po opublikowaniu (permanentnym) nie jest już zmieniany, dostrzeżone błędy korygowane są w publikowanej erracie. Gdy dokument RFC staje się nieaktualny, zostaje formalnie unieważniony przez inny dokument RFC. Dokumenty RFC tworzone są zwykle przez pojedynczych autorów lub niewielkie grupy autorskie o odpowiednim doświadczeniu; autorzy ci wywodzą się z różnych kręgów („strumieni"), głównie z IETF, IAB i IRTF, lecz także z wielu innych środowisk. Przed permanentnym opublikowaniem dokument RFC ma początkowo status tzw. szkicu (*Internet drafi*) służącego do zbierania opinii i komentarzy pomocnych w ustaleniu ostatecznej treści.

Nie wszystkie dokumenty RFC stanowią publikacje standardów — tę cechę mają jedynie dokumenty zaklasyfikowane jako „ścieżka standardów" (*standard track*); zależnie od treści (i intencji autorów) dokumenty mogą być klasyfikowane w innych kategoriach, m.in. w kategorii *sprawdzonych praktyk* (BCP — *best current practice*), *informacyjnej*, *eksperymentalnej* i *historycznej*. Istnieją też dokumenty RFC, co do treści których panują skrajnie sprzeczne opinie.

Rozmiary dokumentów także bywają zróżnicowane — od stron kilku do kilkuset. Tradycyjnie dokument RFC ma format czystego tekstu, choć niektóre dostępne są także w formatach bardziej zaawansowanych.

Kilka dokumentów RFC posiada specjalne znaczenie z tego względu, że służą podsumowywaniu, objaśnianiu lub interpretowaniu innych standardów i dokumentów. Należy do nich m.in. [RFC5000] opublikowany w maju 2008 roku, będący de facto spisem treści ówcześnie obowiązujących standardów (aktualną listę standardów internetowych znaleźć można w witrynie [OIPSW]). Z kolei dokumenty [RFC1122] i [RFC1123] zawierają opis wymagań stawianych hostom realizującym implementacje protokołów internetowych, [RFC1812] zawiera analogiczną listę wymagań dla routerów IPv4. Dla wersji IPv6 obie grupy wymagań zebrane zostały w dokumencie [RFC4294].

1.6.2. Inne standardy

Mimo iż za standaryzację większości protokołów internetowych odpowiedzialne jest IETF, kilka innych organizacji ma także znaczący udział w tym dziele. W tej grupie wymienić należy przede wszystkim IEEE (*Institute of Electrical and Electronics Engineers*), W3C (*World Wide Web Consortium*) i ITU (*International Telecommunication Union*). W kontekście opisywanych w tym rozdziale modeli warstwowych IEEE zajmuje się standardami związanymi z warstwami ulokowanymi poniżej warstwy nr 3 (m.in. Ethernetem i Wi-Fi), natomiast konsorcjum W3C odpowiedzialne jest za protokoły warstwy aplikacyjnej, szczególnie związane z technologiami webowymi (m.in. protokoły bazujące na składni http). Z kolei prace ITU (a konkretnie: ITU-T, dawniej CCITT) koncentrują się wokół sieci telefonicznych i komórkowych, stanowiących obecnie istotny komponent Internetu.

1.7. Implementacje TCP/IP i ich dystrybucja

Implementacje standardu de facto TCP/IP wywodzą swą genealogię z Uniwersytetu Kalifornijskiego w Berkeley, a konkretnie grupy CSRG (*Computer Systems Research Group*). Pierwsza z tych implementacji rozprowadzana była w latach 90. ubiegłego wieku wraz z systemem BSD 4.x (BSD to skrót od *Berkeley Software Distribution* — dystrybucja oprogramowania berkeleyowskiego) oraz oprogramowaniem BSD Networking Releases. Kod źródłowy tych implementacji stanowił punkt wyjścia dla wielu innych. Obecnie każdy z używanych systemów operacyjnych zawiera własną implementację protokołów grupy TCP/IP. W tej książce prezentować będziemy przykłady zaczerpnięte z implementacji dla systemów klasy Linux i Windows oraz (okazjonalnie) z implementacji dla FreeBSD i Mac OS X, wywodzących się w prostej linii ze wspomnianego na początku kodu BSD. W większości przypadków różnice między poszczególnymi implementacjami nie będą miały większego znaczenia.

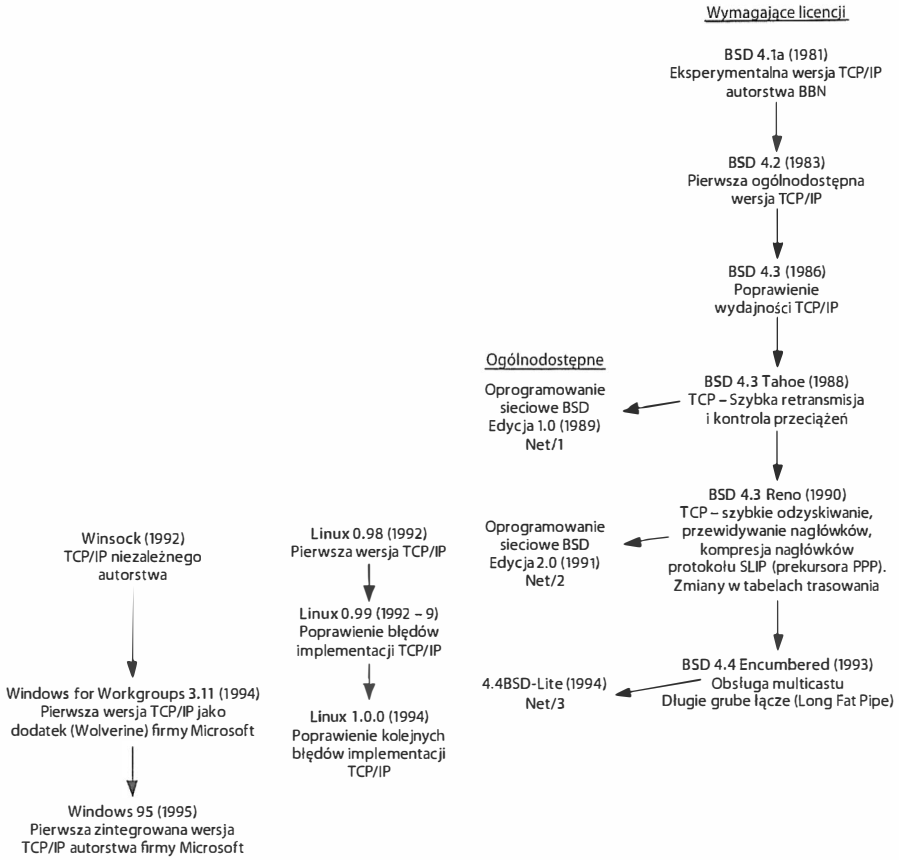
Na rysunku 1.7 przedstawiamy chronologię różnych edycji oprogramowania BSD, z uwzględnieniem istotnych elementów TCP/IP, które opisywać będziemy w następnych rozdziałach; wyodrębniono te rozwiązania — czyli implementacje protokołów z towarzyszącymi aplikacjami użytkowymi i narzędziowymi, m.in. Telnet i FTP) — które (wraz z kodem źródłowym) są ogólnie dostępne (w odróżnieniu od oprogramowania wymienionego w ostatniej kolumnie, wymagającego licencji na użytkowanie). Ukazujemy także historię rozwoju implementacji TCP/IP na platformach Linux i Windows.

Mniej więcej od połowy lat 90. ubiegłego stulecia implementacje protokołów TCP/IP stanowią pełnoprawny komponent popularnych systemów operacyjnych: nowym dystrybucjom Linuksa towarzyszą nieodłącznie nowe (często eksperymentalne) elementy wspomnianych implementacji (wcześniej spotykane tylko w nowych wersjach BSD), zaś w Windows Vista pojawiła się natywna obsługa protokołu IPv6 (i wiele innych elementów dotyczących IPv4); także w Linuksie, Mac OS X i FreeBSD obsługa IPv6 jest opcją domyślną, niewymagającą specjalnego konfigurowania.

1.8. Ataki wymierzone w architekturę Internetu

W tej książce będziemy pokrótce opisywać słabe punkty protokołów grupy TCP/IP, kryjące się zarówno w samych projektach, jak i w tworzonych implementacjach; słabości te mogą być (i skwapliwie są) wykorzystywane do przypuszczania rozmaitych ataków na bezpieczeństwo Internetu. Część tych ataków skierowana jest na architekturę Internetu jako taką; warto jednak zauważyć, że wędrówka datagramów IP odbywa się na podstawie *adresów docelowych* zapisanych w nagłówkach. W rezultacie niesforny użytkownik może umieścić w wysyłanym pakiecie IP dowolny adres źródłowy, realizując w ten sposób *podszywanie się* (*spoofing*) pod innego nadawcę. Utrudnia to (a często uniemożliwia) ustalenie pochodzenia (atrybucji) pakietów.

Spoofing może być kojarzony z innymi technikami ataku. Jedną z takich technik, periodycznie obserwowaną w Internecie, jest *atak przeciążeniowy* (*Denial of Service* — DoS) polegający na systematycznym intensywnym absorbowaniu ważnych zasobów serwera, przez co ten nie jest w stanie świadczyć swych usług legalnym użytkownikom.



Rysunek 1.7. *Historia rozwoju oprogramowania obsługującego TCP/IP, do roku 1995. Po raz pierwszy implementacje TCP/IP pojawiły się w systemie BSD. Na początku lat 90. ubiegłego wieku powstał Linux jako alternatywa dla BSD, adresowana do użytkowników PC (po części jako konsekwencja wątpliwości prawnych związanych z edycjami BSD). Kilka lat później implementacje TCP/IP pojawiły się na platformie Windows*

Sposoby przeciążania serwerów bywają rozmaite. Serwer może np. zostać zarzucony tak intensywnym strumieniem pakietów IP, że ich przetwarzanie nie pozostawia już wolnej mocy na wykonywanie czegokolwiek naprawdę użytecznego. Jednak nawet w pełni operatywny serwer okaże się mało użyteczny, jeśli przeciążona zostanie sieć, za pośrednictwem której kontaktuje się on z użytkownikami. Skuteczność ataku DoS można wyraźnie zwielokrotnić, przypuszczając ów atak jednocześnie z wielu skoordynowanych hostów — ten wariant przeciążenia nazywany jest *rozproszonym atakiem przeciążeniowym* (*Distributed Denial of Service* — DDoS).

Atakiem znacznie mniej widowiskowym, lecz wcale przez to nie mniej groźnym, jest *nieautoryzowany dostęp*, czyli nieuprawnione odczytywanie i (lub) modyfikowanie informacji. Wykorzystując luki w implementacji protokołów, użytkownik może *zawłaszczyć*

system (co w żargonie angielskim określa się jako *0wning* — na początku jest zero, nie „o") i podporządkować go swym niecnym celom, przekształcając w platformę ataku zwaną *zombie* lub *bot*). Nieuprawnione wtargnięcie do systemu realizowane jest często w imieniu legalnego użytkownika, z którego uprawnień korzysta włamujący się intruz.

Bardziej wyrafinowana forma opisanego ataku polega na zarażeniu zawłaszczonych komputerów szkodliwym oprogramowaniem (*malicious software*, w skrócie *malware*), wskutek czego formują one sieć, zwaną *botnetem*, przypuszczającą zmasowany atak na zasoby Internetu.

Generalnie, programiści tworzący i wykorzystujący szkodliwe oprogramowanie (oraz luki w systemach) jako środek osiągania określonych korzyści w nielegalny sposób określani są powszechnie mianem *czarnych kapeluszy* (*black hats*), w przeciwieństwie do *białych kapeluszy*, czyli ekspertów wykorzystujących te same działania do wykrywania i usuwania słabych punktów w protokołach sieciowych.

Jeden ze słabych punktów samej architektury Internetu związany jest z faktem, że oryginalna wersja protokołów internetowych nie wykorzystuje jakiejkolwiek odmiany szyfrowania przy wymianie informacji związanej z uwierzytelnianiem, ochroną integralności czy zapewnianiem poufności danych; w rezultacie cierpliwy intruz może wejść w posiadanie poufnej, prywatnej informacji w wyniku prostej obserwacji przesyłanych pakietów, a modyfikując tę informację, może wcielać się w innego użytkownika, niszczyć przesyłane komunikaty lub fabrykować fałszywe. Mimo iż znaczenie tego problemu znacząco zmalało po pojawieniu się protokołów szyfrujących (patrz rozdział 18.), to jednak protokoły starsze lub źle zaprojektowane wciąż podatne są na ataki podsłuchiwania (*eavesdropping*). Podsłuchiwanie takie staje się łatwiejsze w obliczu lawinowo rosnących sieci bezprzewodowych, należy zatem konsekwentnie unikać wspomnianych protokołów w tworzonych aplikacjach. Warto zauważyć, że jakkolwiek szyfrowanie może być realizowane w pojedynczej warstwie (np. w warstwie łącza danych Wi-Fi), ale jedynie szyfrowanie na całej trasie (*host-to-host*) w warstwie IP lub wyższej zapewnia prawdziwą ochronę pakietu IP w jego wędrówce przez potencjalnie wiele segmentów sieci.

1.9. Podsumowanie

W tym rozdziale przedstawiliśmy ogólne koncepcje architektury sieciowej i projektowania sieci komputerowych, ze szczególnym uwzględnieniem zestawu protokołów TCP/IP, których detale omawiać będziemy szczegółowo w następnych rozdziałach.

Architektura Internetu zaprojektowana została jako środek łączenia ze sobą istniejących różnych sieci, co w zamierzeniu zapewnić ma szeroką gamę oferowanych usług, za pomocą działających współbieżnie wielu różnych protokołów. Kierując się względami niezawodności i wydajności, architekturę tę zrealizowano w oparciu o sieci komutacji pakietów wykorzystujące datagramy; bezpieczeństwo i przewidywalność czasową (rozumianą jako limitowanie czasu opóźnienia) dostarczanych danych uznano za cele drugorzędne.

Autorzy wczesnych implementacji Internetu, bazując na swym doświadczeniu w zakresie koncepcji schematów warstwowych i modularności, nabytym przy projektowaniu systemów operacyjnych, zastosowali również i tym razem podejście warstwowe, połączone

z hermetyzacją (enkapsulacją): trzy najważniejsze warstwy protokołów grupy TCP/IP to warstwy: sieciowa, transportowa i aplikacyjna — każda obarczona innymi zadaniami. Nierozłącznie związana jest z nimi warstwa łącza danych (nienależąca jednak do modelu odniesienia TCP/IP). W następnych rozdziałach omówimy szczegółowo funkcjonowanie każdej z tych warstw.

Na gruncie TCP/IP bardzo wyraźnie uwidaczniają się różnice między warstwami sieciową i transportową. W warstwie sieciowej odbywa się transmisja datagramów, bez gwarancji niezawodności; warstwa ta jest niezbędna w każdym systemie widocznym („adresowalnym") dla Internetu. Dla odmiany, warstwa transportowa (w której funkcjonują m.in. protokoły TCP i UDP) zapewnia transmisję danych między aplikacjami działającymi w hostach końcowych. Protokoły TCP i UDP znacząco różnią się od siebie: TCP zapewnia transmisję strumienia uporządkowanych danych, w połączeniu ze sterowaniem przepływem i kontrolą przeciążeń, natomiast UDP oferuje niewiele ponad usługi dostarczane przez warstwę IP (m.in. obsługę numerów portów i ograniczoną kontrolę błędów). W przeciwieństwie jednak do TCP, UDP obsługuje transmisje typu *multicast*.

Rozróżnienie różnych protokołów (lub różnych połączeń czy skojarzeń w ramach tego samego protokołu) w procesie demultipleksowania możliwe jest dzięki różnym adresom i identyfikatorom. W warstwie łącza danych w sieciach wielodostępnych często spotyka się adresy 48-bitowe; protokół IPv4 używa adresów 32-bitowych, protokół IPv6 — 128-bitowych. Protokoły transportowe TCP i UDP korzystają z rozłącznych zbiorów numerów portów. Niektórym numerom portów przypisane jest oficjalnie ustalone znaczenie, inne natomiast używane są okazjonalnie przez klienty, podczas połączeń z serwerami. Numery portów nie mają żadnego znaczenia fizycznego, służą jedynie do jednoznacznego identyfikowania komunikujących się aplikacji.

Mimo iż adresy IP w połączeniu z numerami portów są wystarczające do identyfikowania konkretnych usług w konkretnych lokalizacjach, z punktu widzenia użytkownika są niewygodne w użyciu i trudne do zapamiętywania (dotyczy to szczególnie długich adresów protokołu IPv6). W związku z tym funkcjonuje w Internecie hierarchiczny system nazw, jednoznacznie odwzorowywanych w adresy IP (i vice versa) przez usługę DNS. Usługa ta, wykorzystująca rozproszoną bazę danych, stanowi jeden z najistotniejszych składników Internetu, nie ustają więc wysiłki zmierzające do tego, by stała się bardziej bezpieczna (patrz rozdział 18.).

Pisany z małej litery rzeczownik pospolity „internet" oznacza międzysieć, stanowiącą rezultat połączenia dwóch lub więcej odrębnych sieci za pomocą routera (routerów), z wykorzystaniem protokołów grupy TCP/IP. Oczywiście, wśród wszelkich internetów zdecydowanie wyróżnia się ten globalny, łączący komputery ponad dwóch miliardów użytkowników (w roku 2010) rozproszone po całym świecie, a wyrazem tego wyróżnienia jest pisownia przez duże „I" („Internet"). Rzeczownikiem „intranet" określamy prywatny internet, zwykle połączony z Internetem za pomocą specjalnych urządzeń (firewalli, patrz rozdział 10.) mających w zamierzeniu zapobiegać nieautoryzowanemu dostępowi z zewnątrz. Wydzieloną część intranetu, udostępnioną zaufanym partnerom, nazywamy „ekstranetem".

Większość aplikacji sieciowych należy do jednej z dwóch kategorii — klient-serwer lub peer-to-peer. Tradycyjnie pierwsza z tych kategorii jest bardziej rozpowszechniona, choć druga też cieszy się niesłabnącym powodzeniem. Niezależnie od kategorii, aplikacja

korzysta z niezbędnych usług sieciowych za pośrednictwem interfejsu programisty (API). Najbardziej znanym API dla sieci TCP/IP jest interfejs gniazd (*socket API*). Wywodzi się z dystrybucji systemu UNIX BSD, tworzonych przez programistów Uniwersytetu Kalifornijskiego w Berkeley; u schyłku ubiegłego wieku implementacje zestawu protokołów TCP/IP i interfejsu gniazd obecne były w każdym popularnym systemie operacyjnym.

Podczas różnicowania priorytetów celów projektowych Internetu przyznano pierwszeństwo wydajności i niezawodności, przesuwając na plan drugi bezpieczeństwo komunikacji. W efekcie określenie pochodzenia pakietów może być trudne lub niemożliwe dla odbiorcy, bo nadawca może fałszować adresy źródłowe zawarte w niechronionych datagramach IP. Zagrożeniem dla Internetu są także ataki przeciążeniowe (DoS — *Denial of Service*), najczęściej realizowane w postaci rozproszonej (DDoS — *Distributed Denial of Service*) przez armie skoordynowanych hostów, zwane botnetami. Oryginalny projekt protokołów TCP/IP nie przywiązuje większej wagi do kwestii ochrony przesyłanej informacji, jednak wiele starszych protokołów uznawanych jest za przestarzałe i są konsekwentnie wycofywane z użycia, ustępując miejsca nowszym protokołom wykorzystującym szyfrowanie w celu zapewnienia poufności i integralności treści przesyłanych między hostami.

1.10. Bibliografia

[B64] P. Baran, „On Distributed Communications: 1. Introduction to Distributed Communications Networks", RAND Memorandum RM-3420-PR, sierpień 1964.

[BT] *http://www.bittorrent.com*

[C88] D. Clark, „The Design Philosophy of the DARPA Internet Protocols", *Proc. ACM SIGCOMM*, sierpień 1988.

[CK74] V. Cerf, R. Kahn, *A Protocol for Packet Network Intercommunication*, „IEEE Transactions on Communications", COM-22(5), maj 1974.

[D08] J. Day, *Patterns in Network Architecture: A Return to Fundamentals* (Prentice Hall, 2008).

[D68] E. Dijkstra, *The Structure of the „THE"-Multiprogramming System*, „Communications of the ACM", 11(5), maj 1968.

[DBSW66] D. Davies, K. Bartlett, R. Scantlebury, P. Wilkinson, „A Digital Communications Network for Computers Giving Rapid Response at Remote Terminals", *Proc. ACM Symposium on Operating System Principles*, październik 1967.

[I96] IBM Corporation, *Systems Network Architecture — APPN Architecture Reference*, dokument SC30-3422-04, 1996.

[IPIS] Ipoque, *Internet Study 2008/2009*, *http://www.ipoque.com/sites/default/files/mediafiles/documents/internet-study-2008-2009.pdf*

[K64] L. Kleinrock, *Communication Nets: Stochastic Message Flow and Delay* (McGraw-Hill, 1964).

[LC04] S. Lin, D. Costello jr., *Error Control Coding*, wyd. drugie (Prentice Hall, 2004).

[LJFK93] S. Leffler, W. Joy, R. Fabry, M. Karels, *Networking Implementation Notes — 4.4BSD Edition*, czerwiec 1993.

[LT68] J.C.R. Licklider, R. Taylor, *The Computer as a Communication Device*, „Science and Technology", kwiecień 1968.

[OIPSW] *http://www.rfc-editor.org/rfcxx00.html*

[P07] J. Pelkey, *Entrepreneurial Capitalism and Innovation: A History of Computer Communications 1968 – 1988*, dostępne pod adresem *http://historyofcomputercommunications.info*

[P73] L. Pouzin, *Presentation and Major Design Aspects of the CYCLADES Computer Network*, NATO Advanced Study Institute on Computer Communication Networks, 1973.

[RFC0871] M. Padlipsky, *A Perspective on the ARPANET Reference Model*, Internet RFC 0871, wrzesień 1982.

[RFC0959] J. Postel, J. Reynolds, *File Transfer Protocol*, Internet RFC 0959/STD 0009, październik 1985.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1123] R. Braden (red.), *Requirements for Internet Hosts — Application and Support*, Internet RFC 1123/STD 0003, październik 1989.

[RFC1812] F. Baker (red.), *Requirements for IP Version 4 Routers*, Internet RFC 1812, czerwiec 1995.

[RFC3493] R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens, *Basic Socket Interface Extensions for IPv6*, Internet RFC 3493 (informational), luty 2003.

[RFC3542] W. Stevens, M. Thomas, E. Nordmark, T. Jinmei, *Advanced Sockets Application Program Interface (API) for IPv6*, Internet RFC 3542 (informational), maj 2003.

[RFC3678] D. Thaler, B. Fenner, B. Quinn, *Socket Interface Extensions for Multicast Source Filters*, Internet RFC 3678 (informational), styczeń 2004.

[RFC3787] J. Parker (red.), *Recommendations for Interoperable IP Networks Using Intermediate System to Intermediate System (IS-IS)*, Internet RFC 3787 (informational), maj 2004.

[RFC4294] J. Loughney (red.), *IPv6 Node Requirements*, Internet RFC 4294 (informational), kwiecień 2006.

[RFC4340] E. Kohler, M. Handley, S. Floyd, *Datagram Congestion Control Protocol (DCCP)*, Internet RFC 4340, marzec 2006.

[RFC4584] S. Chakrabarti, E. Nordmark, Extension to Sockets API for Mobile IPv6, Internet RFC 4584 (informational), lipiec 2006.

[RFC4677] P. Hoffman, S. Harris, *The Tao of IETF — A Novice's Guide to the Internet Engineering Task Force*, Internet RFC 4677 (informational), wrzesień. 2006.

[RFC4960] R. Stewart (red.), *Stream Control Transmission Protocol*, Internet RFC 4960, wrzesień 2007.

[RFC5000] RFC Editor, *Internet Official Protocol Standards*, Internet RFC 5000/STD 0001 (informational), maj 2008.

[RFC5014] E. Nordmark, S. Chakrabarti, J. Laganier, *IPv6 Socket API for Source Address Selection*, Internet RFC 5014 (informational), wrzesień 2007.

[RFC6250] D. Thaler, *Evolution of the IP Model*, Internet RFC 6250 (informational), maj 2011.

[SFR04] W.R. Stevens, B. Fenner, A. Rudoff, *UNIX Network Programming*, tom 1., wyd. trzecie (Prentice Hall, 2004).

[SKYPE] *http://www.skype.com*

[SRC84] J. Saltzer, D. Reed, D. Clark, *End-to-End Arguments in System Design*, „ACM Transactions on Computer Systems", 2(4), listopad 1984.

[W02] M. Waldrop, *The Dream Machine: J. C. R. Licklider and the Revolution That Made Computing Personal*, (Penguin Books, 1992).

[X85] Xerox Corporation, *Xerox Network Systems Architecture — General Information Manual*, XNSG 068504, 1985.

[Z80] H. Zimmermann, *OSI Reference Model — The ISO Model of Architecture for Open Systems Interconnection*, „IEEE Transactions on Communications", COM-28(4), kwiecień 1980.

Rozdział 2. Architektura adresów internetowych

2.1. Wprowadzenie

Rozdział ten poświęcamy adresom wykorzystywanym w warstwie sieciowej Internetu, znanym popularnie jako **adresy IP**. Pokażemy, jak kontroluje się ich przydział, jak przypisuje się je urządzeniom w Internecie i jak ich hierarchiczna natura okazuje się pomocna w skalowaniu trasowania. Przedstawimy różnice między adresami obu wersji protokołu IP — IPv4 i IPv6 — omówimy także adresy o specjalnym znaczeniu — broadcast, multicast i anycast.

Każdemu urządzeniu przyłączonemu do Internetu przypisany jest co najmniej jeden adres IP, dotyczy to również urządzeń stanowiących węzły sieci prywatnych, opartych na architekturze TCP/IP i przyłączonych do Internetu. W obu przypadkach procedury forwardujące zaimplementowane w routerach IP (szczegółowo opisywane w rozdziale 5.) wykorzystują adresy IP do określenia, skąd wysłane zostały pakiety IP i jakie jest ich przeznaczenie (urządzenie docelowe). Aby zrozumieć, jak procedury te funkcjonują i w jaki sposób adresy IP pełnią rolę identyfikacyjną dla urządzeń w Internecie, konieczne jest zrozumienie ich struktury, sposobów użycia oraz metod administrowania nimi.

Pod wieloma względami adresy IP podobne są do numerów telefonicznych, od których jednak różnią się wyraźnie jedną cechą: podczas gdy numery telefoniczne zapamiętywane (zapisywane) są i wykorzystywane w postaci bezpośredniej, adresy IP — z punktu widzenia przeciętnego użytkownika Internetu — skrywają się pod podszewką bardziej przyjaznego mechanizmu nazw mnemonicznych, zarządzanych przez usługę DNS (patrz rozdział 11.). Niekiedy użytkownicy — mimo wszystko — stają oko w oko z adresami IP, gdy przychodzi im samodzielnie konfigurować sieć lub gdy z pewnych powodów usługa DNS staje się niedostępna.

Jako że adresy IP pełnią rolę identyfikacyjną dla urządzeń w Internecie, muszą być owym urządzeniom przyporządkowywane w sposób jednoznaczny, wykluczający powtórzenia; ta sama zasada dotyczy urządzeń składających się na określoną sieć prywatną. Na globalnym poziomie Internetu problem ten rozwiązuje się poprzez formalizację *przydziału* adresów dla organizacji i użytkowników indywidualnych — w gestii tych organizacji

i użytkowników spoczywa następnie *przypisanie* przydzielonych adresów poszczególnym urządzeniom (co zwykle odbywa się zgodnie z pewnym „planem numeracyjnym"). Użytkownicy indywidualni zazwyczaj partycypują w wykorzystywaniu adresów IP za pośrednictwem dostawców usług internetowych (ISP — *Internet Service Providers*), którzy oprócz przydzielania adresów zapewniają także (za odpowiednią opłatą) właściwe przekazywanie pakietów między sieciami prywatnymi a Internetem.

2.2. Zapisywanie adresów IP

Dla większości użytkowników Internetu znajomo wyglądają tradycyjne adresy IPv4, reprezentowane zwykle w formacie „kropkowo-czwórkowym" (*dotted-quad*) zwanym też „kropkowo-dziesiętnym" (*dotted-decimal*): cztery składniki adresu, będące liczbami dziesiętnymi z zakresu 0 – 255, rozdzielone są kropkami — jak w przykładowym adresie 165.195.130.107. Każda z tych liczb odzwierciedla wartość jednego bajta w 4-bajtowym (32-bitowym) adresie. Alternatywą, w wielu przypadkach wygodniejszą, dla formatu kropkowo- dziesiętnego jest binarna postać 32-bitowego słowa. W tabeli 2.1 przedstawiono kilka przykładowych adresów IPv4 w obu wspomnianych formatach; konwersję między formatami wygodniej przeprowadzić, posiłkując się jedną z dostępnych w tym celu witryn internetowych, np. witryną *http://www.subnetmask.info*.

Tabela 2.1. *Przykładowe adresy IPv4 w formatach kropkowo-dziesiętnym i binarnym*

Reprezentacja kropkowo-dziesiętna	Reprezentacja binarna
0.0.0.0	00000000 00000000 00000000 00000000
1.2.3.4	00000001 00000010 00000011 00000100
10.0.0.255	00001010 00000000 00000000 11111111
165.195.130.107	10100101 11000011 10000010 01101011
255.255.255.255	11111111 11111111 11111111 11111111

Adresy IPv6 są nieco mniej popularne (podobnie jak sam protokół IPv6). Mają długość 128 bitów i zapisywane są w podziale na osiem 16-bitowych **bloków** rozdzielonych dwukropkiem, każdy blok reprezentowany jest przez 4 cyfry szesnastkowe — jak w przykładowym adresie 5f05:2000:80ad:5800:0058:0800:2023:1d71. Ze względu na użycie notacji szesnastkowej konwersja na postać binarną jest tu znacznie prostsza niż w przypadku dziesiętnej notacji IPv4. W celu wygodniejszego operowania adresami IPv6 zdefiniowano w dokumencie [RFC4291] kilka dopuszczalnych reguł ich upraszczania. Oto one.

1. W ramach każdego z bloków dopuszczalne jest pominięcie nieznaczących („wiodących") zer; przykładowy adres 5f05:2000:80ad:5800:0058:0800:2023:1d71 można zatem uprościć do postaci 5f05:2000:80ad:5800:58:800:2023:1d71.
2. Ciąg sąsiadujących bloków zerowych można pominąć wraz z rozdzielającymi dwukropkami, co daje w wyniku parę sąsiadujących dwukropków — i tak adres 0:0:0:0:0:0:0:1 można skrócić do postaci ::1, podobnie adres 2001:0db8:0:0:0:0:0:2 redukuje się do postaci 2001:db8::2. Jeżeli w adresie występuje kilka ciągów zerowych bloków, operację tę można wykonać w odniesieniu do *co najwyżej jednego z nich* — naruszenie tego ograniczenia spowodowałoby utratę jednoznaczności notacji.

3. Reprezentowanie adresu IPv4 w notacji IPv6 polega na poprzedzeniu go przedrostkiem `::ffff:` i zachowaniu postaci kropkowo-dziesiętnej, tak więc zapis `::ffff:10.0.0.1` równoważny jest adresowi IPv4 `10.0.0.1`. Zapis taki nosi nazwę **notacji hybrydowej** lub **adresu IPv4 mapowanego w IPv6**.
4. Podobnie 32 najmniej znaczące bity adresu IPv6 mogą być zapisywane w postaci kropkowo-dziesiętnej, co nosi nazwę **notacji kompatybilnej**; przykładowo adres `::0102:f001` można zapisać w postaci `::1.2.240.1`. Zauważmy, że notacja kompatybilna jest czymś innym niż mapowanie opisane w poprzednim punkcie. Notacja kompatybilna bywa użyteczna przy zmianie planu sieci z wersji IPv4 na IPv6, wykorzystywana jest jednak coraz rzadziej (patrz [RFC4291]).

W tabeli 2.2 przedstawiono kilka przykładowych adresów IPv6 wraz z ich reprezentacjami binarnymi.

Tabela 2.2. *Przykładowe adresy IPv6 wraz z reprezentacjami binarnymi*

Notacja szesnastkowa	Reprezentacja binarna
5f05:2000:80ad:5800:58:800:2023:1d71	0101111100000101 0010000000000000 1000000010101101 0101100000000000 0000000001011000 0000100000000000 0010000000100011 0001110101110001
::1	0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000001
::1.2.240.1 or ::102:f001	0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000100000010 1111000000000001

W sytuacji gdy adres IPv6 występuje w kontekście nadającym określone znaczenie znakowi dwukropka, może pojawić się niejednoznaczność; jest tak np. w kontekście lokalizatora URL, w którym dwukropek oddziela adres serwera od numeru portu. W takim przypadku można ująć adres IPv6 w **nawiasy prostokątne**, co usuwa wszelkie wątpliwości, jak w poniższym przykładzie:

```
http://[2001:0db8:85a3:08d3:1319:8a2e:0370:7344]:443/
```

w którym żądanie kierowane jest do portu 443 serwera pod adresem `2001:0db8:85a3:`
`↪08d3:1319:8a2e:0370:7344`.

Wymienione reguły upraszczania adresów IPv6, opisane w dokumencie [RFC4291], są opcjonalne (dobrowolne), co stwarza dość dużą swobodę notacji — swobodę prowadzącą często do nieporozumień, a także do komplikacji w przetwarzaniu samych adresów.

W związku z tym w dokumencie [RFC5952] zdefiniowano dwie kolejne reguły, obligujące niejako do korzystania z udogodnień oferowanych przez [RFC4291]:

1. Usuwanie wiodących zer (o ile występują) jest obowiązkowe w każdym bloku.
2. Jeżeli w adresie występuje kilka ciągów sąsiadujących zerowych bloków, należy zredukować (do postaci `::`) najdłuższy z nich (w przypadku dwóch ciągów o jednakowej długości należy zredukować pierwszy z nich).
3. Cyfry szesnastkowe (`a` – `f`) muszą być zapisywane małymi literami.

Nie są to ograniczenia nazbyt uciążliwe, a mogą ułatwić życie.

2.3. Podstawowa struktura adresu IP

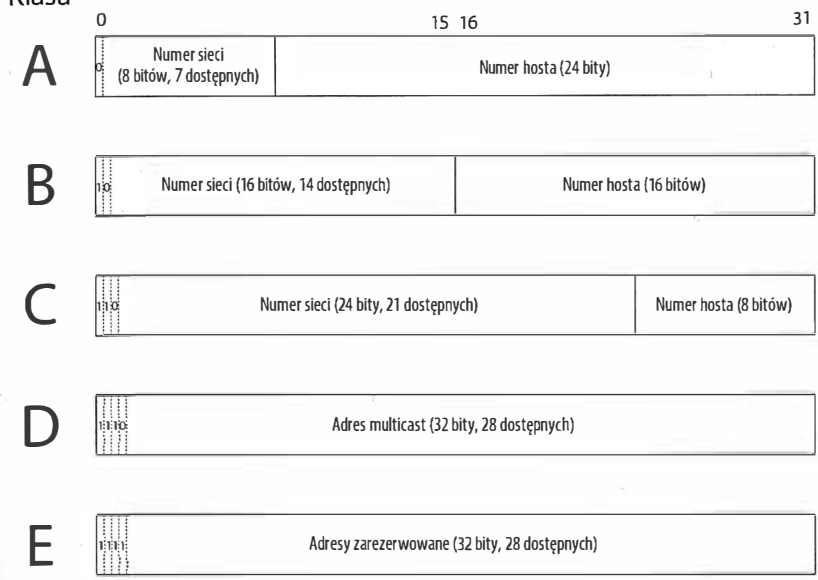
Adresy IPv4 są 32-bitowe, można ich więc skonstruować $2^{32} = 4\,294\,967\,296$. Możliwych adresów IPv6 jest znacznie więcej, bo $2^{128} = 340\,282\,366\,920\,938\,463\,463\,374\,607\,431\,768\,211\,456$. W obu przypadkach przestrzeń adresów jest na tyle liczna, że wygodnie jest podzielić ją na kategorie, uwarunkowane typami i rozmiarami pewnych obiektów. Większość adresów IPv4 to adresy typu unicast, czyli odnoszące się do konkretnego interfejsu sieciowego komputera przyłączonego do internetu (w szczególności: Internetu) lub intranetu. Spośród możliwych adresów IPv6 znakomita większość nie jest obecnie wykorzystywana. Oprócz adresów unicast istnieją jeszcze inne ich typy — multicast, broadcast i anycast — odnoszące się do wielu interfejsów, a także grupa adresów o specjalnym znaczeniu. Aby lepiej zrozumieć strukturę adresów IP, szczególnie zaś powody takiego, a nie innego jej ukształtowania, warto prześledzić historyczną ich ewolucję od samych początków.

2.3.1. Klasy adresów IP

W swej pierwotnej postaci adres IP typu unicast dzielił się na dwie części: pierwsza, obejmująca bity bardziej znaczące, stanowiła identyfikację *sieci* przyłączonej do Internetu, druga, obejmująca resztę bitów, identyfikowała *konkretny host* w tej sieci. Ponieważ większość hostów wyposażona była w pojedynczy interfejs (czyli jedną kartę sieciową), określenia „adres hosta" i „adres interfejsu" używane były jako synonimy.

Oczywisty poniekąd fakt, że poszczególne sieci różnią się rozmiarami (czyli liczbą hostów), znalazł swe odzwierciedlenie w zróżnicowaniu ulokowania granicy dzielącej dwa wymienione komponenty adresu — numer sieci i numer hosta. Przestrzeń adresów IPv4 podzielona została mianowicie na pięć klas, zilustrowanych schematycznie na rysunku 2.1; klasa konkretnego adresu określona jest przez jego najbardziej znaczące bity.

I tak klasy A, B, i C to adresy unicast przeznaczone na potrzeby sieci o maksymalnej liczbie hostów (odpowiednio) 16 777 216, 65 536 i 256, różniące się odmiennym podziałem na numer sieci i numer hosta, odpowiednio 8:24, 16:16 i 24:8 bitów. Klasa D to adresy typu multicast, klasa E obejmuje natomiast adresy zarezerwowane do specjalnego wykorzystania. Podstawowe właściwości wszystkich pięciu klas zestawiono w tabeli 2.3., z której wyraźnie wynika sposób wykorzystywania klasowej struktury adresów na potrzeby sieci różnej wielkości. Struktura ta stanowi wynik pewnego kompromisu między liczbą dostępnych adresów a maksymalną liczbą hostów w sieci opatrzonej konkretnym adresem.



Rysunek 2.1. *Oryginalny podział przestrzeni adresów IPv4 na pięć klas. Klasy A, B i C to adresy unicast przeznaczone dla sieci o różnej (maksymalnej) liczbie hostów oraz adresy używane w pewnych szczególnych przypadkach. Adres klasy A rozpoczyna się od bitu 0, adres klasy B — od ciągu bitów 10, adres klasy C — od ciągu bitów 110. Adresy klasy D, rozpoczynające się od ciągu 1110, to adresy multicast (patrz rozdział 9.), adresy klasy E, rozpoczynające się od ciągu 1111, zarezerwowane są do specjalnego użytku*

Tabela 2.3. *Oryginalne partycjonowanie adresów IP w ramach podziału na klasy*

Klasa	Zakres adresów	Początkowe bity identyfikujące klasę	Przeznaczenie	Udział w puli adresów	Liczba sieci	Maksymalna liczba hostów w sieci
A	0.0.0.0 – 127.255.255.255	0	Unicast lub specjalne	$\frac{1}{2}$	128	16 777 216
B	128.0.0.0 – 191.255.255.255	10	Unicast lub specjalne	$\frac{1}{4}$	16 384	65 536
C	192.0.0.0 – 223.255.255.255	110	Unicast lub specjalne	$\frac{1}{8}$	2 097 152	256
D	224.0.0.0.0 – 239.255.255.255	1110	Multicast	$\frac{1}{16}$	Nie dotyczy	Nie dotyczy
E	240.0.0.0.0 – 255.255.255.255	1111	Zarezerwowane	$\frac{1}{16}$	Nie dotyczy	Nie dotyczy

Tak więc w sieci klasy A[1] (jest nią np. sieć Massachusetts Institute of Technology, obejmująca adresy postaci 18.*.*.*) może znajdować się maksymalnie 2^{24} = 16 777 216 hostów, lecz liczba takich sieci ograniczona jest do 127 w całym Internecie. Dla odmiany, wszystkich sieci klasy C może być 2^{21} = 2 097 152, lecz każda z nich może zawierać nie więcej niż 256 hostów.

Powyższe wyliczenia należy zmodyfikować o tyle, że pewne szczególne postaci adresów nie mogą być używane jako unicast; generalnie dotyczy to najmniejszego i największego adresu w danej klasie, w związku z czym podaną w tabeli 2.2 maksymalną liczbę hostów należy zmniejszyć o 2 — w przykładowej sieci MIT może więc być maksymalnie 16 777 214 hostów.

„Klasowe" podejście do adresowania w Internecie okazało się znakomicie funkcjonować przez pierwszą dekadę, do początku lat 80. ubiegłego wieku, kiedy to wystąpiły pierwsze symptomy problemów ze skalowalnością. Centralny przydział adresów dla każdej sieci przyłączanej do Internetu okazał się zbyt niewygodny, ponadto zasoby adresów dostępnych w sieciach klasy A i B często były wykorzystywane w niewielkiej części, a ograniczenia liczby hostów w sieciach klasy C okazywały się nazbyt dotkliwe.

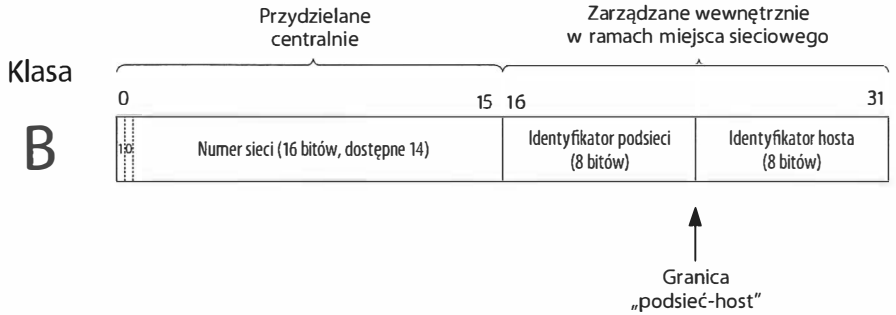
2.3.2. Adresowanie podsieci

Jedną ze znaczących pierwszych niewygód, jakie objawiły się na drodze rozwojowej Internetu, była konieczność centralnego przydzielania adresu IP każdej przyłączanej do Internetu sieci; uciążliwość ta stała się wyjątkowo odczuwalna w latach 80. dwudziestego wieku, kiedy to nastąpił lawinowy wręcz rozwój lokalnych sieci komputerowych (LAN). Rozważano wówczas możliwość częściowego przejęcia kompetencji w tym zakresie przez administratorów poszczególnych sieci — gdyby udało się to wykonać bez ingerowania w „rdzenną" strukturę Internetu, opisany problem można by znacząco złagodzić.

Aby ów pomysł urzeczywistnić, należało stworzyć możliwość przesuwania sztywnej granicy między numerem sieci a numerem hosta, lecz tylko w kontekście sieci przyłączanej do Internetu (w oryginalnej terminologii określa się ją jako *site*, co tłumaczy się na język polski jako „miejsce sieciowe") — z perspektywy „reszty" Internetu wszystko miało pozostać „po staremu", czyli z zachowaniem tradycyjnego partycjonowania adresów unicast na klasy A, B i C. Cel ten realizuje mechanizm **adresowania podsieci** (*subnet addressing*) opisany w dokumencie [RFC0950]. Najbardziej znaczące bity adresu (w liczbie 8, 16 albo 24, zależnie od klasy) zachowują swe dotychczasowe znaczenie; pozostałe bity, zawierające dotychczas numer hosta, oddane zostają natomiast do dyspozycji administratora miejsca sieciowego, który to administrator może podzielić je (według swego uznania) na dwa pola, zawierające (kolejno) *numer podsieci* (*subnet number*) i *numer hosta w podsieci*. Zyskujemy w ten sposób trzecie pole w adresie IP, bez używania dodatkowych bitów, z zachowaniem 32-bitowej długości adresu. Zależnie od potrzeb administrator może zdecydować się na małą liczbę podsieci z dużą liczbą hostów w każdej bądź na większą liczbę podsieci złożonych z niewielu hostów.

[1] Sformułowanie „sieć klasy *x*" jest tu skrótem określenia „sieć identyfikowana przez adres IP klasy *x*" — *przyp. tłum.*

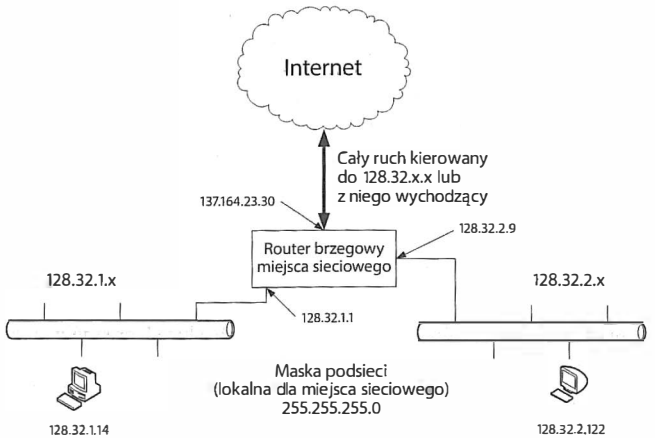
Na rysunku 2.2 przedstawiono przykładowy podział adresu klasy B. Zgodnie z formatem tej klasy 16 najbardziej znaczących bitów ustalane jest centralnie; pozostałe 16 bitów oddane zostaje do dyspozycji administratora miejsca sieciowego — w tym konkretnym przypadku zdecydował się on na 256 podsieci, z których każda może zawierać do 254 hostów (dlaczego nie 256? — patrz ostatnia uwaga). W każdej podsieci tracimy w ten sposób dwa skrajne numery hostów, co jest nowością w stosunku do stanu poprzedniego (bez podsieci), w którym wykluczone były jedynie dwa skrajne numery w ramach całego miejsca sieciowego.



Rysunek 2.2. *Przykładowy podział adresu IP klasy B na pola sieci, podsieci i hosta: każda z 256 podsieci może zawierać do 254 hostów. Położenie granicy „podsieć-host" może być zmieniane przez administratora w obrębie 16 najmniej znaczących bitów*

Jak już wspominaliśmy, mechanizm ten jest niewidoczny dla Internetu, musi być zatem zrealizowany wewnątrz sieci docelowej. Osiągnięte udogodnienie zostaje więc okupione pewnym kosztem, bo na barkach routerów miejsca sieciowego spoczywa obowiązek wydzielenia numeru podsieci i numeru hosta (przed pojawieniem się pomysłu adresowania podsieci podział adresu na numer miejsca sieciowego i numer hosta odbywał się automatycznie, na podstawie klasy adresu). Administrator sieci musi nauczyć routery tej sztuki — zajmiemy się tym w punkcie 2.3.3. Przykład funkcjonującego w ten sposób miejsca sieciowego przedstawiono na rysunku 2.3.

Widzimy tu hipotetyczne miejsce sieciowe, przyłączone do Internetu za pośrednictwem routera brzegowego, zawierające dwie sieci LAN. Adres IP tego miejsca sieciowego rozpoczyna się od centralnie przydzielonego członu 128.32 — cały ruch w Internecie kierowany do adresu o takim początku dostarczony zostanie do wspomnianego routera brzegowego (dokładniej — do jego interfejsu sieciowego identyfikowanego przez adres 137.164.23.30). Zadaniem tegoż routera brzegowego jest zidentyfikowanie docelowej (pod)sieci LAN: adresy w postaci 128.32.1.*x* przynależne są sieci lewostronnej, adresy w postaci 128.32.2.*x* — sieci prawostronnej; *x* może przyjmować wartości z przedziału 0 – 255. Taka klasyfikacja adresu wymuszana jest przez zdefiniowanie odpowiedniej *maski sieciowej*.



Rysunek 2.3. *Miejsce sieciowe identyfikowane w Internecie przez adres IPv4 rozpoczynający się od 128.32. Administrator sieci zdefiniował maskę podsieci 255.255.255.0, co oznacza utworzenie 256 podsieci, z których każda zawierać może maksymalnie 256–2 = 254 hosty. Wszystkie hosty z podsieci przedstawionej po lewej stronie rysunku mają adresy rozpoczynające się od 128.32.1, zaś hosty z podsieci prawostronnej — adresy 128.32.2*

2.3.3. Maski podsieci

Maska podsieci to ciąg bitów, który w wyniku koniunkcji bitowej z adresem IP ekstrahuje z niego konkatenację pól zawierających numer miejsca sieciowego i numer podsieci (czyli oddziela te pola od numeru hosta). Długość maski podsieci jest tożsama z długością adresu — czyli równa 32 bitom w wersji IPv4 i 128 bitom w wersji IPv6. Maska podsieci rozpoczyna się więc ciągiem bitów jedynkowych, po których następuje ciąg bitów zerowych. W wersji IPv4 można zapisywać tę maskę w postaci kropkowo-dziesiętnej, w obu natomiast wersjach przyjęło się powszechnie jej zapisywanie w postaci tzw. **długości prefiksu**: ponieważ długość maski jest określona a priori (bo tożsama z długością adresu), wystarczające jest podanie rozpoczynającego ją ciągu jedynkowego, zwanego w tym kontekście prefiksem. W tabeli 2.4 podano kilka przykładów masek IPv4 we wszystkich trzech zapisach, w tabeli 2.5 widzimy natomiast dwie specyficzne maski dla adresów IPv6.

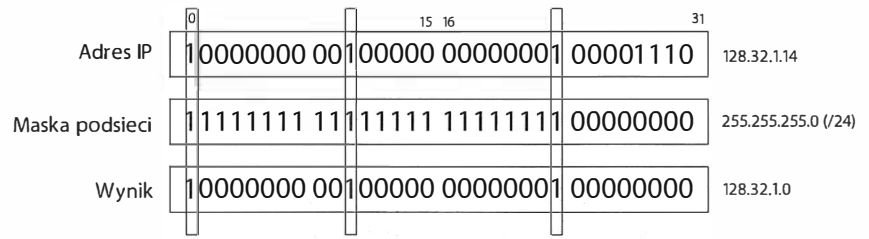
Tabela 2.4. *Przykładowe maski podsieci dla adresów IPv4*

Reprezentacja bitowa	Zapis kropkowo-dziesiętny	Zapis prefiksowy
10000000 00000000 00000000 00000000	128.0.0.0	/1
11111111 00000000 00000000 00000000	255.0.0.0	/8
11111111 11000000 00000000 00000000	255.192.0.0	/10
11111111 11111111 00000000 00000000	255.255.0.0	/16
11111111 11111111 11111110 00000000	255.255.254.0	/23
11111111 11111111 11111111 11100000	255.255.255.192	/27
11111111 11111111 11111111 11111111	255.255.255.255	/32

Tabela 2.5. *Przykładowe maski podsieci dla adresów IPv6*

Notacja szesnastkowa	Reprezentacja bitowa	Zapis prefiksowy
ffff:ffff:ffff:ffff::	1111111111111111 1111111111111111 1111111111111111 1111111111111111 0000000000000000 0000000000000000 0000000000000000 0000000000000000	/64
ff00::	1111111100000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000	/8

Maska podsieci wykorzystywana jest przez router(y) miejsca sieciowego w celu zidentyfikowania podsieci zawierającej docelowy host. Bit będący wynikiem koniunkcji dwóch bitów (AND) równy jest 1 tylko wtedy, gdy oba mają wartość 1, zatem wynikiem operacji AND na masce podsieci i adresie IP jest wyzerowanie w tym adresie bitów reprezentujących numer hosta. Na rysunku 2.4 widzimy efekt zastosowania maski 255.255.255.0 do adresu 128.32.1.14.

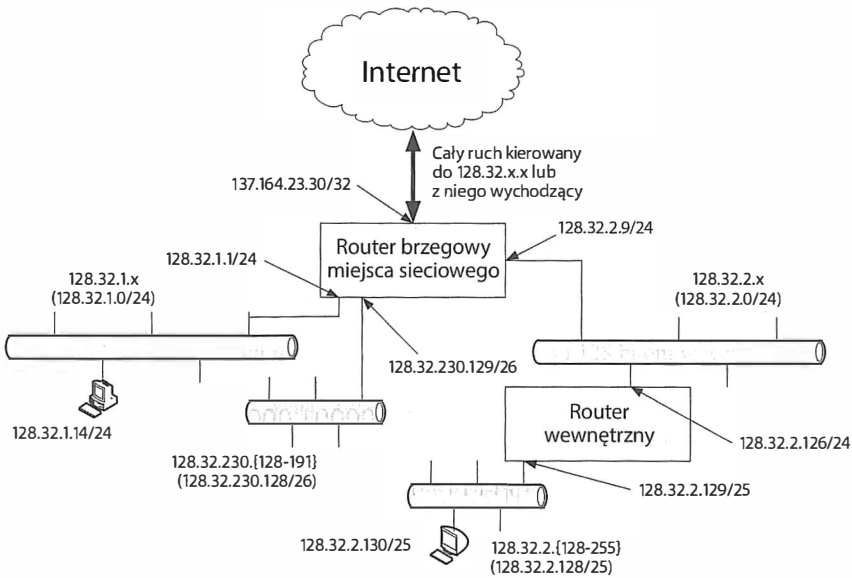


Rysunek 2.4. *Wynik koniunkcji bitowej adresu IP i maski podsieci; z adresu 128.32.1.14 wyekstrahowana została podsieć 128.32.1.0/24*

Maska podsieci definiowana może być w sposób statyczny (co jest typowe w przypadku routerów) lub za pomocą jakiegoś dynamicznego systemu używanego do przydzielania adresów IP (np. DHCP — *Dynamic Host Configuration Protocol*, patrz rozdział 6.). Routery miejsca sieciowego wykorzystują tę maskę do zidentyfikowania docelowej podsieci; należy w tym miejscu przypomnieć, że mechanizm ten nie jest zauważalny dla routerów w Internecie — dla nich istotny jest tylko numer sieci (na rysunku 2.4, przedstawiającym adres klasy B, jest to 16-bitowy numer 128.32), zapewniający skierowanie datagramu do odpowiedniego miejsca sieciowego.

2.3.4. Zmienna długość maski podsieci (VLSM)

Ponieważ podział „końcówki" adresu IP na numer podsieci i numer hosta jest prywatną sprawą routerów miejsca sieciowego, możliwe jest jego zrealizowanie w formie bardziej elastycznej niż przez sztywne ustalenie a priori granicy między wymienionymi polami. Elastyczność taka, choć komplikująca zarządzanie konfiguracją miejsca sieciowego, stwarza większe możliwości w zakresie jego skalowania, bo poszczególnym podsieciom można przypisywać zróżnicowane ograniczenia maksymalnej liczby hostów. Oczywiście, w tych warunkach maska podsieci nie ma już długości sztywno ustalonej dla danego miejsca sieciowego, lecz zmienia się dla poszczególnych podsieci. Mechanizm ten, zwany po prostu **maską sieciową o zmiennej długości** (VLSM — *VariableLlength Subnet Mask*) obsługiwany jest obecnie przez większość routerów i protokołów trasowania, a przykład jego wykorzystania przedstawiono na rysunku 2.5 (stanowiącym rozszerzenie rysunku 2.3 o dwie dodatkowe podsieci).



Rysunek 2.5. *Użycie mechanizmu VLSM do partycjonowania miejsca sieciowego w sposób różnicujący maksymalną liczbę hostów w poszczególnych podsieciach. Różne routery wykorzystują maski podsieci o różnych długościach: dla routera brzegowego maska podsieci jest 24-bitowa, dla routera wewnętrznego 25-bitowa. Mechanizm ten jest obsługiwany przez większość obecnego sprzętu i oprogramowania, nie honorują go niektóre starsze protokoły trasowania (m.in. RIP w wersji 1.)*

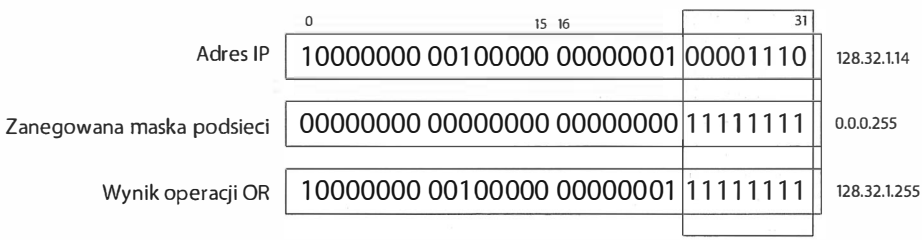
W konfiguracji przedstawionej na rysunku 2.5 — notabene bliższej rzeczywistości niż ta z rysunku 2.3 — widzimy użycie trzech różnych masek podsieci — /24, /25 i /26 — w miejscu sieciowym o numerze 128.32. W efekcie trzy różne podsieci cechują się odmiennymi ograniczeniami na maksymalną liczbę hostów: przy masce /24 na numer hosta pozostaje 32–24 = 8 bitów, co (przy zarezerwowaniu dwóch skrajnych numerów) daje możliwość podłączenia 254 hostów; dla masek /25 i /26 mamy (odpowiednio) 126 hostów i 62 hosty. Przy zastosowaniu odpowiedniego protokołu trasowania dynamicznego

(np. OSPF, IS-IS lub RIPv2) zapewnia to prawidłowy przepływ datagramów zarówno między wszystkimi hostami miejsca sieciowego, jak i między miejscem sieciowym a Internetem.

W skrajnym przypadku — przy użyciu maski podsieci /31 w IPv4 i maski /127 w IPv6 — otrzymujemy podsieć składającą się jedynie z dwóch hostów. Choć na pierwszy rzut oka konfiguracja taka wydaje się cokolwiek dziwna, to jednak stosowana jest w sytuacjach, gdy dwa routery połączone są łączem wymagającym adresów IP na swoich końcach. Konfiguracja taka opisana jest m.in. w dokumencie [RFC6164].

2.3.5. Adresy rozgłoszeniowe (broadcast)

W każdej podsieci IPv4 jeden z adresów jest zarezerwowany jako **adres rozgłoszeniowy** tej podsieci (*subnet broadcast address*): jest to adres, w którym numer hosta ma wartość maksymalną, czyli składa się z samych bitów jedynkowych. W skrajnej lewostronnej podsieci na rysunku 2.5 — tej o prefiksie 128.32.1.0/24 — adresem rozgłoszeniowym jest 128.32.1.255. Adres rozgłoszeniowy danej podsieci otrzymuje się w sposób zgoła nieskomplikowany: należy mianowicie *zanegowaną* maskę tej podsieci zsumować logicznie (OR) z adresem miejsca sieciowego. Wynik operacji OR ma wartość 0 tylko wtedy, gdy oba argumenty są zerowe: dodanie zera nie zmienia więc wartości argumentu, a dodanie jedynki zawsze daje wynik 1. Zerowe bity z zanegowanego prefiksu maski nie zmieniają więc numeru sieci i numeru podsieci, zaś jedynkowe bity reszty zanegowanej maski kopiowane są wprost do pola numeru hosta. Ilustrację tej operacji dla adresu IPv4 128.32.1.14 przedstawiono na rysunku 2.6.



Rysunek 2.6. *Konstrukcja adresu rozgłoszeniowego dla podsieci o prefiksie 128.32.1.14/24. Pierwsze 24 bity adresu pozostają niezmienione, pozostałe 32–24 = 8 bitów ustawionych zostaje na 1, w wyniku czego otrzymujemy adres rozgłoszeniowy 128.32.1.255*

Jak widać na rysunku, adresem rozgłoszeniowym dla podsieci 128.32.1.0/24 jest 128.32.1.255. Tradycyjnie datagram zawierający adres docelowy w opisanej postaci znany jest pod nazwą **rozgłaszania ukierunkowanego** lub **rozgłaszania bezpośredniego** (*directed broadcast*): wędruje on przez Internet tak jak inne datagramy (przynajmniej teoretycznie), a po dotarciu do docelowego miejsca sieciowego zostaje rozszczepiony na kolekcję datagramów docierających do wszystkich hostów określonej podsieci. Powracając do rysunków 2.3 i 2.5 oraz generalizując opisaną ideę, możemy powiedzieć, że datagram IPv4 z adresem docelowym 128.32.255.255 po dotarciu do miejsca sieciowego 128.32 zostałby powielony we wszystkich hostach tegoż miejsca.

Ponieważ pakiety rozgłaszania bezpośredniego stanowią duży problem z punktu widzenia bezpieczeństwa sieci, ich forwardowanie jest obecnie zablokowane w Internecie. Początkowo było inaczej — w dokumencie [RFC1812] znajduje się wręcz zalecenie, by forwardowanie rozgłoszeń bezpośrednich było nie tylko obowiązkową funkcją routerów, lecz by było również domyślnie włączone. W dokumencie [RFC2644] rekomendacja ta została dokładnie odwrócona: nie tylko nakazuje się domyślne wyłączenie rozgłoszeń bezpośrednich, lecz także dopuszcza rezygnację z implementowania tej funkcji w routerach. Różne typy rozgłaszania w sieciach IPv4 opisane są w dokumencie [RFC0919].

W uzupełnieniu do rozgłaszania bezpośredniego można dodać, że jeden z adresów klasy E — 255.255.255.255 — zarezerwowany jest na potrzeby **rozgłaszania lokalnego** (zwanego także **rozgłaszaniem ograniczonym** — *limited broadcast*). Mimo iż routery mogą blokować forwardowanie pakietów o takim adresie docelowym, pakiety te mogą z powodzeniem funkcjonować w ramach danej podsieci lub w ramach całego miejsca sieciowego, o ile ich obsługa nie została zablokowana w hostach docelowych. Ich obsługa nie angażuje wówczas routera, wspomagana jest natomiast przez mechanizmy rozgłoszeniowe warstwy łącza danych (jeśli takowe są używane — patrz rozdział 3.). Adresy rozgłoszeniowe wykorzystywane są zwykle przez protokoły w rodzaju UDP/IP (patrz rozdział 10.) czy ICMP (patrz rozdział 8.) nienawiązujące dwustronnej konwersacji. W protokole IPv6 w ogóle nie istnieją adresy rozgłoszeniowe; sytuacje, w których w protokole IPv4 używa się rozgłaszania, na gruncie protokołu IPv6 obsługiwane są za pomocą adresów multicast (patrz rozdział 9.).

2.3.6. Adresy IPv6 i identyfikatory interfejsów

Adresy IPv6, poza tym że czterokrotnie dłuższe od adresów IPv4, cechują się dodatkowo wewnętrzną strukturą. Jednym z elementów tej struktury są specjalne prefiksy oznaczające **zakres** — pod tym pojęciem rozumiemy tę część sieci, w której dany adres jest obowiązujący: adresy **lokalne dla węzła** (*local-node*) obowiązują jedynie w komunikacji wewnątrz konkretnego komputera, adresy **lokalne dla łącza** (*link-local*) wykorzystywane są w komunikacji przez określone łącze, a adresy **globalne** mają jednakowe znaczenie na obszarze całego Internetu. Zgodnie z zasadami protokołu IPv6, większość węzłów sieci opatrzona jest kilkoma adresami (często związanymi z tym samym interfejsem) — jakkolwiek taka możliwość istnieje także w ramach protokołu IPv4, wykorzystywana jest raczej rzadko. W dokumencie [RFC4291] wyszczególniony jest zbiór adresów wymaganych dla każdego węzła sieci IPv6, w tym adresów multicast (patrz punkt 2.5.2).

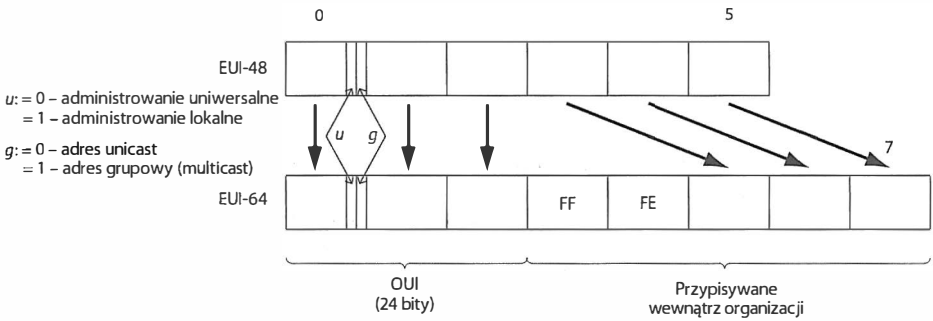
Zdefiniowany początkowo zakres **lokalny dla miejsca sieciowego** (*site-local*), identyfikowany za pomocą prefiksu `fec0::/10`, został w treści dokumentu [RFC3879] uznany za niezalecany (*deprecated*) ze względu na niejednoznaczność interpretacji, wynikającą z możliwości używania tego samego adresu w różnych miejscach sieciowych i nieprecyzyjności samej definicji „miejsca sieciowego".

Adresy IPv6 lokalne dla łącza (i niektóre adresy globalne) wykorzystują **identyfikatory interfejsów** (IID — *Interface IDentifiers*) jako podstawę dla przypisywania adresów typu unicast. Identyfikatory te znajdują swe odzwierciedlenie w najmniej znaczących bitach adresu IPv6, z wyjątkiem adresów rozpoczynających się od bitów 000, które muszą być unikalne w ramach sieci o tym samym prefiksie. Każdy IID ma długość 64 bitów i generowany jest na podstawie adresu MAC łącza danych w formacie **zmodyfikowanego**

EUI-64 (patrz [EUI64]) bądź też na postawie generatora liczb (pseudo)losowych, mającego w zamierzeniu zapewnić pewien stopień prywatności i ochronę przed śledzeniem adresów MAC (patrz rozdział 6.).

Akronim EUI to skrót od *extended unique identifier* — „unikatowy rozszerzony identyfikator". Zgodnie ze standardami IEEE identyfikator EUI-64 składa się z dwóch części: 24 mniej znaczące bity to identyfikator OUI (*Organizationally Unique Identifier* — unikatowy identyfikator organizacji), pozostałe 40 bitów to **identyfikator rozszerzający** (*extension identifier*) wybierany dowolnie w ramach organizacji identyfikowanej przez OUI. Identyfikatory OUI zarządzane są centralnie przez urząd rejestracyjny IEEE ([IEEERA]). Administrowanie identyfikatorami EUI może odbywać się w sposób uniwersalny (są to tzw. *universally administered EUI*) albo w sposób lokalny (*locally administered EUI*). W kontekście Internetu większość EUI należy do pierwszej z wymienionych kategorii.

Wiele interfejsów sieciowych zgodnych ze standardami IEEE — należy do nich np. Ethernet — wykorzystywało przez lata krótsze, 48-bitowe identyfikatory EUI. Zasadniczo nie różnią się one od EUI-64 niczym innym oprócz długości — co przedstawiono na rysunku 2.7.



Rysunek 2.7. *Zdefiniowane przez IEEE formaty EUI-48 oraz EUI-64, używane w adresach IPv6 do tworzenia identyfikatorów interfejsów, przez zanegowanie bitu u. U dołu — sposób konwersji między formatami EUI*

W obu formatach pierwsze trzy bajty to identyfikator OUI[2]; dwa najmniej znaczące bity pierwszego bajta mają specjalne znaczenie. Bit o wadze 2, oznaczany u, określa sposób administrowania EUI: 0 oznacza administrowanie uniwersalne, 1 administrowanie lokalne. Skrajny bit (ten o wadze 1), oznaczany g, służy do rozróżnienia między adresem unicast (0) a adresem multicast (1). Przyjmiemy chwilowo, że bit g ma wartość 0.

Przekształcenia EUI-48 na format EUI-64 dokonuje się, kopiując trzy początkowe bajty (0 – 2) pierwszego na drugi, następnie kopiując trzy kolejne bajty pierwszego (3 – 5) na trzy ostatnie (najmniej znaczące) bajty drugiego (5 – 7) i ostatecznie wypełniając bajty 3 – 4 drugiego wartością (szesnastkowo) FFFE. Przykładowy EUI-48 o postaci 00-11-22-33-44-55 po konwersji na EUI-64 przyjmie więc postać 00-11-22-FF-FE-33-44-55. Ostatnim krokiem przekształcenia EUI-64 na identyfikator IID jest zanegowanie bitu u, w wyniku czego otrzymujemy 02-11-22-FF-FE-33-44-55.

[2] Konsekwentnie przyjmujemy konwencję *big endian* — pod najmniejszym adresem znajduje się najbardziej znaczący bajt, adresy bajtów na rysunku wzrastają od lewej do prawej — *przyp. tłum.*

W sytuacji gdy producent nie przyporządkował urządzeniu adresu EUI-48, lecz z urządzeniem tym związany jest jakiś inny adres (np. AppleTalk), identyfikator IID otrzymuje się, uzupełniając lewostronnie ów adres bitami zerowymi. Dla interfejsów pozbawionych jakichkolwiek identyfikatorów (np. tuneli czy łączy szeregowych) IID tworzy się zwykle na podstawie innego interfejsu należącego do tego samego komputera (lub jakiegoś innego unikatowego identyfikatora związanego z tym komputerem). W ostateczności pozostaje zawsze opcja ręcznego przypisania IID.

2.3.6.1. Przykłady

Za pomocą linuksowego polecenia `ifconfig` można prześledzić opisaną procedurę generowania adresów IPv6 lokalnych dla łącza:

```
Linux% ifconfig eth1
eth1 Link encap:Ethernet HWaddr 00:30:48:2A:19:89
inet addr:12.46.129.28 Bcast:12.46.129.127
Mask:255.255.255.128
inet6 addr: fe80::230:48ff:fe2a:1989/64 Scope:Link
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:1359970341 errors:0 dropped:0 overruns:0 frame:0
TX packets:1472870787 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:4021555658 (3.7 GiB) TX bytes:3258456176 (3.0 GiB)
Base address:0x3040 Memory:f8220000-f8240000
```

Widzimy tu przekształcanie ethernetowego adresu (48-bitowego) `00:30:48:2A:19:89` na adres IPv6. Najpierw adres 48-bitowy konwertowany jest do postaci EUI-64, co daje w wyniku `00:30:48:ff:fe:2a:19:89`. Negując bit `u`, otrzymujemy `02:30:48:ff:fe:2a:19:89`. Ostatni krok to poprzedzenie otrzymanego wyniku prefiksem `fe80::/10` reprezentującym adres lokalny dla łącza (patrz podrozdział 2.5) — otrzymujemy tym samym adres IPv6 `fe80::230:48ff:fe2a:1989`. Prefiks `/64` wynika z wyodrębniania numeru sieci standardowego dla adresów IPv6 otrzymywanych na podstawie identyfikatorów IID, zgodnie z wymogami dokumentu [RFC4291].

Oto inny interesujący przykład, tym razem z systemu Windows. Widzimy tu specjalny **punkt końcowy tunelu** wykorzystywanego do przesyłania pakietów IPv6 przez sieci obsługujące jedynie IPv4:

```
c:\> ipconfig /all
     ...
Tunnel adapter Automatic Tunneling Pseudo-Interface:
Sufiks DNS konkretnego połączenia : foo
Opis. . . . . . . . . . . . . . . : Automatic Tunneling Pseudo-Interface
Adres fizyczny . . . . . . . . . . : 0A-99-8D-87
DHCP włączone . . . . . . . . . . : Nie
Adres IPv6 . . . . . . . . . . . . : fe80::5efe:10.153.141.135%2
Brama domyślna . . . . . . . . . :
Serwery DNS . . . . . . . . . . . : fec0:0:0:ffff::1%2
                                    fec0:0:0:ffff::2%2
                                    fec0:0:0:ffff::3%2
NetBIOS przez Tcpip . . . . . . . : Wyłączony
```

Powyższy listing ilustruje użycie specjalnego interfejsu tunelowania o nazwie ISATAP (patrz [RFC5214]). Tak zwany „adres fizyczny" (`0A-99-8D-87`) to nic innego jak zakodowany szesnastkowo adres IPv4 `10.153.141.135`. Obecny w adresie IPv6 identyfikator OUI równy `00-00-5E` jest jednym z przypisanych organizacji IANA (patrz [IANA]); występuje tu w połączeniu z wartością szesnastkową `fe`, oznaczającą wbudowany (*embedded*) adres IPv4. Całość poprzedzona jest prefiksem `fe80::/10` oznaczającym lokalność dla łącza. W ten sposób otrzymujemy ostatecznie adres IPv6 `fe80::5efe:10.153.141.135`; przyrostek %2 nazywany jest w Windows „identyfikatorem strefy" (*zone ID*) i reprezentuje indeks konkretnego interfejsu w zbiorze wszystkich interfejsów komputera używającego danego adresu IPv6. Adresy IPv6 są najczęściej generowane przez proces automatycznej konfiguracji, którym zajmiemy się dokładniej w rozdziale 6.

2.4. CIDR i agregacja

Wprowadzenie w latach 90. ubiegłego wieku adresowania podsieci jako jednego z mechanizmów łagodzących problemy ze skalowalnością Internetu nie mogło — oczywiście — rozwiązać tych problemów na dłuższą skalę. Trzy problemy wydawały się być tak dotkliwe, że wymagały podjęcia natychmiastowych działań.

1. W roku 1994 wykorzystano już ponad połowę adresów IP klasy B; według przewidywań, w roku 1995 ich pula powinna się całkowicie wyczerpać.
2. Zbiór możliwych 32-bitowych adresów IP ewidentnie okazywał się zbyt mały na to, by w oparciu o niego możliwe było w ogóle funkcjonowanie Internetu po roku 2000.
3. Systematycznie rosnąca liczba pozycji w globalnej tablicy trasowania (w roku 1995 było ich około 65 000) powodowała systematyczną degradację wydajności trasowania. Każde przyłączenie nowej sieci klasy A, B lub C skutkowało dodaniem nowej pozycji do tej tabeli.

By stawić czoło powyższym problemom, jeszcze w 1992 roku powołano do życia grupę roboczą ROAD (od *ROuting and ADdresing* — trasowanie i adresowanie). Zakwalifikowała ona problemy nr 1. i 3. jako wymagające natychmiastowej interwencji, pozostawiając problem nr 2. do rozwiązania w dłuższej perspektywie czasowej. Jako doraźny cel natychmiastowych działań określono rezygnację z podziału adresów IP na klasy oraz dokonywanie ich przydziału w sposób umożliwiający agregowanie. Jedynym możliwym rozwiązaniem problemu nr 2. okazało się natomiast zdefiniowanie i wdrożenie nowej wersji protokołu IP — IPv6.

2.4.1. Prefiksy

Jako sposób na powstrzymanie tempa wyczerpywania się zasobu dostępnych adresów IP — szczególnie tych klasy B — przyjęto rezygnację ze sztywnego podziału adresów na klasy na rzecz zaimplementowania na globalnym poziomie Internetu mechanizmu podobnego do VLSM; wymagało to — oczywiście — zmian w globalnym systemie trasowania, a konkretnie wzbogacenie tego systemu o funkcję **bezklasowego trasowania międzydomenowego** (*Classless Inter-Domain Routing*, w skrócie CIDR) opisanego w dokumencie [RFC4632]. Dzięki temu niewykorzystane jeszcze ciągle przedziały adresów klas B i C

mogły być odtąd przydzielane na potrzeby miejsc sieciowych o wielkościach pośrednich między tymi klasami, czyli z maksymalną liczbą hostów plasującą się między 256 a 65 536.

Rzecz jasna, rezygnacja z „klasowości" adresów oznacza utratę informacji o liczbie bitów tworzących numer miejsca sieciowego; dotychczas informacja ta wynikała bezpośrednio z klasy adresu, określonej przez jego początkowe bity, obecnie musi być dostarczona jawnie w postaci maski (analogicznej do maski podsieci), w tym kontekście nazywanej **maską CIDR**. W przeciwieństwie do masek podsieci, ma ona znaczenie globalne i przetwarzana jest przez system trasujący Internetu. Zapisuje się ją jako liczbę bitów składających się na numer miejsca sieciowego, oddzielając od adresu IP znakiem ukośnika („/"); kombinacja adresu IP i maski CIDR, nazywana popularnie **prefiksem**, wykorzystywana jest w obu wersjach protokołu — IPv4 i IPv6. W tabeli 2.6 przedstawiono kilka przykładowych prefiksów dla adresów w obu wersjach; bity objęte prefiksem obwiedzione są ramką.

Tabela 2.6. *Przykłady prefiksów IPv4 i IPv6 wraz z odpowiadającymi im zakresami adresów*

Prefiks	Reprezentacja binarna	Zakres adresów
0.0.0.0/0	00000000 00000000 00000000 00000000	0.0.0.0 – 255.255.255.255
128.0.0.0/1	1 0000000 00000000 00000000 00000000	128.0.0.0 – 255.255.255.255
128.0.0.0/24	10000000 00000000 00000000 00000000	128.0.0.0 – 128.0.0.255
198.128.128.192/27	11000110 10000000 10000000 11000000	198.128.128.192 – 198.128.128.223
165.195.130.107/32	10100101 11000011 10000010 01101011	165.195.130.107
2001:db8::/32	0010000000000001 0000110110111000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000 0000000000000000	2001:db8::-2001:db8:ffff:ffff

Pozostałe bity adresu mogą być ustawione dowolnie — wszystkie ich kombinacje wyczerpują zbiór adresów o danym prefiksie. Oczywiście, im krótszy prefiks, tym więcej możliwych adresów. Warto przy okazji zauważyć, że schemat adresowania klasowego znakomicie wpisuje się w strukturę prefiksów: przykładowo adres klasy C, identyfikujący miejsce sieciowe 192.125.3.0, można zapisać jako 192.125.3.0/24 lub 192.125.3/24, podobnie adresy klasy A i B można zapisywać przy użyciu maski CIDR (odpowiednio /8 i /16.

2.4.2. Agregowanie prefiksów

Rezygnacja z tradycyjnego podziału adresów IPv4 na klasy powoduje, iż adresami tymi gospodaruje się bardziej oszczędnie, co przyczynia się do pewnego złagodzenia jedynie pierwszego z trzech palących problemów podniesionych na początku podrozdziału 2.4. W najmniejszym jednak stopniu nie dotyka trzeciego problemu, czyli sukcesywnie rosnącego rozmiaru globalnej tablicy trasowania i wynikającego stąd systematycznego spadku wydajności Internetu.

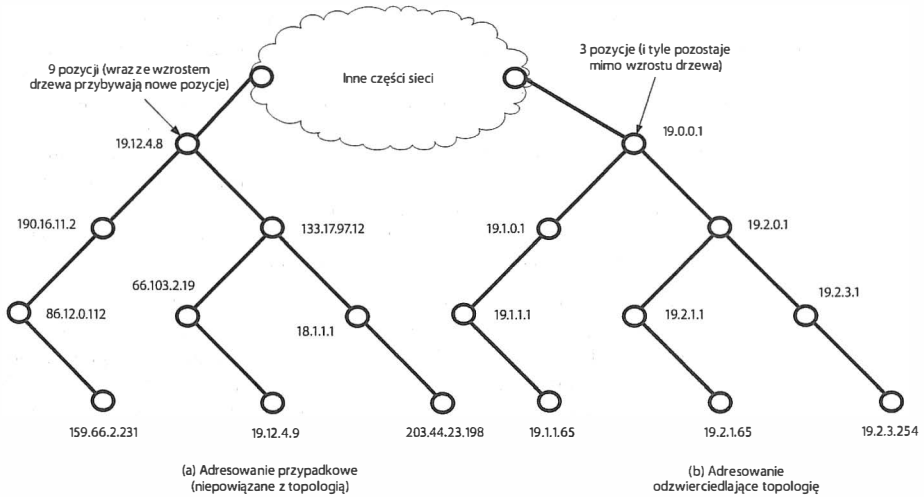
Router, po odczytaniu adresu docelowego z otrzymanego pakietu IP, przegląda swą tablicę trasowania w celu określenia routera stanowiącego następny „przeskok" wspomnianego pakietu na jego trasie: pozycje tablicy przeglądane są aż do znalezienia pozycji „pasującej" do rzeczonego adresu (bądź do stwierdzenia, że taka pozycja w tablicy nie istnieje). Przypomina to nieco podróż samochodem po sieci dróg, w której na każdym skrzyżowaniu znajduje się zestaw drogowskazów pokazujących następne skrzyżowanie na trasie do *każdego możliwego* miejsca przeznaczenia. Jeżeli teraz skonfrontujemy to z ogromem istniejących skrzyżowań (analizując np. mały tylko fragment mapy drogowej Polski), uzyskamy zgrubny obraz skali problemu, jakim wydajność trasowania w Internecie okazywała się być już na początku lat 90. ubiegłego wieku.

Jest więc zrozumiałe, że w poszukiwaniu rozwiązania tegoż problemu (co ciekawe, jeszcze na długo przedtem, zanim faktycznie dał o sobie znać) zaproponowano kilka technik umożliwiających znaczącą redukcję rozmiarów tablic trasowania, przy jednoczesnym zachowaniu kryterium najkrótszej możliwej drogi do wszystkich możliwych miejsc docelowych w Internecie. Najbardziej znaną techniką tego rodzaju jest podejście zaproponowane przez Kleinrocka i Kamouna jeszcze u schyłku lat 70. dwudziestego wieku, w związku ze studiami nad **trasowaniem hierarchicznym** (*hierarchical routing* — patrz [KK77]). Odkryli oni mianowicie, że jeśli topologia sieci zostanie zaaranżowana jako drzewo[3], a metoda przypisywania adresów podporządkowana zostanie tej topologii, liczebność pozycji w tablicach trasowania poszczególnych routerów może zostać wydatnie zredukowana.

Spójrzmy na rysunek 2.8 — kółka oznaczają tu routery, a linie reprezentują łącza między routerami. Obie przedstawione na rysunku sieci mają topologię drzewiastą, różni je natomiast metoda przyporządkowywania adresów IP poszczególnym węzłom. W sieci po lewej stronie jest to przyporządkowanie dość przypadkowe, w każdym razie niepozostające w żadnym wyraźnym związku z lokalizacją odnośnych węzłów. W sieci po stronie prawej jest całkiem odwrotnie — adres węzła wynika wprost z jego relatywnego położenia w drzewie.

Korzeniem drzewa sieci lewostronnej jest router o adresie 19.12.4.8; aby możliwe było osiągnięcie każdego z ośmiu innych węzłów drzewa — 190.16.11.2, 86.12.0.112, 159.66.2.231, 133.17.97.12, 66.103.2.19, 18.1.1.1, 19.12.4.9 oraz 203.44.23.198 — wspomniany router musi dysponować informacją o lokalizacji każdego z nich, czyli ośmioma pozycjami w tablicy trasowania; dodatkowa, dziewiąta pozycja reprezentuje wszystkie węzły zlokalizowane w ramach chmury oznaczonej jako „Inne części sieci". Dla odróżnienia, w sieci prawostronnej zapewnić można osiągalność wszystkich węzłów, zapisując w routerze 19.0.0.1 tylko trzy pozycje: zauważmy, że w lewym poddrzewie tej sieci adresy wszystkich węzłów rozpoczynają się od 19.1, zaś w prawym poddrzewie — od 19.2. W „szczytowym" routerze konieczne są więc dwie pozycje wskazujące następny „przeskok": 19.1.0.1 dla wszystkich lokalizacji o adresach rozpoczynających się od 19.1 i 19.2.0.1 dla wszystkich lokalizacji o adresach rozpoczynających się od 19.2. Dodatkowa, trzecia pozycja, reprezentująca wszystkie inne lokalizacje, reprezentuje wspomnianą wcześniej chmurę. Tak właśnie wygląda istota trasowania hierarchicznego opisanego w [KK77].

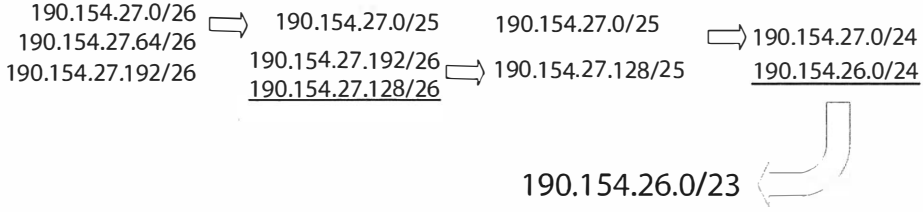
[3] W teorii grafów „drzewem" nazywamy graf spójny pozbawiony cykli; w takim grafie między *każdą parą* wierzchołków istnieje *dokładnie jedna* droga (rozumiana jako ciąg sąsiadujących krawędzi). W przełożeniu na sieć, której routery są analogami wierzchołków, a łącza między routerami analogami krawędzi grafu, oznacza to istnienie dokładnie jednej prostej (nieduplikowanej) ścieżki między dowolnymi dwoma routerami.



Rysunek 2.8. *W sieci o topologii drzewiastej przypisywanie adresów może odbywać się w specjalny sposób umożliwiający minimalizowanie informacji trasującej („stanu") przechowywanej w routerze. Jeśli adresy przypisywane są w sposób przypadkowy (jak w sieci (a)), nie można zagwarantować najkrótszych ścieżek trasowania bez przechowywania w routerze informacji o rozmiarze proporcjonalnym do liczby uwzględnianych węzłów. Gdy jednak przypisywanie adresów odbywa się z uwzględnieniem topologii sieci (jak w sieci (b)), wspomniana informacja ma znacznie mniejszą objętość, jednakże zmiana tej topologii wymaga zwykle ponownego przyporządkowania adresów*

W kontekście Internetu trasowanie hierarchiczne może zostać wykorzystane w nieco inny sposób, mianowicie w formie procedury agregowania tras, przyczyniającej się do redukowania rozmiaru tablicy trasowania. Istotą tej procedury jest redukowanie dwóch sąsiadujących numerycznie prefiksów[4] do jednego prefiksu, zwanego *prefiksem zagregowanym* lub (krótko) *agregatem*. Spójrzmy na rysunek 2.9. Z lewej strony znajdują się trzy prefiksy; dwa z nich — 190.154.27.0/26 i 190.154.27.64/26 — są numerycznie sąsiadujące, więc mogą zostać zagregowane do pojedynczego prefiksu 190.154.27.0/25 wyznaczającego dokładnie taki sam zbiór adresów IP (190.154.27.0 – 190.154.27.127), jaki sumarycznie wyznaczany był przez oba prefiksy wejściowe (190.154.27.0 – 190.154.27.63 plus 190.154.27.64 – 190.154.27.127). Na rysunku agregacja oznaczona jest strzałką. Trzyelementowy zbiór prefiksów został zredukowany do dwuelementowego; gdy do tego zbioru dołączymy kolejny prefiks (na rysunku podkreślony) — 190.154.27.128/26 — pojawia się znowu możliwość jego agregacji z prefiksem 190.154.27.192/26, w wyniku czego otrzymujemy prefiks zagregowany 190.154.27.128/25. Ten ostatni jest sąsiadujący numerycznie z prefiksem 190.154.27.0/25 — wynikiem kolejnej agregacji jest 190.154.27.0/24. W ostatnim kroku scenariusza przedstawionego na rysunku pojawia się nowy (podkreślony) prefiks 190.154.26.0/24, który zagregowany zostaje z sąsiadem do prefiksu 190.154.26.0/23. Tak oto pięć różnych prefiksów zagregowaliśmy do jednego, sumarycznie wyznaczającego ten sam zbiór adresów.

[4] Dwa prefiksy nazywamy „sąsiadującymi numerycznie", jeżeli różnią się wyłącznie ostatnim bitem. Cytowane w tekście prefiksy 190.154.27.0/26 i 190.154.27.64/26 są przykładem takiego sąsiedztwa, bowiem ich binarne formy to 1011 1110 1001 1010 0001 1011 00 i 1011 1110 1001 1010 0001 1011 01 — *przyp. tłum.*



Rysunek 2.9. *Przykład agregowania prefiksów; strzałka symbolizuje agregację, podkreśleniem wyróżniono nowe prefiksy pojawiające się w poszczególnych krokach. W pierwszym kroku agregacja 190.154.27.0/26 i 190.154.27.64/26 daje 190.154.27.0/25; prefiks 190.154.27.192/26 nie sąsiaduje numerycznie z żadnym z partnerów ani z wynikiem ich agregacji, nie można go więc zagregować. Możliwość jego agregacji pojawia się jednak w drugim kroku, z nowym prefiksem 190.154.27.128/26. W trzecim kroku agregowane są wyniki dwóch poprzednich agregacji; rezultat agregowany jest w kroku czwartym z nowo dodanym prefiksem — ostatecznie otrzymujemy pojedynczy prefiks jako wynik agregacji pięciu oddzielnych*

2.5. Adresy specjalnego znaczenia

W zbiorze adresów obu formatów — IPv4 i IPv6 — znajdują się adresy specjalnego przeznaczenia, niepełniące roli adresów unicast. Prefiksy identyfikujące poszczególne zakresy tych adresów dla protokołu IPv4 zebrane zostały w tabeli 2.7 (zaczerpniętej z [RFC5735]). Analogiczne zestawienie dla adresów IPv6 znajduje się w tabeli 2.8, zaczerpniętej z [RFC5156].

Tabela 2.7. *Adresy IPv4 specjalnego przeznaczenia (stan definicji ze stycznia 2010)*

Prefiks	Zastosowanie adresów	Na podstawie dokumentu
0.0.0.0/8	Hosty w sieci lokalnej. Adresy z tego zakresu mogą występować wyłącznie jako adresy źródłowe.	[RFC1122]
10.0.0.0/8	Sieci prywatne (intranety). Adresy z tego zakresu nigdy nie są widoczne publicznie w Internecie.	[RFC1918]
127.0.0.0/8	Pętla sprzężenia zwrotnego (*loopback*) wewnątrz hosta. Zwykle używany jest tylko adres 127.0.0.1.	[RFC1122]
169.254.0.0/16	Adresy „lokalne dla łącza" — używane tylko na pojedynczym łączu i zazwyczaj przydzielane dynamicznie (patrz rozdział 6.).	[RFC3927]
172.16.0.0/12	Sieci prywatne (intranety). Adresy z tego zakresu nigdy nie są widoczne publicznie w Internecie.	[RFC1918]
192.0.0.0/24	Adresy zarezerwowane przez IANA na potrzeby protokołów IETF.	[RFC5736]
192.0.2.0/24	Adresy testowe TEST-NET-1 dopuszczone do używania w dokumentacji. Adresy z tego zakresu nigdy nie są widoczne publicznie w Internecie.	[RFC5737]
192.88.99.0/24	Adresy anycast wykorzystywane na potrzeby przekaźników 6to4.	[RFC3068]
192.168.0.0/16	Sieci prywatne (intranety). Adresy z tego zakresu nigdy nie są widoczne publicznie w Internecie.	[RFC1918]

Tabela 2.7. *Adresy IPv4 specjalnego przeznaczenia (stan definicji ze stycznia 2010) — ciąg dalszy*

Prefiks	Zastosowanie adresów	Na podstawie dokumentu
198.18.0.0/15	Testy wydajnościowe i benchmarki.	[RFC2544]
198.51.100.0/24	Adresy testowe TEST-NET-2 dopuszczone do używania w dokumentacji.	[RFC5737]
203.0.113.0/24	Adresy testowe TEST-NET-3 dopuszczone do używania w dokumentacji.	[RFC5737]
224.0.0.0/4	Adresy multicast — dawna klasa D. Wykorzystywane wyłącznie jako adresy docelowe.	[RFC5771]
240.0.0.0/4	Dawna klasa E — adresy aktualnie nieużywane, z wyjątkiem 255.255.255.255.	[RFC1112]
255.255.255.255/32	Rozgłaszanie lokalne (*limited broadcast*).	[RFC0919] [RFC0922]

Tabela 2.8. *Adresy IPv6 specjalnego przeznaczenia (stan definicji z kwietnia 2008)*

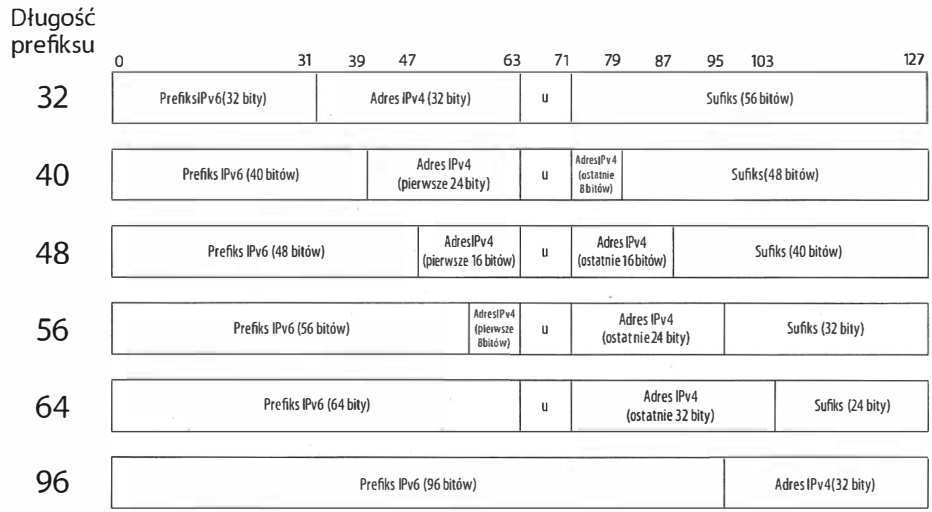
Prefiks	Zastosowanie adresów	Na podstawie dokumentu
::/0	Domyślna zawartość pozycji w tabeli trasowania; adres tej postaci nie jest używany.	[RFC5156]
::/128	Adres nieokreślony, może być użyty w roli adresu źródłowego.	[RFC4291]
::1/128	Pętla sprzężenia zwrotnego (*loopback*) wewnątrz hosta; adres nie jest używany w datagramach wychodzących na zewnątrz hosta.	[RFC4291]
::ffff:0:0/96	Mapowany adres IPv4; adres tej postaci nigdy nie pojawia się w nagłówku pakietu, wykorzystywany jest wyłącznie wewnątrz hosta.	[RFC4291]
::{adres IPv4}/96	Postać kompatybilna adresu IPv4; niezalecana do użytku.	[RFC4291]
2001::/32	Adresy tunelowania Teredo.	[RFC4380]
2001:10::/28	Prefiks dla adresów protokołu ORCHID (*Overlay Routable Cryptographic Hash Identifiers*). Adresy z tego zakresu nigdy nie są widoczne publicznie w Internecie.	[RFC4843]
2001:db8::/32	Adresy przeznaczone jedynie do użytku w przykładach i dokumentacjach, nigdy nie są widoczne publicznie w Internecie.	[RFC3849]
2002::/16	Adresy wykorzystywane w automatycznym tunelowaniu 6to4.	[RFC3056]
3ffe::/16	Adresy eksperymentalne sieci 6bone, nieprzeznaczone do innego użytku.	[RFC3701]
5f00::/16		[RFC3701]
fc00::/7	Unikatowe, lokalne adresy unicast, niewykorzystywane w publicznym Internecie.	[RFC4193]
fe80::/10	Adresy unicast lokalne dla łącza.	[RFC4291]
ff00::/8	Adresy multicast, mogą występować wyłącznie jako adresy docelowe.	[RFC4291]

Adresy nieoznaczone jako multicast, zarezerwowane lub specjalne, mogą być wykorzystywane jako adresy unicast, jednak niektóre ich zakresy (10/8, 172.16/12, i 192.168/16 w IPv4 oraz fc00::/7 w IPv6) przeznaczone są na użytek sieci prywatnych, czyli na potrzeby współpracy hostów i routerów wewnątrz tych sieci; w globalnym Internecie adresy te nie mają ustalonego znaczenia, nazywane są więc często adresami **nietrasowalnymi** (*nonroutable*). Ich zarządzanie powierzone jest w całości administratorom wspomnianych sieci prywatnych, gdzie wykorzystywane są często w połączeniu z mechanizmem **translacji adresów sieciowych** (NAT — *Network Address Translation*), realizowanym przez router łączący sieć lokalną z Internetem (szczegółami NAT zajmiemy się w rozdziale 7.).

2.5.1. Translatory IPv4/IPv6

Często użyteczna okazuje się możliwość przeprowadzania translacji adresów IPv4 na IPv6 (patrz [RFC6127]). Opracowano framework takiej translacji dla adresów unicast ([RFC6144]), obecnie trwają prace nad analogicznym frameworkiem dla multicastingu ([IDv-4v6mc]). Podstawowym założeniem wspomnianej translacji jest jej algorytmiczny charakter: w dokumencie [RFC6052] opisane jest wykonywanie tejże translacji dla adresów unicast, na bazie „dobrze znanego" prefiksu 64:ff9b::/96 i innych prefiksów służących temu samemu celowi.

Idea owej translacji jest zgoła nieskomplikowana: adres IPv4 zostaje **wbudowany** (*embedded*) w ramy adresu IPv6, w jednym z sześciu wariantów, różniących się długością prefiksu — 32, 40, 48, 56, 64 lub 96 bitów — co szczegółowo przedstawiono na rysunku 2.10.



Rysunek 2.10. *Różne warianty wbudowywania adresu IPv4 w ramy adresu IPv6. „Dobrze znany" prefiks 64:ff9b::/96 może zostać wykorzystany do automatycznej konwersji adresu unicast IPv4 na adres IPv6*

Prefiks rozpoczynający wynikowy adres IPv6 może być bądź to „dobrze znanym" prefiksem `64:ff9b::/96`, bądź innym prefiksem unikatowym dla organizacji wdrażającej translację. Zasada wbudowywania jest we wszystkich wariantach taka sama. Prefiks zostaje skonkatenowany z przedmiotowym adresem IPv4, przy czym (dla zachowania zgodności z identyfikatorami zdefiniowanymi w [RFC4291]) bity 64 – 71 muszą być wyzerowane; jeśli bity te plasują się pośrodku wspomnianego adresu IPv4 (jest tak przy prefiksie 40-, 48- i 56-bitowym), należy *wstawić w środek tegoż adresu* ciąg 8 bitów zerowych (na rysunku 2.10 oznaczony u). Całość dopełniamy ciągiem bitów zerowych, zwanym **sufiksem**, do długości 128 bitów (wyjątek stanowi wariant z prefiksem 96-bitowym, gdzie sufiks nie występuje). Wariant z prefiksem `64:ff9b::/96` może być też użyty do realizacji automatycznego mapowania adresów IPv4 na IPv6 (o czym wspominaliśmy w podrozdziale 2.2 i co opisane jest w dokumencie [RFC4291]), przykładowo adres `198.51.100.16` odwzorowany zostaje wówczas na adres `64:ff9b::198.51.100.16`.

2.5.2. Adresy grupowe (multicast)

Adresy multicast występują w obu wersjach protokołu — IPv4 oraz IPv6. Adres multicast, zwany także **adresem grupowym**, identyfikuje *grupę* interfejsów hostów, nie pojedynczy interfejs. Ogólnie rzec biorąc, zbiór interfejsów identyfikowanych przez dany adres nazywa się **zakresem** (*scope*) tegoż adresu. Zakres może obejmować pojedynczy węzeł (*node-local*), określoną podsieć (*link-local*), określone miejsce sieciowe (*site-local*) lub cały Internet (*global*). Specjalnym zakresem jest **zakres administracyjny** (*administrative*) obejmujący te obszary sieci, w których routery konfigurowane są ręcznie przez administratora. Administrator może zdefiniować określone routery jako **brzegowe dla zakresu** (*admin-scope boundaries*), wskutek czego ruch multicast generowany w określonej grupie nie będzie wydostawał się na zewnątrz tej grupy. Zauważmy, że zakres tożsamy z miejscem sieciowym oraz zakres administracyjny mają sens jedynie w odniesieniu do adresów multicast.

Implementacja protokołów TCP/IP w każdym hoście Internetu zapewnia jego przyłączenie do grupy multicast oraz odłączenie od tej grupy. Host zamierzający wysłać datagram do określonej grupy wpisuje do tego datagramu swój adres unicast jako źródłowy oraz adres multicast tej grupy jako docelowy. Datagram powinien wówczas dotrzeć do każdego hosta należącego do wspomnianej grupy; host nadawca nie ma żadnej kontroli nad docieraniem kopii datagramu do poszczególnych hostów adresatów (chyba że odsyłają one potwierdzenie otrzymania); co więcej, nie można nawet określić *liczby* adresatów. Multicasting w takiej formie, czyli uniezależnienie trasowania datagramu od adresu źródłowego, określany jest akronimem ASM, od *any-source multicast*. Host zamierzający dołączyć do danej grupy czyni to wyłącznie na podstawie jej adresu multicast. Na gruncie nowszego podejścia, określanego akronimem SSM, od *source-specific multicast*, opisanego w dokumentach [RFC3569] [RFC4607], dla każdej grupy wyznaczony jest konkretny nadawca, czyli ustalony źródłowy adres IP (patrz errata dokumentu [RFC4607]). Host dołączający do grupy czyni to przez podanie adresu *kanału*, łączącego adres grupy *oraz* adres IP nadawcy. SSM opracowano w celu zniwelowania pewnych uciążliwych komplikacji związanych z wdrażaniem modelu ASM. Mimo iż generalnie żadna z dwóch wymienionych form multicastingu nie jest obecnie szeroko dostępna w globalnym Internecie, to SSM wydaje się bardziej prawdopodobnym kandydatem do tej roli.

W czasie kilkunastu ostatnich lat na forum społeczności Internetu poczyniono wiele wysiłków zmierzających do zrozumienia i upowszechnienia multicastingu, czego wyrazem jest choćby duża liczba opracowanych w związku z tym protokołów. Nie sposób więc opisać szczegółów tego zagadnienia w ograniczonych ramach tej książki; zainteresowanym czytelnikom możemy polecić pozycję [IMR02]. Szczegóły funkcjonowania lokalnego multicastingu opisujemy w rozdziale 9., obecnie zajmiemy się natomiast formatami i znaczeniem adresów multicast w wersjach IPv4 i IPv6.

2.5.3. Multicasting w IPv4

Na potrzeby multicastingu zarezerwowano klasę D adresów IPv4, czyli przedział 224.0.0.0 – 239.255.255.255. Dysponując 28 bitami, można określić 2^{28} = 268 435 456 grup hostów. Przestrzeń ta została podzielona na kilka głównych sekcji (bloków) w zależności od sposobu przydzielania i funkcjonowania w kontekście trasowania (patrz [IP4MA]) — podział ten przedstawiamy w tabeli 2.9.

Tabela 2.9. *Podział adresów IPv4 klasy D na bloki*

Zakres (włącznie)	Zastosowanie adresów	Na podstawie dokumentu
224.0.0.0 – 224.0.0.255	Sterowanie ruchem w ramach sieci lokalnej, pakiety z takimi adresami docelowymi nie są forwardowane.	[RFC5771]
224.0.1.0 – 224.0.1.255	Sterowanie ruchem w ramach sieci globalnej, normalne forwardowanie.	[RFC5771]
224.0.2.0 – 224.0.255.255	Blok ad hoc I.	[RFC5771]
224.1.0.0 – 224.1.255.255	Zarezerwowane.	[RFC5771]
224.2.0.0 – 224.2.255.255	SDP/SAP.	[RFC4566]
224.3.0.0 – 224.4.255.255	Blok ad hoc II.	[RFC5771]
224.5.0.0 – 224.255.255.255	Zarezerwowane.	[IP4MA]
225.0.0.0 – 231.255.255.255	Zarezerwowane.	[IP4MA]
232.0.0.0 – 232.255.255.255	Multicasting ze specyficznym źródłem (SSM).	[RFC4607] [RFC4608]
233.0.0.0 – 233.251.255.255	GLOP.	[RFC3180]
233.252.0.0 – 233.255.255.255	Blok ad hoc III (adres 233.252.0.0/24 jest zarezerwowany dla dokumentacji).	[RFC5771]
234.0.0.0 – 234.255.255.255	Adresy bazujące na prefiksie unicast IPv4.	[RFC6034]
235.0.0.0 – 238.255.255.255	Zarezerwowane.	[IP4MA]
239.0.0.0 – 239.255.255.255	Zakres administracyjny.	[RFC2365]

Bloki obejmujące adresy od 224.0.0.0 do 224.255.255.255 przeznaczone są do użytku określonych protokołów lub organizacji i przydzielane są z inicjatywy IANA lub IETF.

I tak blok sterowania ruchem w ramach sieci lokalnej obejmuje adresy, które umieszczone w datagramach jako docelowe powodują ograniczenie ich zasięgu do sieci lokalnej nadawcy — pakiety te nigdy nie są forwardowane przez routery multicastingu. Adres 224.0.0.1 reprezentuje grupę wszystkich hostów w sieci. Blok sterowania ruchem w ramach sieci globalnej ma podobne znaczenie, jednakże pakiety opatrzone jego adresami docelowymi przeznaczone są do forwardowania przez łącze lokalne. Przykładem grupy z tego bloku jest grupa reprezentowana przez adres 224.0.1.1, wykorzystywana przez **sieciowy protokół czasu** (NTP — *Network Time Protocol*, patrz [RFC5905]).

Pierwszy blok ad hoc obejmuje adresy niewchodzące w skład żadnego z dwóch bloków sterowania ruchem. Większość tych adresów wykorzystywana jest przez usługi komercyjne, z których większość nie wymaga globalnego przydzielania adresów; niektóre adresy z tego bloku mogą być wykorzystywane (zwracane) w ramach adresowania GLOP[5] (o tym dalej, w tym rozdziale).

Adresy bloku SDP/SAP używane są przez aplikacje, takie jak narzędzie katalogu sesji (SDR — *Session Directory Tool*), wysyłające oznajmienia o rozpoczęciu sesji multicastingu za pomocą protokołu SAP (*Session Announcement Protocol* — patrz [RFC2974]). Nowy **protokół opisu sesji** (SDP — *Session Description Protocol* — patrz [RFC4566]), stanowiący wcześniej komponent protokołu SAP, wykorzystywany jest nie tylko w kontekście multicastingu IP, lecz także w połączeniu z innymi mechanizmami, na potrzeby opisu sesji multimedialnych.

Pozostałe bloki definiowano sukcesywnie później, w miarę ewoluowania multicastingu IP. I tak blok SSM stosowany jest przez aplikacje wykorzystujące (opisane wcześniej) kanały SSM. Adresy bloku GLOP bazują na numerach **systemów autonomicznych** (AS — *Autonomous Systems*) używanych w Internecie do agregowania tras w ruchu między dostawcami oraz realizacji wspólnych założeń trasowania. Każdy AS ma swój unikatowy numer — obecnie 32-bitowy, kiedyś 16-bitowy (patrz [RFC4893]). Adres GLOP tworzy się, umieszczając 16-bitowy numer AS w drugim i trzecim bajcie (w pierwszym jest już liczba 233), do dyspozycji pozostaje więc jeden bajt, co umożliwiła utworzenie 256 różnych adresów multicast dla danego AS. Jak widać, schemat ten tworzy prostą odpowiedniość między numerem AS a jego adresami multicast, a „przeliczenia" w obie strony, choć same z siebie proste, stają się jeszcze łatwiejsze dzięki dostępnym online[6] rozmaitym kalkulatorom.

Najnowszy schemat przydziału adresów multicast dla danego miejsca sieciowego opiera się na jego prefiksie dla adresów unicast. Określany akronimem UBM (od *Unicast-prefix Based Multicast*) i opisany w dokumencie [RFC6034] schemat ten wzorowany jest na podobnym rozwiązaniu opracowanym wcześniej dla IPv6 (piszemy o nim w następnym punkcie). Począwszy od drugiego bajta (w pierwszym znajduje się liczba 234), na kolejnych bitach umieszczany jest wspomniany prefiks dla adresów unicast, pozostałe bity mogą mieć dowolne wartości. Dla prefiksu unicast o długości n daje to więc możliwość utworzenia 2^{24-n} różnych adresów multicast — oczywiście n nie może przekraczać 24, dla dłuższych prefiksów schemat ten jest nieprzydatny. Na rysunku 2.11 przedstawiono schematycznie tę strukturę.

[5] GLOP to nie akronim, a po prostu nazwa zakresu adresów.

[6] Przykładowo pod adresem *http://gigapop.uoregon.edu/glop/*.

Rysunek 2.11. *Format adresu UBM dla IPv4. Prefiks unicast /24 lub krótszy konkatenowany jest z prefiksem 234/8, na pozostałych spośród 32 bitów umieszczany jest identyfikator grupy — im więc krótszy prefiks unicast, tym więcej możliwych identyfikatorów grup*

I tak np. dla miejsca sieciowego o prefiksie `192.0.2.0/24` utworzyć można tylko jeden adres UBM — `234.192.0.2`. Dla prefiksu `128.32/16` (przydzielonego Uniwersytetowi Kalifornijskiemu w Berkeley, co można sprawdzić za pomocą usługi `WHOIS` — patrz podpunkt 2.6.1.1) utworzyć można 256 adresów multicast o wspólnym prefiksie `234.128.32/24`, i vice versa — podczas analizy kolejnych bitów adresu multicast, począwszy od drugiego bajta, możemy zidentyfikować długość prefiksu unicast właściciela, a następnie sam prefiks. Zgodnie z tabelą 2.2, dwa pierwsze bity `10` identyfikują adres klasy B, czyli prefiks o długości 16; prefiksem unicast jest więc `128.32/16`. Podobnie dla adresu multicast `234.192.0.2` początkowe bity `110` wskazują klasę C, czyli prefiks 24-bitowy (`192.0.2/24`).

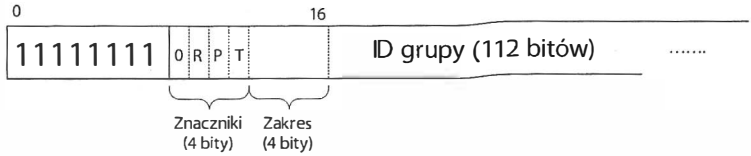
Adresy UBM wykazują wyraźną przewagę nad innymi schematami adresowania w multicastingu IPv4. Po pierwsze, cechują się mniej dotkliwym ograniczeniem długości „materiału wejściowego" niż adresy GLOP: zamiast maksymalnie 16-bitowego numeru AS mamy maksymalnie 24-bitowy prefiks unicast. Po drugie, jako że adresy multicast tworzone są wyłącznie na podstawie prefiksu unicast, ich przydział nie wymaga żadnej koordynacji z „resztą" Internetu. Po trzecie wreszcie, adresy UBM zapewniają lepszą granulację przydziału niż adresy GLOP: system autonomiczny o danym numerze może rozciągać się na wiele miejsc sieciowych, co znacznie komplikuje zależność między adresem multicast a prefiksem unicast jego właściciela.

Bloki administracyjne umożliwiają ograniczanie ruchu multicast do określonej kolekcji routerów lub hostów i w tym sensie stanowią analogie prywatnych adresów unicast. Adresy te powinny zapobiegać wydostawaniu się pakietów do Internetu, bo pakiety te będą w większości blokowane przez routery brzegowe. Duże miejsca sieciowe dzielone są zwykle na grupy, którym przypisywane są odrębne zakresy adresów multicast; podział ten odzwierciedla zazwyczaj strukturę organizacyjną firmy (komórki, wydziały, oddziały) lub jej rozproszenie geograficzne.

2.5.4. Multicasting w IPv6

Protokół IPv6 jest znacząco bardziej niż IPv4 przesiąknięty multicastingiem; na jego potrzeby zarezerwowano adresy rozpoczynające się od prefiksu `ff00::/8`, do dyspozycji pozostaje więc 112 bitów, co daje 2^{112} = 5 192 296 858 534 827 628 530 496 329 220 096 kombinacji. Strukturę tę przedstawiono na rysunku 2.12.

Pierwszy bajt ma więc wartość `0xff`. W pierwszej (bardziej znaczącej) połówce drugiego znajdują się *znaczniki*, druga połówka określa *zakres* obowiązywania adresu. Znacznikami zajmiemy się za chwilę, znaczenia poszczególnych wartości zakresu wyjaśnione są natomiast w tabeli 2.10, bazującej na sekcji 2.7 dokumentu [RFC4291]. Wartości `0`, `3` i `f` są zarezerwowane, wartościom `6`, `7`, `9`, `a`, `b`, `c` i `d` nie przypisano dotąd żadnego znaczenia.



Rysunek 2.12. *Podstawowymi składnikami adresu multicast w IPv6 są: znaczniki (R — adres bazujący na punkcie spotkań (rendezvous point), P — adres bazujący na prefiksie unicast, T — adres tymczasowy) i zakres (patrz tabela 2.10). Gdy oba bity P i R są zerowe, na następnych 112 bitach kodowany jest ID grupy, w przeciwnym razie format adresu jest bardziej złożony*

Tabela 2.10. *Desygnatory zakresu obowiązywania adresu multicast IPv6*

Wartość	Znaczenie
0	Zarezerwowane
1	Adres lokalny dla interfejsu lub hosta
2	Adres lokalny dla łącza lub podsieci
3	Zarezerwowane
4	Zakres administracyjny
5	Adres lokalny dla miejsca sieciowego
6	Niezdefiniowane
7	Niezdefiniowane
8	Adres lokalny dla organizacji
9	Niezdefiniowane
a	Niezdefiniowane
b	Niezdefiniowane
c	Niezdefiniowane
d	Niezdefiniowane
e	Adres globalny
f	Zarezerwowane

Wiele adresów multicast przydzielanych przez IANA do permanentnego użytku celowo rozciąga się na wiele zakresów; każdy zakres definiowany jest przez przesunięcie (*offset*) w stosunku do adresu początkowego w zakresie (dlatego adresy takie nazywamy **relatywnymi do zakresu** — *scope-relative* — lub **zmiennymi w ramach zakresu** — *variable-scope*). Przykładem takiego wielozakresowego zbioru adresów multicast jest zbiór `ff0x::101` przeznaczony dla serwerów NTP (patrz [IP6MA]) — *x* oznacza konkretny zakres, niektóre z jego wartości wyjaśniono w tabeli 2.11

Tabela 2.11. *Przykład wielozakresowego zbioru adresów multicast, przeznaczonego dla protokołu NTP (101)*

Adres	Zastosowanie
ff01::101	Wszystkie serwery funkcjonujące na tym samym komputerze
ff02::101	Wszystkie serwery NTP w danej podsieci lub na tym samym łączu
ff04::101	Wszystkie serwery NTP w pewnym zakresie administracyjnym
ff05::101	Wszystkie serwery NTP w danym miejscu sieciowym
ff08::101	Wszystkie serwery NTP danej organizacji
ff0e::101	Wszystkie serwery NTP w Internecie

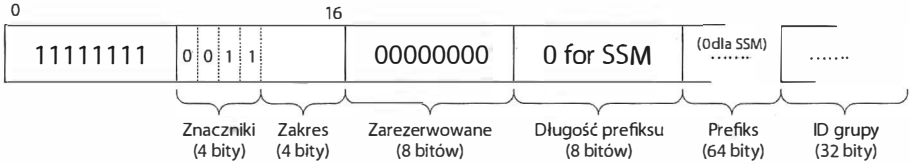
Prosty format adresu przedstawiony na rysunku 2.12 jest obowiązujący w sytuacji, gdy oba znaczniki P i R mają wartość 0. Gdy znacznik P ma wartość 1, interpretowanie adresu odbywa się według jednej z dwóch zasad, opisanych w dokumentach [RFC3306] i [RFC4489], niewymagających uzgodnień na globalnym poziomie Internetu. Pierwsza z nich opiera się na prefiksie unicast IPv6, przydzielanym w sposób kontrolowany (przez dostawcę usług lub inne centrum rejestracyjne), identyfikatory grup ustalane są już w ramach miejsca sieciowego; konieczność globalnej koordynacji w celu uniknięcia kolizji adresów zostaje więc znacznie zredukowana. Druga zasada, dotycząca adresów **ograniczonych do łącza** (*link-scoped*), opiera się na identyfikatorach interfejsów (IID) w hostach podsieci. Aby zrozumieć znaczenie szczegółów obu zasad, konieczne jest uprzednie zapoznanie się ze szczegółami interpretacji dowolnego adresu multicast IPv6. W tabeli 2.12 wyjaśnione jest znaczenie poszczególnych znaczników.

Tabela 2.12. *Znaczniki adresu multicast IPv6*

Znacznik	Znaczenie	Na podstawie dokumentu
R	= 0 — adres regularny = 1 — adres tworzony na podstawie adresu RP	[RFC3956]
P	= 0 — adres regularny = 1 — adres tworzony na podstawie prefiksu unicast	[RFC3306]
T	= 0 — adres trwale przydzielony = 1 — adres tymczasowy, przydzielony dynamicznie	[RFC4291]

Ustawienie znacznika T na 1 oznacza, że adres ma charakter tymczasowy i przydzielony został dynamicznie, nie jest więc standardowym adresem definiowanym w [IP6MA]. Gdy znacznik P ma wartość 1, znacznik T również musi być ustawiony; adres interpretowany jest wówczas zgodnie z formatem przedstawionym na rysunku 2.13.

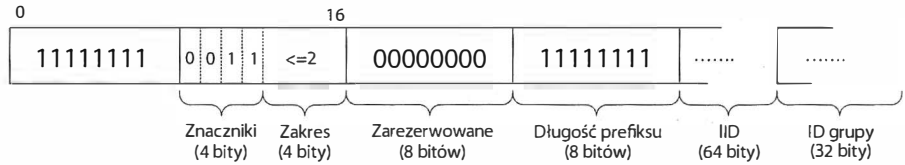
Ponieważ korzysta się tu z przydzielonego wcześniej prefiksu unicast, nie jest konieczne dokonywanie jakichkolwiek uzgodnień na poziomie globalnym w celu uniknięcia kolizji adresów. Po prefiksie (ewentualnie dopełnionym zerami z prawej strony do długości 64 bitów) następuje 32-bitowy identyfikator grupy. I tak np. organizacja otrzymująca centralnie przydział prefiksu unicast 3ffe:ffff:1::/48 dostaje tym samym prefiks dla adresów



Rysunek 2.13. *Adresy multicast IPv6 mogą być tworzone na podstawie przydzielonych adresów unicast (patrz [RFC3306]). Znacznik P ma wówczas wartość 1, a elementem adresu jest prefiks unicast, po którym następuje 32-bitowy identyfikator grupy. Generowanie takiego adresu eliminuje konieczność dokonywania uzgodnień na poziomie globalnym Internetu*

multicast, równy `ff3x:30:3ffe:ffff:1::/96`, gdzie *x* jest dowolnym identyfikatorem zakresu. Jeżeli jednak pola *Długość prefiksu* i *Prefiks* są wyzerowane — co efektywnie oznacza prefiks multicast `ff3x::/32` — adres interpretowany jest jako SSM.

Dla adresów multicast tworzonych na podstawie identyfikatorów interfejsu (IID — patrz [RFC4489]) zamiast prefiksu unicast wykorzystywany jest IID; pole *Długość prefiksu* zawiera wówczas wartość `0xff` (rysunek 2.14).



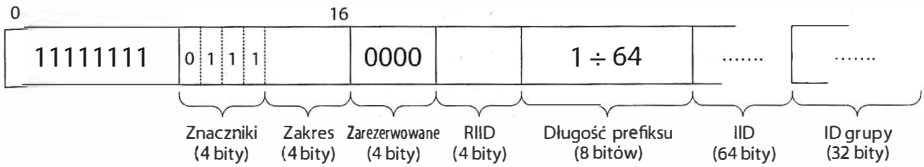
Rysunek 2.14. *Format adresu multicast IPv6 lokalnego dla łącza (lub o mniejszym zakresie) — identyfikator interfejsu (IID) konkatenowany jest z 32-bitowym identyfikatorem grupy. Prefiks takiego adresu równy jest ff3x:0011/32, gdzie x jest identyfikatorem zakresu, mniejszym od 3*

Formowanie adresu multicast IPv6 na podstawie IID ma tę zaletę, że nie wymaga żadnej informacji o charakterze globalnym, znakomicie nadaje się więc do sieci ad hoc pozbawionych routerów — poszczególne hosty mogą generować unikatowe adresy multicast na podstawie własnych identyfikatorów interfejsów i to bez angażowania skomplikowanych protokołów uzgadniania informacji. Oczywiście, jak wcześniej wspomnieliśmy, ten schemat funkcjonuje tylko w zakresie lokalności dla węzła lub lokalności dla łącza (identyfikator zakresu ma wartość nieprzekraczającą 3). Dla przykładowego hosta, posiadającego interfejs o identyfikatorze `02-11-22-33-44-55-66-77`, generowane w ten sposób adresy multicast będą miały postać `ff3x:0011:0211:2233:4455:6677:gggg:gggg`, gdzie *x* ma wartość nie większą od 2, a *gggg:gggg* jest 32-bitowym identyfikatorem grupy.

Znacznik `R` ustawiany jest wówczas, gdy adres multicast generowany jest na podstawie prefiksu unicast (czyli przy ustawionym znaczniku `P`) i używany protokół trasowania wymaga informacji na temat punktu spotkań.

Punkt spotkań (*rendezvous point*, w skrócie RP) to router obsługujący multicasting kierowany do jednej lub kilku grup, wykorzystywany przez protokół PIM-SM (*Protocol Independent Multicast — Sparse Mode* — patrz [RFC4601]), ułatwiający wzajemne odnajdywanie się nadawcy i odbiorcy, którzy należą do tej samej grupy, lecz różnych podsieci. Ponieważ adres RP jest jednym ze składników adresu multicast, lokalizowanie RP wykonuje się przez ekstrakcję odpowiedniego zbioru bitów z tego adresu.

Interpretowanie adresu multicast z ustawionym znacznikiem R odbywa się zgodnie z formatem przedstawionym na rysunku 2.15.



Rysunek 2.15. *Adres multicast IPv6 z wbudowanym adresem RP ([RFC3956]), który można wydobyć za pomocą nieskomplikowanej operacji. Zadaniem RP jest koordynacja nadawców i odbiorców multicastingu w sytuacji, gdy znajdują się oni w różnych podsieciach*

W odróżnieniu od formatu z rysunku 2.13, w formacie przedstawionym na rysunku 2.15 pole *Długość prefiksu* musi mieć wartość różną od zera, ponieważ nie jest wykorzystywany SSM; pojawia się ponadto nowe 4-bitowe pole *RIID*. Ekstrakcja adresu RP odbywa się w tym formacie w sposób zgoła nieskomplikowany: z pola *Prefiks* bierzemy początkowe bity w liczbie wskazanej w polu *Długość prefiksu*, dopełniamy prawostronnie zerami do długości 124 bitów, a do wyniku dołączmy zawartość pola *RIID*. Innymi słowy, prefiks staje się najbardziej znaczącą częścią adresu RP, pole *RIID* staje się jego najmniej znaczącą częścią, a przestrzeń między nimi zostaje wypełniona przez bity zerowe. Przykładowo w adresie multicast `ff75:940:2001:db8:dead:beef:f00d:face` zakres określony jest jako 5 (lokalny dla miejsca sieciowego), w polu *RIID* znajduje się wartość 9, a prefiks jest 64-bitowy (pole *Długość prefiksu* zawiera wartość `0x40`). W wyniku opisanej operacji otrzymujemy ciąg `2001:db8:dead:beef:0000:0000:0000:0009`, czyli adres IPv6 `2001:db8:dead:beef::9`. Więcej podobnych przykładów zainteresowani czytelnicy znajdą w dokumencie [RFC3956].

Podobnie jak w wersji IPv4, tak i w wersji IPv6 niektóre adresy multicast mają specjalne znaczenie lub są zarezerwowane do specjalnego użytku. Ich znaczenie wyjaśnione jest w tabeli 2.13, dodatkowe informacje na ich temat dostępne są w dokumencie [IP6MA].

Tabela 2.13. *Specjalne i zarezerwowane adresy multicast IPv6*

Adres lub prefiks	Zakres	Znaczenie	Na podstawie dokumentu
`ff01::1`	Węzeł	Wszystkie węzły	[RFC4291]
`ff01::2`	Węzeł	Wszystkie routery	[RFC4291]
`ff01::fb`	Węzeł	mDNSv6	[IDChes]
`ff02::1`	Łącze	Wszystkie węzły	[RFC4291]
`ff02::2`	Łącze	Wszystkie routery	[RFC4291]
`ff02::4`	Łącze	Routery DVMRP	[RFC1075]
`ff02::5`	Łącze	OSPFIGP	[RFC2328]
`ff02::6`	Łącze	Wyznaczone routery OSPFIGP	[RFC2328]
`ff02::9`	Łącze	Routery RIPng	[RFC2080]
`ff02::a`	Łącze	Routery EIGRP	[EIGRP]

Tabela 2.13. *Specjalne i zarezerwowane adresy multicast IPv6 — ciąg dalszy*

Adres lub prefiks	Zakres	Znaczenie	Na podstawie dokumentu
ff02::d	Łącze	Routery PIM	[RFC5059]
ff02::16	Łącze	Routery z obsługą MLDv2	[RFC3810]
ff02::6a	Łącze	Wszystkie urządzenia nasłuchujące (*All snoopers*)	[RFC4286]
ff02::6d	Łącze	Routery LL-MANET	[RFC5498]
ff02::fb	Łącze	mDNSv6	[IDChes]
ff02::1:2	Łącze	Wszystkie agenty DHCP	[RFC3315]
ff02::1:3	Łącze	LLMNR	[RFC4795]
ff02::1:ffxx:xxxx	Łącze	Zakres adresów *Solicited-node*	[RFC4291]
ff05::2	Miejsce sieciowe	Wszystkie routery	[RFC4291]
ff05::fb	Miejsce sieciowe	mDNSv6	[IDChes]
ff05::1:3	Miejsce sieciowe	Wszystkie serwery DHCP	[RFC3315]
ff0x::	Zmienny	Zarezerwowane	[RFC4291]
ff0x::fb	Zmienny	mDNSv6	[IDChes]
ff0x::101	Zmienny	NTP	[RFC5905]
ff0x::133	Zmienny	Aggregate Server Access Protocol	[RFC5352]
ff0x::18c	Zmienny	Wszystkie adresy AC[7] (CAPWAP[8])	[RFC5415]
ff3x::/32	(Specjalny)	Blok SSM	[RFC4607]

2.5.5. Adresy anycast

Adresem *anycast* jest adres unicast IPv4 lub IPv6 identyfikujący różne hosty, w zależności od tego, w którym miejscu sieci zostanie użyty. Ten stan rzeczy osiąga się przez skonfigurowanie routerów Internetu w taki sposób, by te same trasy dla adresów unicast ogłaszane były z różnych lokalizacji; wówczas określony adres identyfikować będzie nie pojedynczy host, lecz „najbardziej odpowiedni" czy też „najbliższy" z hostów, które na to ogłaszanie odpowiedzą. Technika ta stosowana jest powszechnie do odnajdywania komputerów świadczących popularne usługi (patrz [RFC4786]), m.in. serwerów DNS (poświęcamy im rozdział 11.), bram 6to4 enkapsulujących pakiety IPv6 w tunelach IPv4 (patrz [RFC3068]) czy też punktów spotkań multicastingu (patrz [RFC4610] i punkt 2.5.4).

[7] AC (*Access Controller* — sterownik dostępowy) — kontroler służący do scentralizowanego zarządzania siecią WLAN — *przyp. tłum.*

[8] CAPWAP (*Control and Provisioning of Wireless Access Points,* dosł. „sterowanie bezprzewodowymi punktami dostępowymi i ich zaopatrywanie") — standardowy protokół zarządzania przez kontroler kolekcją bezprzewodowych punktów dostępowych — *przyp. tłum*

2.6. Przydzielanie adresów IP

Dysponowaniem przestrzenią adresów IP, a szczególnie przydzielaniem jej fragmentów, zajmują się **urzędy rejestracyjne** (*authorities*), zorganizowane w hierarchiczną strukturę. Są to organizacje przydzielające adresy różnym podmiotom — zwykle dostawcom usług internetowych (ISP) lub urzędom rejestracyjnym na niższym poziomie hierarchii. Przydział ten dotyczy głównie adresów unicast, choć niekiedy przydzielane są także adresy multicast i adresy o specjalnym przeznaczeniu. Przydziały mogą mieć charakter permanentny — na czas nieokreślony — lub tymczasowy, np. na czas przeprowadzania pewnych eksperymentów. Na szczycie wspomnianej hierarchii plasuje się IANA (*Internet Assigned Numbers Authority* — patrz [IANA]), sprawująca nadzór nad przydziałem adresów IP i innych numerów związanych z protokołami internetowymi.

2.6.1. Adresy unicast

W kwestii przydziału adresów IPv4 i IPv6 IANA deleguje większość swych kompetencji do kilku **regionalnych urzędów rejestracyjnych** (RIRs — *Regional Internet Registries*), które koordynują swe prace w ramach powołanej do życia w 2003 roku organizacji NRO (*Number Resource Organization* — patrz [NRO]). Funkcjonujące obecnie (jest wiosna 2012) urzędy regionalne wymieniono w tabeli 2.14. Warto przy okazji wspomnieć, że 1 lutego 2011 roku ostatnia niewykorzystana jeszcze porcja adresów IPv4 została rozdysponowana przez IANA pomiędzy urzędy regionalne. Osiągnięto tym samym jeden z kamieni milowych na drodze rozwoju Internetu — wyczerpanie puli adresów IPv4.

Tabela 2.14. *Regionalne urzędy rejestracyjne Internetu zrzeszone w NRO*

Nazwa urzędu	Obszar zarządzany	Kontakt
AfriNIC — African Network Information Center	Afryka	*http://www.afrinic.net*
APNIC — Asia Pacific Network Information Center	Azja, strefa Pacyfiku	*http://www.apnic.net*
ARIN — American Registry for Internet Numbers	Ameryka Północna	*http://www.arin.net*
LACNIC — Regional Latin America and Caribbean IP Address Registry	Ameryka Łacińska i niektóre wyspy karaibskie	*http://lacnic.net/en/index.html*
RIPE NCC — Réseaux IP Européens	Europa, Środkowy Wschód, Azja Środkowa	*http://www.ripe.net*

Wymienione urzędy regionalne sprawują kontrolę nad ogromnymi blokami adresów (patrz [IP4AS] i [IP6AS]), a dokładniej — ich przydziałem dla instytucji rejestracyjnych niższych szczebli, działających w poszczególnych krajach, i dużych dostawców (ISP). Dostawcy z kolei wykorzystują otrzymaną pulę adresów na potrzeby swoich klientów i własne — gdy użytkownik wykupuje u dostawcy dostęp do Internetu, otrzymuje do dyspozycji pewien (zazwyczaj niewielki) zbiór adresów w formie prefiksu (rzadziej — kilku prefiksów). Przydzielane w ten sposób adresy określane są mianem **agregowalnych przez**

dostawcę (*provider-aggregatable*, w skrócie PA), ponieważ dostawca może agregować poszczególne prefiksy (patrz punkt 2.4.2). Adresy takie są **nieprzenośne** (*nonportable*) w tym sensie, że są zależne od konkretnego dostawcy — zmiana dostawcy wiąże się z koniecznością zmiany prefiksów dla wszystkich komputerów klienta podłączonych do Internetu (w szczególności: routerów) — która to (mało przyjemna) operacja nazywana jest popularnie „przenumerowaniem" (*renumbering*).

Alternatywą dla adresów PA są adresy **niezależne od dostawcy** (*provider-independent*, w skrócie PI). Zgodnie z nazwą, klient otrzymuje pewien zbiór adresów (prefiks) w sposób niezależny od dostawcy świadczącego usługi, zmiana dostawcy nie wiąże się więc z przenumerowywaniem hostów. Ponieważ jednak prefiks klienta nie sąsiaduje numerycznie z prefiksami oferowanymi przez dostawcę, nie może być z nimi agregowany; jego obsługa jest dla dostawcy bardziej kłopotliwa (dodatkowa pozycja w tabelach trasowania), co zwykle znajduje odzwierciedlenie w dodatkowych opłatach (niektórzy dostawcy w ogóle nie obsługują adresów PI). Wielu użytkowników skłonnych jest jednak ponosić owe opłaty, bo w ten sposób uwalniają się od kłopotliwego **uzależnienia od dostawcy** (*provider-lock*).

2.6.1.1. Przykłady

Informację na temat przydziału określonego adresu IP można uzyskać za pośrednictwem internetowej usługi WHOIS. Przykładowo wpisując w przeglądarkę URL *http://whois.arin.net/rest/ip/72.1.140.203.txt*, otrzymamy następującą odpowiedź:

```
NetRange:       72.1.140.192 - 72.1.140.223
CIDR:           72.1.140.192/27
OriginAS:
NetName:        SPEK-SEA5-PART-1
NetHandle:      NET-72-1-140-192-1
Parent:         NET-72-1-128-0-1
NetType:        Reassigned
RegDate:        2005-06-29
Updated:        2005-06-29
Ref:            http://whois.arin.net/rest/net/NET-72-1-140-192-1
```

Tak więc adres 72.1.140.203 znajduje się wewnątrz sieci o nazwie SPEK-SEA5-PART-1, której przydzielono adresy rozpoczynające się od prefiksu 72.1.140.192/27. Widzimy ponadto, że zakres adresów sieci SPEK-SEA5-PART-1 jest częścią przestrzeni adresów PA, noszącej nazwę NET-72-1-128-0-1. Informację na temat tej sieci uzyskać możemy za pomocą URL-a *http://whois.arin.net/rest/net/NET-72-1-128-0-1.txt*, otrzymując w odpowiedzi:

```
NetRange:       72.1.128.0 - 72.1.191.255
CIDR:           72.1.128.0/18
OriginAS:
NetName:        SPEAKEASY-6
NetHandle:      NET-72-1-128-0-1
Parent:         NET-72-0-0-0-0
NetType:        Direct Allocation
RegDate:        2004-09-09
Updated:        2012-03-02
Ref:            http://whois.arin.net/rest/net/NET-72-1-128-0-1
```

Z wyświetlonej informacji wynika, że zakres adresów 72.1.128.0/18 (powoływany przez „uchwyt" — *handle* — lub nazwę NET-72-1-128-0-1) wydzielony został z zakresu 72.0.0.0/8 zarządzanego przez ARIN. Więcej informacji na temat formatów danych i różnych metod zapytania WHOIS znaleźć można pod adresem [WRWS].

W podobny sposób uzyskać możemy informację pochodzącą z innego RIR. Jeśli przykładowo interesuje nas adres 193.5.93.80, wpisując go w pole wyszukiwarki dostępnej pod adresem *http://www.ripe.net/whois*, otrzymamy odpowiedź widoczną na rysunku 2.16.

Rysunek 2.16. *Informacja na temat adresu IPv4 193.5.93.80*

Search results

This is the RIPE database search service.
The objects are in RPSL format.
The RIPE Database is subject to Terms and Conditions.
See http://www.ripe.net/db/support/db-terms-conditions.pdf

```
inetnum:        193.5.88.0 - 193.5.95.255
netname:        WIPONET
descr:          World Intellectual Property Organization
descr:          UN Specialized Agency
descr:          Geneva
country:        CH
admin-c:        AM4504-RIPE
tech-c:         AM4504-RIPE
status:         ASSIGNED PI
mnt-by:         CH-UNISOURCE-MNT
mnt-by:         DE-COLT-MNT
source:         RIPE #Filtered

person:         Andras Makadi
address:        World Intellectual Property Organization
address:        34 chemin des Colombettes
address:        CH-1211 Geneva 20
address:        Switzerland
phone:          +41 22 338 9425
phone:          +41 22 338 9111
e-mail:         andras.makadi@wipo.int
e-mail:         callcentre@unicc.org
nic-hdl:        AM4504-RIPE
mnt-by:         AS8659-MNT
source:         RIPE #Filtered

route:          193.5.92.0/22
descr:          World Intellectual Property Organization
origin:         AS8659
mnt-by:         AS8659-MNT
source:         RIPE #Filtered

route:          193.5.93.0/24
descr:          World Intellectual Property Organization
origin:         AS8659
mnt-by:         AS8659-MNT
source:         RIPE #Filtered
```

Widzimy tu, że adres 193.5.93.80 zawarty jest w zakresie identyfikowanym przez prefiks 193.5.88.0/21 przydzielony dla Światowej Organizacji Własności Intelektualnej (WIPO) w Genewie. Zwróćmy uwagę na status tego zakresu — ASSIGNED PI — oznaczający, że mamy do czynienia z adresami niezależnymi od dostawcy. Klauzula „The objects are in RPSL format" oznacza, że treść rekordów bazy zapisana została językiem RPSL (*Routing Policy Specification Language* — język opisu założeń trasowania, patrz [RFC2622] i [RFC4012]) — używanym przez dostawców do opisu przyjętej polityki trasowania. Informacja zawarta w tej bazie pozwala operatorom sieci na skonfigurowanie routerów pod kątem minimalizacji prawdopodobieństwa wprowadzania zakłóceń w funkcjonowaniu Internetu.

2.6.2. Adresy multicast

Adresy multicast w IPv4 i IPv6 można scharakteryzować na podstawie ich zakresu, metody generowania (statycznie, dynamicznie w połączeniu z uzgadnianiem bądź algorytmicznie) oraz odniesienia do nadawcy (ASM albo SSM — patrz punkt 2.5.2). Dokument [RFC5771] jest swoistym przewodnikiem w kwestii przydzielania adresów multicast w IPv4, analogiczne wskazówki dla IPv6 zawarte są w dokumencie [RFC3307], a ogólna ich architektura opisana jest w dokumencie [RFC6308]. Adresy o znaczeniu lokalnym (czyli kontrolowane administracyjnie lub lokalne dla łącza) nie są unikatowe w Internecie (mogą powtarzać się w różnych jego częściach); nadawane są przez administratora lub generowane automatycznie przez hosty końcowe. Adresy o znaczeniu globalnym przydzielane są statycznie, są permanentnie ustalone i z tego względu mogą być „na sztywno" kodowane w aplikacjach.

Przestrzeń statycznych adresów multicast jest ograniczona (szczególnie w IPv4), zaprojektowano je więc z myślą o uniwersalności, czyli możliwości wykorzystywania w kontekście dowolnego miejsca sieciowego. Źródłem dla adresów generowanych algorytmicznie może być numer systemu autonomicznego — AS (adresy GLOP) — lub prefiks unicast danej lokalizacji. Zauważmy, że jeden z bloków adresów globalnych dedykowany jest mechanizmowi SSM; w adresach tego bloku prefiks ma zerową długość, natomiast pole *Zakres* może zawierać dowolną wartość, zgodnie z tabelą 2.10.

Ze względu na znaczne zróżnicowanie formatu adresów multicast oraz relatywnie dużą liczbę protokołów związanych z multicastingiem, zarządzanie tymi adresami (nie mówiąc już o trasowaniu globalnego multicastingu — patrz [RFC5110]) stanowi dla Internetu niebagatelne wyzwanie. Z perspektywy zwykłego użytkownika większość tych problemów jest po prostu niewidoczna — styka się on z multicastingiem sporadycznie, często nieświadomie. Z punktu widzenia programisty aplikacji internetowych wskazane jest uwzględnienie funkcji multicastingu już na etapie ich projektowania; można w tym celu wykorzystać wskazówki zawarte w dokumencie [RFC3170]. Administrator implementujący multicasting w swej sieci nie uniknie prawdopodobnie konieczności kontaktu z dostawcą Internetu dla tej sieci; niektórzy producenci sprzętu i oprogramowania dostarczają pewnych wskazówek w kwestii przydzielania adresów multicast, czego przykładem jest chociażby [CGEMA].

2.7. Przypisywanie adresów unicast do węzłów sieci

Gdy dla miejsca sieciowego zostanie już przydzielony (zwykle przez dostawcę Internetu) zakres adresów unicast, zadaniem administratora jest rozdysponowanie tych adresów między poszczególne interfejsy oraz zdefiniowanie podziału miejsca sieciowego na podsieci. Gdy miejsce sieciowe jest tylko pojedynczym fizycznym segmentem sieci (jak w przypadku większości sieci domowych), są to zabiegi raczej nieskomplikowane; w przypadku bardziej skomplikowanych konfiguracji — szczególnie korzystających z usług wielu dostawców Internetu i (lub) z wielu segmentów, często rozproszonych geograficznie — jest już trochę trudniej. Rozpatrzmy kilka przykładowych sytuacji tego rodzaju, rozpoczynając od konfiguracji najprostszej.

2.7.1. Jeden dostawca, jeden komputer, jeden adres

W najprostszym wariancie dostępu do Internetu klient otrzymuje od dostawcy pojedynczy adres IP (w większości przypadków jest to jeszcze adres IPv4) do wykorzystania przez pojedynczy komputer. W przypadku usług, takich jak DSL, wspomniany adres może mieć charakter tymczasowy, jako adres końcówki łącza „punkt-punkt". Jeżeli np. w konkretnej chwili będzie to adres 63.204.134.177, dowolny program uruchomiony na wspomnianym komputerze, współdziałający z Internetem, używać go będzie jako adresu źródłowego. Nawet w tak elementarnej konfiguracji z komputerem skojarzonych jest kilka innych adresów, m.in. pętla zwrotna (127.0.0.1) i kilka adresów multicast, wśród których absolutne minimum stanowi adres 224.0.0.1 identyfikujący wszystkie hosty w grupie. Jeżeli w komputerze zaimplementowane są protokoły TCP/IP w wersji IPv6, obowiązkowe są: adres ff02::1 identyfikujący wszystkie węzły w sieci, adres pętli zwrotnej ::1, po jednym adresie lokalnym dla łącza dla każdego interfejsu i — oczywiście — jeden lub kilka adresów przydzielonych przez dostawcę.

W systemie Linux można zobaczyć adresy multicast związane aktualnie z hostem, używając poleceń `ifconfig` i `netstat`:

```
Linux% ifconfig ppp0
ppp0 Link encap:Point-to-Point Protocol
inet addr.71.141.244.213
P-t-P:71.141.255.254 Mask:255.255.255.255
UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1492 Metric:1
RX packets:33134 errors:0 dropped:0 overruns:0 frame:0
TX packets:41031 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:3
RX bytes:17748984 (16.9 MiB) TX bytes:9272209 (8.8 MiB)

Linux% netstat -gn
IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      224.0.0.1
ppp0            1      224.0.0.251
ppp0            1      224.0.0.1
lo              1      ff02::1
```

Jak widzimy, urządzeniu ppp0 (jako końcówce połączenia „punkt-punkt") przyporządkowany został adres IPv4 71.141.244.213; nie przypisano żadnych adresów IPv6, jednak w systemie komputera funkcjonują protokoły IPv6, czego dowodem jest obecność adresu pętli zwrotnej (lo) ff02::1. Ponadto aktywna jest grupa wszystkich hostów (adres 224.0.0.1) oraz adres 224.0.0.251 (przydzielony statycznie) usługi mDNS (*multicast* DNS — patrz [IDChes]).

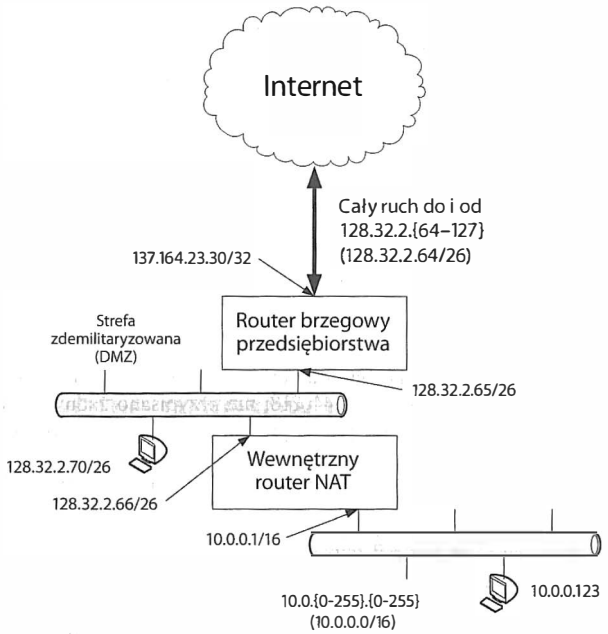
2.7.2. Jeden dostawca, jedna sieć, jeden adres

Konfiguracje z jednym komputerem coraz częściej ustępują miejsca sieciom lokalnym (LAN i WLAN) przyłączonym do Internetu za pośrednictwem routera (lub komputera pełniącego funkcję routera). Router ten, organizujący transfer pakietów między siecią lokalną a Internetem, zwykle realizuje także funkcję **tłumaczenia adresów sieciowych**

(NAT — patrz rozdział 7.), w Windows nazywaną **udostępnianiem połączenia internetowego** (ICS — *Internet Connection Sharing*); dzięki tej funkcji cała sieć zadowala się *jednym* adresem IP otrzymanym od dostawcy (z którego perspektywy niczym nie różni się od konfiguracji z pojedynczym komputerem). Współczesny sprzęt i oprogramowanie automatyzują w dużym stopniu czynności związane z konfigurowaniem sieci NAT, m.in. dzięki użyciu protokołu DHCP (*Dynamic Host Configuration Protocol* — protokół dynamicznego konfigurowania hostów), którego szczegółami zajmiemy się w rozdziale 6.

2.7.3. Jeden dostawca, wiele sieci, wiele adresów

Dla wielu organizacji pojedynczy adres IP — zmieniający się w czasie — okazuje się niewystarczający. W strukturze IT tych organizacji obecne są serwery świadczące rozmaite usługi (np. serwery WWW obsługujące firmowe witryny), których funkcjonowanie wymaga ustalonego permanentnie adresu IP. Infrastruktura ta składa się często z wielu sieci LAN, z których tylko wybrane widoczne są dla Internetu, inne natomiast odizolowane od niego za pomocą firewalli i urządzeń NAT. W tych warunkach uzyskanie odpowiedniej puli adresów IP od dostawcy, podział na podsieci i odizolowanie niektórych podsieci od Internetu to zadania dla administratora (administratorów); przedstawiona na rysunku 2.17 konfiguracja typowa jest dla małych i średnich przedsiębiorstw.



Rysunek 2.17. *Schemat sieci firmowej typowy dla małego lub średniego przedsiębiorstwa. Dostawca przydzielił sieci 64 publiczne adresy IPv4 identyfikowane za pomocą prefiksu 128.32.2.64/26. W strefie zdemilitaryzowanej zlokalizowane są serwery widoczne z Internetu, kontakt pozostałych hostów z Internetem kontrolowany jest przez router NAT*

W konfiguracji tej wykorzystywane są 64 publiczne adresy IPv4, odpowiadające prefiksowi 128.32.2.64/26, co umożliwia uruchomienie 64–2 = 62 węzłów widocznych z Internetu. „Strefa zdemilitaryzowana" (DMZ — *DeMilitarized Zone*), którą zajmiemy się dokładniej w rozdziale 7., to wydzielony fragment sieci, niechroniony przez firmowy firewall, obejmujący serwery dostępne dla użytkowników za pośrednictwem Internetu, realizujące m.in. dostęp do firmowych witryn WWW, uwierzytelnianie itp. Serwery te wykorzystują zwykle niewielką część wspomnianej puli 62 publicznych adresów; pozostałe adresy oddane są do dyspozycji routera NAT jako tzw. „pula NAT" (patrz rozdział 7.); mówiąc ogólnie, router ten zajmuje się konwertowaniem datagramów wychodzących z wewnętrznych hostów w taki sposób, że wewnętrzny adres źródłowy IP hosta zastępowany jest jednym z adresów ze wspomnianej puli NAT; przychodzące z Internetu datagramy przeznaczone dla wewnętrznych hostów podlegają odwrotnemu przekształceniu.

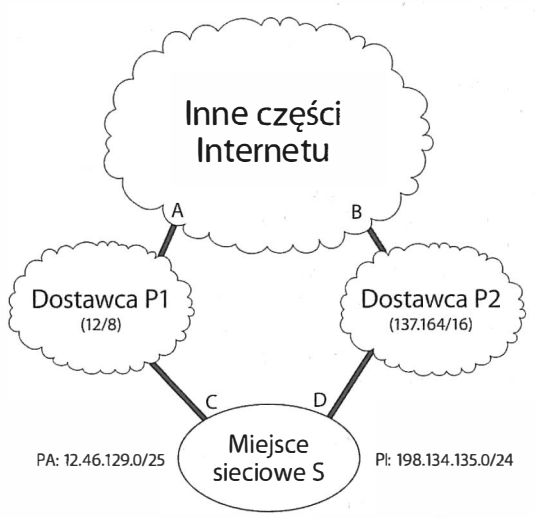
Opisana konfiguracja ma dwie bardzo korzystne cechy. Po pierwsze, w sytuacji skompromitowania któregoś z serwerów w strefie DMZ hosty wewnętrzne chronione są przez firewall stanowiący nieodłączną część routera NAT; dodatkowy czynnik ochronny wynika z faktu, że adresy IP serwerów działających w strefie DMZ są inne niż te przydzielone do puli NAT. Po drugie, obsługa „wewnętrznego" segmentu sieci spoczywa całkowicie w gestii routera NAT; mówiąc szczegółowo, liczba hostów w tym segmencie nie jest w żaden sposób ograniczona liczbą dostępnych publicznych adresów IP w puli NAT — w skrajnym przypadku wystarczający okazuje się nawet jeden adres IP. Oczywiście, schemat z rysunku 2.17 przedstawia tylko pewną ideę; poszczególne parametry jej implementacji — m.in. liczba adresów IP uzyskiwanych od dostawcy — są wynikiem szczegółowych decyzji administracyjnych podyktowanych względami kosztów, wydajności, niezawodności i bezpieczeństwa.

2.7.4. Wielu dostawców, wiele sieci, wiele adresów (multihoming)

Jeżeli dostęp do Internetu jest czynnikiem krytycznych z perspektywy funkcjonowania przedsiębiorstwa, względy niezawodności, dostępności czy wydajności przemawiać mogą za zorganizowaniem tego dostępu za pośrednictwem kilku dostawców — co określane jest popularnie mianem *multihomingu*. Ze względu na mechanizm CIDR, dostawcy Internetu wykorzystują powszechnie agregowanie prefiksów, więc adresy udostępniane klientom są adresami agregowanymi (PA); gdy klient decyduje się na drugiego dostawcę (bez rezygnacji z usług pierwszego), pojawia się problem optymalnego podziału adresów IP (pochodzących teraz z dwóch źródeł) między poszczególne segmenty sieci. Opracowano kilka użytecznych wskazówek związanych z taką sytuacją (i podobną, jaką jest zmiana dostawcy); w dokumencie [RFC4116] znajdują się takie wskazówki dotyczące adresów IPv4 kategorii PI oraz PA.

Przeanalizujmy konfigurację przedstawioną na rysunku 2.18. Miejsce sieciowe S (poniekąd fikcyjne) korzysta z usług dwóch dostawców Internetu — P1 i P2. Gdy używane są adresy PA z bloku dostawcy P1 (12.46.129.0/25), ich prefiksy są oznajmiane dostawcom P1 i P2 w punktach (odpowiednio) C i D. U dostawcy P1 prefiks 12.46.129.0/25 agregowany jest do postaci 12/8 i w takiej formie wędruje do Internetu przez punkt A. Ponieważ podobna operacja nie jest wykonalna u dostawcy P2 — prefiks 12.46.129.0/25 nie sąsiaduje numerycznie z prefiksem 137.164/16 — w punkcie B pojawia się prefiks 12.46.129.0/25 w oryginalnej postaci. Ponieważ trasowanie w Internecie sterowane jest

algorytmem **najdłuższego pasującego prefiksu** (*longest matching prefix* — patrz rozdział 5.), prefiksy dłuższe („bardziej specyficzne") preferowane są kosztem prefiksów krótszych („bardziej ogólnych"); w konsekwencji ruch kierowany zgodnie z prefiksem 12.46.129.0/25 trasowany będzie raczej do dostawcy P2 niż P1 — hosty zlokalizowane w obszarze oznaczonym jako *Inne części Internetu*, mając dwie możliwości zlokalizowania adresu 12.46.129.1 — poprzez prefiks 12.0.0.0/8 i poprzez prefiks 12.46.129.0/25 — wybiorą tę drugą, ze względu na bardziej specyficzny prefiks. Prowadzi to do sytuacji, w której dostawca P2 zostaje obciążony znakomitą większością ruchu kierowanego do miejsca sieciowego S i jednocześnie pozbawiony jest możliwości agregowania prefiksu tegoż miejsca.



Rysunek 2.18. *Użycie adresów IPv4 agregowanych (PA) i niezależnych od dostawcy (PI) w hipotetycznej sieci multihomed przedsiębiorstwa. Administratorzy miejsc sieciowych preferują adresy PI, ponieważ uniezależnia ich to od konkretnych dostawców; dostawcy preferują adresy PA, ponieważ umożliwiają one agregację prefiksów i w konsekwencji redukcję rozmiaru tablic trasowania*

Gdy administrator miejsca sieciowego S zdecyduje się na używanie adresów niezależnych od dostawcy (PI) — 198.134.135.0/24 na rysunku 2.18 — sytuacja stanie się bardziej symetryczna, ponieważ żaden z dostawców P1 i P2 nie będzie miał możliwości agregowania prefiksu tego miejsca. W punktach A i B obaj dostawcy „ogłaszać" będą ten sam prefiks 198.134.135.0/24, a wybór między P1 i P2 dokonywany będzie na podstawie ich względnego położenia („odległości") w stosunku do hosta wysyłającego datagram — co wynika z naturalnej w Internecie tendencji „trasowania po najkrótszej ścieżce". Administrator miejsca sieciowego, decydując się na wybór adresów PI, ułatwia sobie ewentualną zmianę dostawców w przyszłości; z drugiej jednak strony, niemożność agregowania tych adresów przez dostawców jest zjawiskiem problematycznym z perspektywy przyszłej skalowalności Internetu.

Mutihoming w IPv6 był przez lata przedmiotem intensywnych studiów w ramach IETF; rezultatem tych studiów jest architektura *Multi6* (patrz [RFC4177]) i protokół *Shim6*. Ogólnie rzecz biorąc, architektura Multi6 przejmuje zasadniczo metody multihomingu

stosowane w IPv4, rozszerzone o mechanizmy mobilności („mobilne IP6" — patrz [RFC6275]), wprowadza jednocześnie nową koncepcję rozdzielenia dwóch funkcji spełnianych obecnie przez adres IP: *identyfikację* węzła i jego *lokalizację* w kontekście trasowania. Koncepcja ta ma z założenia zapewnić nieprzerwane działanie implementacji protokołów mimo zmian, jakie zachodzą w adresie IP węzła, co jest zjawiskiem typowym dla komunikacji mobilnej. Protokoły obsługujące taką separację opatrywane są skrótowym przydomkiem *id/loc split*.

Protokół Shim6 wprowadza nową warstwę pośredniczącą („przekładkę" — *shim*) odpowiedzialną za odseparowanie „identyfikatora protokołu warstwy wyższej" wykorzystywanego przez protokoły transportowe od adresu IP. Multihoming realizowany jest poprzez wybór adresów IP do pełnienia roli lokalizatorów, w oparciu o dynamicznie zmieniające się uwarunkowania w sieci, bez potrzeby przydzielania adresów PI. Komunikujące się hosty uzgadniają między sobą użycie konkretnych lokalizatorów i przełączanie się między lokalizatorami.

Koncepcja oddzielenia identyfikatora węzła od jego lokalizatora stała się również źródłem innego pomysłu, jakim jest wykorzystywanie **kryptograficznych identyfikatorów** hostów, którymi to identyfikatorami są po prostu klucze publiczne tych hostów lub pary kluczy „prywatny-publiczny"; praktyczną realizacją tego pomysłu jest protokół HIP (*Host Identify Protocol* — protokół identyfikacji hostów) opisany w dokumencie [RFC4423]. Komunikujące się hosty nie są już anonimowe; ponieważ ich identyfikowanie opiera się na mechanizmach kryptograficznych, może być połączone z wzajemnym uwierzytelnianiem, co ma istotne znaczenie dla bezpieczeństwa komunikacji (bezpieczeństwu Internetu poświęcamy rozdział 18.).

2.8. Ataki z wykorzystaniem adresów IP

Gdy weźmie się pod uwagę, że adresy IP to po prostu liczby, repertuar możliwych ataków sieciowych z ich wyłącznym udziałem wydaje się raczej niewielki. Generalnie najczęściej stosowaną techniką ataku (a raczej — utrudnień w przeciwdziałaniu atakom) jest fałszowanie pochodzenia (czyli adresu źródłowego) w pakietach (powszechnie określane jako *spoofing* — patrz rozdział 5.). Wysiłki zmierzające do ustalenia źródła podejrzanej aktywności (np. naruszenia praw autorskich w sieciach p2p lub nielegalnej dystrybucji treści) na podstawie adresów IP mogą więc prowadzić do chybionych rezultatów i to nie tylko ze wspomnianej przyczyny, ale i z kilku innych powodów. W wielu przypadkach adresy IP przydzielane są dynamicznie i ten sam adres IP może w różnym czasie być kojarzony z wieloma różnymi węzłami, drobny więc nawet błąd w synchronizacji zegarów może skierować prowadzone śledztwo na manowce. Ponadto mechanizmy kontroli dostępu nie zawsze są prawidłowo i bezpiecznie zaimplementowane, zatem wspomniane śledztwo prowadzić może do komputera, do którego intruz włamał się przez niezabezpieczoną sieć Wi-Fi (a którego właściciel nic nie podejrzewa i żadnych niecnych działań absolutnie nie wykonywał). Co więcej, skompromitowane hosty mogą służyć do formowania sieci botnetów (ba, w Internecie działa nawet czarny rynek wynajmu takich hostów) wykorzystywanych do proliferacji zabronionych treści, wykradania poufnych informacji, nielegalnego transferu plików i popełniania wielu innych przestępstw (patrz [RFC4948]).

2.9. Podsumowanie

Adresy IP służą do identyfikowania i lokalizowania interfejsów sieciowych w urządzeniach przyłączonych do Internetu (adresy unicast) lub grup tych interfejsów (adresy multicast, broadcast i anycast). Do każdego interfejsu przypisany jest przynajmniej jeden 32-bitowy adres IPv4 (gdy wykorzystywany jest protokół IPv4) i zwykle kilka 128-bitowych adresów IPv6 (gdy urządzenie implementuje protokół IPv6). Adresy unicast przydzielane są blokami, zorganizowanymi w hierarchiczną strukturę zarządzanych administracyjnie encji. Reprezentujący daną encję prefiks wyznacza w istocie pewien zakres (przedział) adresów IP, pozostający w dyspozycji dostawcy Internetu (ISP) lub stanowiący własność klienta (niezależną od ISP). Sąsiadujące numerycznie prefiksy mogą być ze sobą agregowane, co przyczynia się do zmniejszenia rozmiarów tablic trasowania i poprawia skalowalność Internetu.

Prefiksy adresów IP pojawiły się jako sukcesor stosowanej wcześniej „klasowej" struktury adresów, zarzuconej na rzecz bardziej efektywnego trasowania „bezklasowego" (CIDR), pozwalającego lepiej dopasowywać zakresy przydzielanych adresów do potrzeb, rozumianych jako maksymalna liczba hostów w przyłączanej do Internetu sieci prywatnej: zamiast limitów 16 777 214, 65 534 i 254 hostów, odpowiadających klasom A, B i C, dzięki zróżnicowanym prefiksom istnieje możliwość bardziej elastycznego limitowania.

Adres anycast to adres identyfikujący jeden z wielu możliwych hostów — wybór konkretnego hosta zależny jest od tego, w którym miejscu sieci następuje odwołanie do tegoż adresu. Mechanizm ten wykorzystywany jest powszechnie do odnajdywania najbliższych serwerów świadczących konkretną usługę.

Adresy unicast IPv6 są pod wieloma względami różne od ich odpowiedników z IPv4. Najważniejszą nowością adresów IPv6 jest koncepcja lokalności, czyli zakresu obowiązywania danego adresu: adres IPv6 może być lokalny dla węzła (poza którym nie ma znaczenia), lokalny dla łącza, lokalny dla miejsca sieciowego lub globalny (czyli jednakowo interpretowany na przestrzeni całego Internetu). Adresy lokalne dla łączy tworzone są zwykle jako kombinacja standardowego prefiksu oraz identyfikatora interfejsu (IID) dostarczanego przez protokoły warstw niższych (adresu sprzętowego, adresu MAC itp.) albo wartości losowej. Ułatwia to konfigurowanie adresów IPv6 w sieciach prywatnych.

Zarówno w wersji IPv4, jak i IPv6 istnieją adresy odnoszące się nie do pojedynczych węzłów, lecz do ich grup. W IPv4 są to adresy broadcast i multicast, w wersji IPv6 jedynie adresy multicast — adres broadcast jest szczególnym przypadkiem adresu grupowego, reprezentującego *wszystkie* hosty podsieci. Adresy multicast funkcjonują w sposób przypominający nieco kanały telewizji: nadawca wysyła datagram do wszystkich hostów danej grupy, nie znając żadnego z nich ani nawet ich liczby w grupie. W ciągu ostatnich kilkunastu lat dokonał się znaczący rozwój wsparcia dla multicastingu w Internecie, zarówno pod względem trasowania, przydzielania i koordynowania adresów, jak i dołączania hostów do grup (i opuszczania tychże grup) — co znalazło wyraz w opracowaniu wielu protokołów wymienionych kategorii. W IPv6 struktura adresu multicast jest znacząco rozbudowana w porównaniu z IPv4: adresy multicast IPv6 mogą być tworzone na bazie prefiksów unicast, identyfikatorów interfejsów (IID) oraz informacji trasowania wbudowywanych w grupy (w postaci adresu RP).

Opracowanie i wdrożenie CIDR to bodaj najbardziej spektakularna zmiana w rdzennym systemie trasowania Internetu od momentu jego powstania. Koncepcja CIDR narodziła się jako odpowiedź na potrzebę bardziej elastycznego systemu dysponowania globalnymi adresami oraz jako środek promowania skalowalności poprzez agregację.

Przewidując trafnie, iż pula adresów IPv4 ulegnie wyczerpaniu w bliskiej perspektywie, rozpoczęto w latach 90. ubiegłego wieku intensywne prace na nową wersją protokołu IP — IPv6 — w przekonaniu, że rychło staną się nieodzowne adresy o większej długości. Nieoczekiwane rozpowszechnienie technologii NAT (patrz rozdział 7.) spowolniło znacząco adaptację IPv6 w Internecie, bo złagodzone zostało tempo konsumowania nowych adresów: pojedynczy globalny adres IP stał się wystarczający dla całego miejsca sieciowego, co stanowiło atrakcyjną alternatywę dla osobnych adresów poszczególnych hostów.

Nie zmienia to jednak faktu, że pula adresów IPv4 rzeczywiście się kończy: w lutym 2011 roku IANA rozdysponowała pięć ostatnich prefiksów /8, po jednym dla każdego z urzędów regionalnych (RIR). 15 kwietnia 2011 roku wyczerpały się wszystkie prefiksy będące w dyspozycji APNIC; te, którymi dysponują jeszcze pozostałe urzędy regionalne, wyczerpią się najdalej w ciągu kilku najbliższych lat. Zainteresowani czytelnicy znaleźć mogą pod adresem [IP4R] aktualny stan wykorzystania puli adresów IPv4.

2.10. Bibliografia

[CGEMA] Cisco Systems „Guidelines for Enterprise IP Multicast Address Allocation", 2004, *http://www.cisco.com/warp/public/cc/techno/tity/prodlit/ipmlt_wp.pdf*

[EIGRP] B. Albrightson, J. J. Garcia-Luna-Aceves, J. Boyle, „EIGRP — A Fast Routing Protocol Based on Distance Vectors", *Proc. Infocom*, 2004.

[EUI64] Institute for Electrical and Electronics Engineers, „Guidelines for 64-Bit Global Identifier (EUI-64) Registration Authority" marzec 1997, *http://standards.ieee.org/regauth/oui/tutorials/EUI64.html*

[H96] M. Handley, „The SDR Session Directory: An Mbone Conference Scheduling and Booking System", Department of Computer Science, University College London, kwiecień 1996, *http://cobweb.ecn.purdue.edu/~ace/mbone/mbone/sdr/intro.html*

[IANA] Internet Assigned Numbers Authority, *http://www.iana.org*

[IDChes] S. Cheshire, M. Krochmal, „Multicast DNS", Internet draftcheshire-dnsext-multicastdns, work in progress, październik 2010.

[IDv4v6mc] S. Venaas, X. Li, C. Bao, „Framework for IPv4/IPv6 Multicast Translation", Internet draft-venaas-behave-v4v6mc-framework, work in progress, grudzień 2010.

[IEEERA] IEEE Registration Authority, *http://standards.ieee.org/regauth*

[IMR02] B. Edwards, L. Giuliano, B. Wright, *Interdomain Multicast Routing: Practical Juniper Networks and Cisco Systems Solutions* (Addison-Wesley, 2002).

[IP4AS] *http://www.iana.org/assignments/ipv4-address-space*

[IP4MA] *http://www.iana.org/assignments/multicast-addresses*

[IP4R] IPv4 Address Report, *http://www.potaroo.net/tools/ipv4*

[IP6AS] *http://www.iana.org/assignments/ipv6-address-space*

[IP6MA] *http://www.iana.org/assignments/ipv6-multicast-addresses*

[KK77] L. Kleinrock, F. Kamoun, *Hierarchical Routing for Large Networks, Performance Evaluation and Optimization*, „Computer Networks", 1(3), 1977.

[NRO] Number Resource Organization, *http://www.nro.net*

[RFC0919] J.C. Mogul, *Broadcasting Internet Datagrams*, Internet RFC 0919/BCP 0005, październik 1984.

[RFC0922] J.C. Mogul, *Broadcasting Internet Datagrams in the Presence of Subnets*, Internet RFC 0922/STD 0005, październik 1984.

[RFC0950] J.C. Mogul, J. Postel, *Internet Standard Subnetting Procedure*, Internet RFC 0950/STD 0005, sierpień 1985.

[RFC1075] D. Waitzman, C. Partridge, S.E. Deering, *Distance Vector Multicast Routing Protocol*, Internet RFC 1075 (experimental), listopad 1988.

[RFC1112] S.E. Deering, *Host Extensions for IP Multicasting*, Internet RFC 1112/STD 0005, sierpień 1989.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1812] F. Baker (red.), *Requirements for IP Version 4 Routers*, Internet RFC 1812/STD 0004, czerwiec 1995.

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, E. Lear, *Address Allocation for Private Internets*, Internet RFC 1918/BCP 0005, luty 1996.

[RFC2080] G. Malkin, R. Minnear, *RIPng for IPv6*, Internet RFC 2080, styczeń 1997.

[RFC2328] J. Moy, *OSPF Version 2*, Internet RFC 2328/STD 0054, kwiecień 1988.

[RFC2365] D. Meyer, *Administratively Scoped IP Multicast*, Internet RFC 2365/BCP 0023, lipiec 1998.

[RFC2544] S. Bradner, J. McQuaid, *Benchmarking Methodology for Network Interconnect Devices*, Internet RFC 2544 (informational), marzec 1999.

[RFC2622] C. Alaettinoglu, C. Villamizar, E. Gerich, D. Kessens, D. Meyer, T. Bates, D. Karrenberg, M. Terpstra, *Routing Policy Specification Language (RPSL)*, Internet RFC 2622, czerwiec 1999.

[RFC2644] D. Senie, *Changing the Default for Directed Broadcasts in Routers*, Internet RFC 2644/BCP 0034, sierpień 1999.

[RFC2974] M. Handley, C. Perkins, E. Whelan, *Session Announcement Protocol*, Internet RFC 2974 (experimental), październik 2000.

[RFC3056] B. Carpenter, K. Moore, *Connection of IPv6 Domains via IPv4 Clouds*, Internet RFC 3056, luty 2001.

[RFC3068] C. Huitema, *An Anycast Prefix for 6to4 Relay Routers*, Internet RFC 3068, czerwiec 2001.

[RFC3170] B. Quinn, K. Almeroth, *IP Multicast Applications: Challenges and Solutions*, Internet RFC 3170 (informational), wrzesień. 2001.

[RFC3180] D. Meyer, P. Lothberg, *GLOP Addressing in 233/8*, Internet RFC 3180/BCP 0053, wrzesień 2001.

[RFC3306] B. Haberman, D. Thaler, *Unicast-Prefix-Based IPv6 Multicast Addresses*, Internet RFC 3306, sierpień 2002.

[RFC3307] B. Haberman, *Allocation Guidelines for IPv6 Multicast Addresses*, Internet RFC 3307, sierpień 2002.

[RFC3315] R. Droms (red.), J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, Internet RFC 3315, lipiec 2003.

[RFC3569] S. Bhattacharyya (red.), *An Overview of Source-Specific Multicast (SSM)*, Internet RFC 3569 (informational), lipiec 2003.

[RFC3701] R. Fink, R. Hinden, *6bone (IPv6 Testing Address Allocation) Phaseout*, Internet RFC 3701 (informational), marzec 2004.

[RFC3810] R. Vida, L. Costa (red.), *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*, Internet RFC 3810, czerwiec 2004.

[RFC3849] G. Huston, A. Lord, P. Smith, *IPv6 Address Prefix Reserved for Documentation*, Internet RFC 3849 (informational), lipiec 2004.

[RFC3879] C. Huitema, B. Carpenter, *Deprecating Site Local Addresses*, Internet RFC 3879, wrzesień 2004.

[RFC3927] S. Cheshire, B. Aboba, E. Guttman, *Dynamic Configuration of IPv4 Link-Local Addresses*, Internet RFC 3927, maj 2005.

[RFC3956] P. Savola, B. Haberman, *Embedding the Rendezvous Point (RP) Address in an IPv6 Multicast Address*, Internet RFC 3956, listopad 2004.

[RFC4012] L. Blunk, J. Damas, F. Parent, A. Robachevsky, *Routing Policy Specification Language Next Generation (RPSLng)*, Internet RFC 4012, marzec 2005.

[RFC4116] J. Abley, K. Lindqvist, E. Davies, B. Black, V. Gill, *IPv4 Multihoming Practices and Limitations*, Internet RFC 4116 (informational), lipiec 2005.

[RFC4177] G. Huston, *Architectural Approaches to Multi-homing for IPv6*, Internet RFC 4177 (informational), wrzesień 2005.

[RFC4193] R. Hinden, B. Haberman, *Unique Local IPv6 Unicast Addresses*, październik 2005.

[RFC4286] B. Haberman, J. Martin, *Multicast Router Discovery*, Internet RFC 4286, grudzień 2005.

[RFC4291] R. Hinden, S. Deering, *IP Version 6 Addressing Architecture*, Internet RFC 4291, luty 2006.

[RFC4380] C. Huitema, *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)*, Internet RFC 4380, luty 2006.

[RFC4423] R. Moskowitz, P. Nikander, *Host Identity Protocol (HIP) Architecture*, Internet RFC 4423 (informational), maj 2006.

[RFC4489] J.-S. Park, M.-K. Shin, H.-J. Kim, *A Method for Generating Link-Scoped IPv6 Multicast Addresses*, Internet RFC 4489, kwiecień 2006.

[RFC4566] M. Handley, V. Jacobson, C. Perkins, *SDP: Session Description Protocol*, Internet RFC 4566, lipiec 2006.

[RFC4601] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, *Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification (Revised)*, Internet RFC 4601, sierpień 2006.

[RFC4607] H. Holbrook, B. Cain, *Source-Specific Multicast for IP*, Internet RFC 4607, sierpień 2006.

[RFC4608] D. Meyer, R. Rockell, G. Shepherd, *Source-Specific Protocol Independent Multicast in 232/8*, Internet RFC 4608/BCP 0120, sierpień 2006.

[RFC4610] D. Farinacci, Y. Cai, *Anycast-RP Using Protocol Independent Multicast (PIM)*, Internet RFC 4610, sierpień 2006.

[RFC4632] V. Fuller, T. Li, *Classless Inter-domain Routing (CIDR): The Internet Address Assignment, Aggregation Plan*, Internet RFC 4632/BCP 0122, sierpień 2006.

[RFC4786] J. Abley, K. Lindqvist, *Operation of Anycast Services*, Internet RFC 4786/BCP 0126, grudzień 2006.

[RFC4795] B. Aboba, D. Thaler, L. Esibov, *Link-Local Multicast Name Resolution (LLMNR)*, Internet RFC 4795 (informational), styczeń 2007.

[RFC4843] P. Nikander, J. Laganier, F. Dupont, *An IPv6 Prefix for Overlay Routable Cryptographic Hash Identifiers (ORCHID)*, Internet RFC 4843 (experimental), kwiecień 2007.

[RFC4893] Q. Vohra, E. Chen, *BGP Support for Four-Octet AS Number Space*, Internet RFC 4893, maj 2007.

[RFC4948] L. Andersson, E. Davies, L. Zhang, eds., *Report from the IAB Workshop on Unwanted Traffic March 9 – 10, 2006*, Internet RFC 4948 (informational), sierpień 2007.

[RFC5059] N. Bhaskar, A. Gall, J. Lingard, S. Venaas, *Bootstrap Router (BSR) Mechanism for Protocol Independent Multicast (PIM)*, Internet RFC 5059, styczeń 2008.

[RFC5110] P. Savola, *Overview of the Internet Multicast Routing Architecture*, Internet RFC 5110 (informational), styczeń 2008.

[RFC5156] M. Blanchet, *Special-Use IPv6 Addresses*, Internet RFC 5156 (informational), kwiecień 2008.

[RFC5214] F. Templin, T. Gleeson, D. Thaler, *Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)*, Internet RFC 5214 (informational), marzec 2008.

[RFC5352] R. Stewart, Q. Xie, M. Stillman, M. Tuexen, *Aggregate Server Access Protocol (ASAP)*, Internet RFC 5352 (experimental), wrzesień 2008.

[RFC5415] P. Calhoun, M. Montemurro, D. Stanley (red.), *Control, Provisioning of Wireless Access Points (CAPWAP) Protocol Specification*, Internet RFC 5415, marzec 2009.

[RFC5498] I. Chakeres, *IANA Allocations for Mobile Ad Hoc Network (MANET) Protocols*, Internet RFC 5498, marzec 2009.

[RFC5533] E. Nordmark, M. Bagnulo, *Shim6: Level 3 Multihoming Shim Protocol for IPv6*, Internet RFC 5533, czerwiec 2009.

[RFC5735] M. Cotton, L. Vegoda, *Special Use IPv4 Addresses*, Internet RFC 5735/BCP 0153, styczeń 2010.

[RFC5736] G. Huston, M. Cotton, L. Vegoda, *IANA IPv4 Special Purpose Address Registry*, Internet RFC 5736 (informational), styczeń 2010.

[RFC5737] J. Arkko, M. Cotton, L. Vegoda, *IPv4 Address Blocks Reserved for Documentation*, Internet RFC 5737 (informational), styczeń 2010.

[RFC5771] M. Cotton, L. Vegoda, D. Meyer, *IANA Guidelines for IPv4 Multicast Address Assignments*, Internet RFC 5771/BCP 0051, marzec 2010.

[RFC5952] S. Kawamura, M. Kawashima, *A Recommendation for IPv6 Address Text Representation*, Internet RFC 5952, sierpień 2010.

[RFC5905] D. Mills, J. Martin (red.), J. Burbank, W. Kasch, *Network Time Protocol Version 4: Protocol, Algorithms Specification*, Internet RFC 5905, czerwiec 2010.

[RFC6034] D. Thaler, *Unicast-Prefix-Based IPv4 Multicast Addresses*, Internet RFC 6034, październik 2010.

[RFC6052] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, X. Li, *IPv6 Addressing of IPv4/IPv6 Translators*, Internet RFC 6052, październik 2010.

[RFC6217] J. Arkko, M. Townsley, *IPv4 Run-Out and IPv4-IPv6 Co-Existence Scenarios*, Internet RFC 6127 (experimental), maj 2011.

[RFC6144] F. Baker, X. Li, C. Bao, K. Yin, *Framework for IPv4/IPv6 Translation*, Internet RFC 6144 (informational), kwiecień 2011.

[RFC6164] M. Kohno, B. Nitzan, R. Bush, Y. Matsuzaki, L. Colitti, T. Narten, *Using 127-Bit IPv6 Prefixes on Inter-Router Links*, Internet RFC 6164, kwiecień 2011.

[RFC6275] C. Perkins (red.), D. Johnson, J. Arkko, *Mobility Support in IPv6*, Internet RFC 3775, lipiec 2011.

[RFC6308] P. Savola, *Overview of the Internet Multicast Addressing Architecture*, Internet RFC 6308 (informational), czerwiec 2011.

[WRWS] *http://www.arin.net/resources/whoisrws*

Rozdział 3.
Warstwa łącza danych

3.1. Wprowadzenie

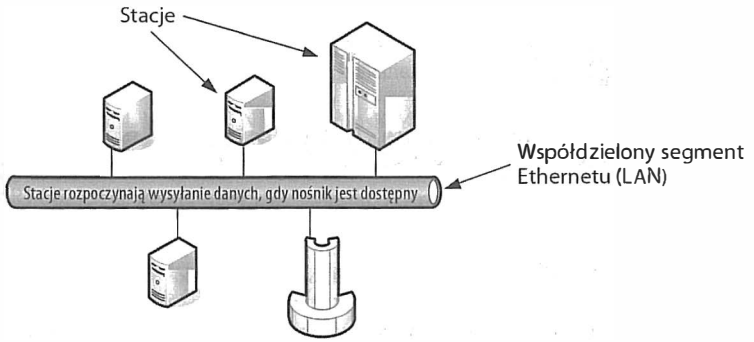
Jak wyjaśnialiśmy w rozdziale 1., w zestawie protokołów TCP/IP zadaniem warstwy łącza danych jest przesyłanie datagramów na rzecz modułu IP i kilku innych protokołów wspomagających, m.in. ARP (patrz rozdział 4.). Protokoły TCP/IP współpracują z różnymi technologiami łącza danych: popularnym w sieciach LAN Ethernetem, kablami TV i modemami DSL w sieciach metropolitarnych (MAN), liniami telefonicznymi, łącznością bezprzewodową Wi-Fi, a ostatnio technologiami komórkowymi — HSPA, EV-DO, LTE i WiMax. Wyczerpujące omówienie wszystkich tych technologii wymagałoby zapewne osobnego księgozbioru, w tym rozdziale zajmiemy się więc dokładniej Ethernetem i łącznością Wi-Fi w kontekście TCP/IP, omówimy funkcjonowanie protokołu „punkt-punkt" (PPP — *Point-to-Point Protocol*), zajmiemy się także techniką **tunelowania**, czyli transferu pakietów warstwy łącza danych w otoczkach utworzonych z pakietów warstwy wyższej lub równorzędnej.

Dla każdej niemal technologii warstwy łącza danych zdefiniowano szczegółowo protokoły i formaty jednostek protokołów (PDU), stosownie do specyfiki funkcjonowania określonego sprzętu sieciowego. W kontekście warstwy łącza danych jednostki protokołu nazywane są **ramkami** (*frames*), w odróżnieniu od jednostek warstw wyższych — pakietów (warstwa sieciowa) i segmentów (warstwa transportowa). Format ramki zazwyczaj dopuszcza jej zróżnicowaną długość, od kilku bajtów do kilku kilobajtów; maksymalną dopuszczalną wielkość ramki oznacza się skrótem MTU, od *Maximum Transmission Unit* — „maksymalna jednostka transmisji". W przypadku niektórych technologii MTU jest pochodną fizycznych właściwości nośników, dla innych — jak modemy i łącza szeregowe — jest ona jawnie definiowana przez użytkowników.

3.2. Ethernet i standardy IEEE 802 LAN/MAN

Termin **Ethernet** odnosi się do zbioru standardów, opublikowanych po raz pierwszy w roku 1980 i zweryfikowanych dwa lata później, opracowanych przez firmy DEC, Intel i Xerox. Pierwsza ostatecznie uzgodniona wersja Ethernetu, zwana „Ethernetem 10-megowym" (*10Mb/s Ethernet* — od stosowanej szybkości transmisji) lub „Ethernetem

współdzielonym" (*shared Ethernet* — od sposobu wykorzystywania łącza), przyjęta została przez IEEE jako standard o numerze 802.3. Zasadę funkcjonowania sieci tego standardu zilustrowano na rysunku 3.1.



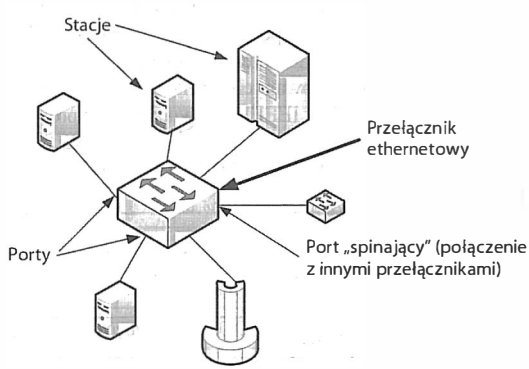
Rysunek 3.1. *W podstawowym wariancie sieć Ethernet obejmuje jedną lub kilka stacji, połączonych wspólnym segmentem kabla. Stacja rozpoczyna wysyłanie danych dopiero po upewnieniu się, że segment ten jest wolny (nie używa go żadna inna stacja). Nie wyklucza to jednak możliwości rozpoczęcia wysyłania danych przez kilka stacji równocześnie (głównie wskutek opóźnień w propagacji sygnałów) — mamy wówczas do czynienia ze zjawiskiem zwanym kolizją. Kolizje są prawidłowo wykrywane przez stacje; po wykryciu kolizji stacja staje się bezczynna na określony interwał czasowy, po czym ponawia próbę wysłania danych. Schemat ten nazywamy wielodostępem z badaniem kanału i wykrywaniem kolizji (CSMA/CD)*

Ponieważ w danej chwili współdzielony segment kablowy może obsłużyć co najwyżej jedną transmisję, dostęp do niego musi być synchronizowany. Synchronizacja ta realizowana jest przez rozproszony algorytm zaimplementowany w każdej stacji. Stacja zamierzająca wysłać dane rozpoczyna transmisję dopiero po upewnieniu się, że kabel nie jest zajęty (czyli nie trwa aktualnie transmisja na rzecz innej stacji). Ze względu na skończoną prędkość rozchodzenia się sygnałów w nośniku (i niezerowy czas samej transmisji) może się mimo wszystko zdarzyć, że dwie stacje (lub więcej) równocześnie rozpoczną wysyłanie danych — wystąpi wówczas zjawisko zwane **kolizją**. Jest ono niezawodnie wykrywane przez uwikłane w nie stacje (co gwarantuje wspomniany algorytm synchronizacyjny); każda ze stacji odczekuje wtedy pewien interwał czasowy (jego długość jest iloczynem wartości 51,2 milisekundy i wartością losową o rozkładzie równomiernym z przedziału (0 ; 1)) i ponawia próbę. Nie jest wykluczone, że ponowienie takie wystąpi kilkakrotnie — za każdym razem interwał bezczynności jest dwukrotnie dłuższy niż w poprzedniej próbie (interwały rosną w tempie wykładniczym, stąd scenariusz ten nosi nazwę **wykładniczej procedury wyczekiwania** — *exponential backoff procedure*). Ostatecznie stacja zamierzająca wysłać dane albo zakończy swój zamiar z powodzeniem, albo skapituluje po przekroczeniu pewnej maksymalnej liczby prób — w klasycznym Ethernecie jest to najwyżej 16 prób. Reasumując, w opisanym schemacie synchronizacja dostępu do współdzielonego nośnika obejmuje dwa zasadnicze elementy: testowanie zajętości wspomnianego nośnika oraz prawidłowe reagowanie na zaistniałe ewentualnie kolizje. Znajduje to odzwierciedlenie w nazwie samego schematu — **wielodostęp z badaniem nośnika i wykrywaniem kolizji**, po angielsku *carrier sense, multiple access with collision detection*, w skrócie CSMA/CD. Jego prostota stała się jednym z głównych czynników szerokiego rozpowszechnienia Ethernetu.

Generalnie metody synchronizacyjne, takie jak CSMA/CD, nazywane są formalnie protokołami **sterowania dostępem do nośnika** (*Media Access Control*, w skrócie MAC). Istnieje wiele różnych protokołów tego typu; niektóre z nich (jak CSMA/CD) funkcjonują na zasadzie rywalizowania o dostęp, inne opierają się na zaplanowanej koordynacji (realizowanej np. w oparciu o cykliczne przydzielanie „okien" czasowych poszczególnym stacjom).

Coraz szybsze komputery i wciąż rosnąca popularność Ethernetu sprawiły, że szybkość transmisji 10 Mb/s okazała się zbyt mała w stosunku do przysłowiowego apetytu rosnącego w miarę jedzenia. Na bazie coraz lepszego sprzętu i oprogramowania powstawały nowe wersje Ethernetu, o szybkościach 100 Mb/s, 1000 Mb/s, 10 Gb/s, 40 Gb/s i jeszcze więcej — 100 Gb/s jest obecnie (rok 2012) standardem rynkowym. A pomyśleć, że podstawą pierwszej, eksperymentalnej wersji Ethernetu była prędkość 3 Mb/s, którą standard DIX (akronim od nazw trzech twórców, firm DEC, Intel i Xerox) zwiększył do 10 Mb/s, w oparciu o zbiór segmentów kablowych połączonych za pomocą wzmacniaczy sygnału. Na początku lat 90. ubiegłego wieku współdzielony kabel wyparty został przez **skrętkę** (*twisted pair wiring*) przypominającą przewody telefoniczne i zwaną stąd popularnie „10BASE-T".

Gdy dostępne stały się realizacje o przepustowości 100 Mb/s — z „szybkim Ethernetem" (*fast Ethernet*) znanym również jako „100BASE-TX" na czele — protokół wielodostępu sterowanego rywalizacją (CSMA/CD) okazał się być mało atrakcyjny, co w konsekwencji doprowadziło do popularyzacji innego rozwiązania, czyli dedykowanych połączeń każdej stacji z **przełącznikiem** (*switch*), tworzących sieć o topologii gwiazdy, widocznej na rysunku 3.2.



Rysunek 3.2. *Przełączana (komutowana) sieć Ethernet, w której każda stacja połączona jest dedykowanym łączem z przełącznikiem. W większości przypadków sieć taka funkcjonuje w pełnym dupleksie, algorytm CSMA/CD okazuje się niepotrzebny. Sieci takie można rozbudowywać, łącząc poszczególne przełączniki za pomocą ich portów „spinających" (uplink ports)*

Obecnie przełączane sieci Ethernet (zwane także sieciami komutowanymi) są w powszechnym użyciu; każda stacja może niezależnie od innych wymieniać dane z przełącznikiem w obu kierunkach jednocześnie (co nosi nazwę pełnego dupleksu — *full duplex*) i choć operacje półdupleksowe (czyli transmisje tylko w jedną stronę w danej chwili) nadal są

obsługiwane nawet w wersji 1000 Mb/s („1000BASE-T"), to z możliwości tej korzysta się bardzo rzadko. Szczegółami przetwarzania ethernetowych PDU przez przełącznik zajmiemy się w dalszej części rozdziału.

Jednym z najpopularniejszych obecnie środków dostępu do Internetu są sieci bezprzewodowe; lokalna sieć bezprzewodowa (*Wireless LAN*, w skrócie WLAN) to jeden ze standardów IEEE znany pod nazwą *Wi-Fi* (od *Wireless Fidelity* — dosł. bezprzewodowa wierność), a także pod nazwą „bezprzewodowego Ethernetu" i pod oznaczeniem 802.11. Mimo iż standard ten różni się od standardów grupy 802 przewodowego Ethernetu, format jego ramki oraz ogólny interfejs zostały w dużym stopniu zapożyczone ze specyfikacji 802.3, dlatego też większość mechanizmów przewodowego Ethernetu wykorzystywanych przez protokoły TCP/IP stosowana jest również w sieciach Wi-Fi. Dalej w tym rozdziale przeanalizujemy je szczegółowo, tymczasem jednak przyjrzymy się szerzej grupie standardów IEEE 802 pozostających w ścisłym związku z sieciami domowymi i korporacyjnymi, przy okazji wspomnimy o standardach, jakimi rządzą się sieci miejskie (MAN, m.in. IEEE 802.16 — WiMAX), oraz o standardach niezależnego od nośników przekazywania transmisji w sieciach komórkowych (IEEE 802.21).

3.2.1. Standardy sieci LAN/MAN IEEE 802

Oryginalny format ramki Ethernetu (i algorytmy jego przetwarzania) to wynik uzgodnień grupy trzech wymienionych wcześniej producentów. Format ten znany jest pod akronimem DIX oraz nazwą „Ethernet II". Sieci oparte na tym formacie (po jego drobnych modyfikacjach), wykorzystujące synchronizację CSMA/CD, zostały przyjęte przez IEEE jako standard 802.3. Generalnie, w świecie IEEE przedrostek „802" identyfikuje standardy związane z sieciami lokalnymi i miejskimi; najpopularniejszymi przedstawicielami tej grupy są standardy 802.3 (podstawowy Ethernet) i 802.11 (sieci WLAN/Wi-Fi). Wraz z upływem czasu ulegały one naturalnej ewolucji, co spowodowało pojawienie się licznych ich wariantów, identyfikowanych odmiennymi nazwami (np. 802.11g). W tabeli 3.1 przedstawiono listę standardów IEEE 802 związanych z protokołami grupy TCP/IP.

Tabela 3.1. *Standardy IEEE 802 sieci LAN i MAN pozostające w związku z protokołami grupy TCP/IP (stan z połowy roku 2011)*

Nazwa	Opis	Oficjalna specyfikacja
802.1ak	Protokół wielorejestracyjny (*Multiple Registration Protocol* — MRP	[802.1Q-2011]
802.1AE	Bezpieczeństwo MAC (MACSec)	[802.1AE-2006]
802.1AX	Agregacja łączy (dawniej 802.3ad)	[802.1AX-2008]
802.1d	Mostki MAC	[802.1D-2004]
802.1p	Klasy/priorytet ruchu/QoS	[802.1D-2004]
802.1Q	Wirtualne sieci mostkowane, poprawki MRP	[802.1Q-2011]
802.1s	Protokół wielu drzew rozpinających (MSTP — *Multiple Spanning Tree Protocol*)	[802.1Q-2011]

Tabela 3.1. *Standardy IEEE 802 sieci LAN i MAN pozostające w związku z protokołami grupy TCP/IP (stan z połowy roku 2011) — ciąg dalszy*

Nazwa	Opis	Oficjalna specyfikacja
802.1w	Błyskawiczny protokół drzewa rozpinającego (*Rapid Spanning Tree Protocol* — RSTP)	[802.ID-2004]
802.1X	Kontrola dostępu do sieci bazująca na portach (*Port-Based Network Access Control* — PNAC)	[802.1X-2010]
802.2	Sterowanie połączeniem logicznym (LLC — *Logical Link Control*)	[802.2-1998]
802.3	Ethernet podstawowy i Ethernet 10 Mb/s	[802.3-2008] (sekcja 1)
802.3u	Ethernet 100 Mb/s („Fast Ethernet")	[802.3-2008] (sekcja 2)
802.3x	Operacje pełnodupleksowe i sterowanie przepływem	[802.3-2008]
802.3z/ 802.3ab	Ethernet 1000 Mb/s („Gigabit Ethernet")	[802.3-2008] (sekcja 3)
802.3ae	Ethernet 10 Gb/s („Ten-Gigabit Ethernet")	[802.3-2008] (sekcja 4)
802.3ad	Agregacja łączy	[802.1AX-2008]
802.3af	*Power over Ethernet* — PoE (do 15,4 W)	[802.3-2008] (sekcja 2)
802.3ah	*Access Ethernet* („Ethernet in the First Mile" — EFM)	[802.3-2008] (sekcja 5)
802.3as	Rozszerzenie formatu ramki (do 2000 bajtów)	[802.3-2008]
802.3at	Rozszerzenie *Power over Ethernet* (PoE+ — do 30 W)	[802.3at-2009]
802.3ba	Ethernet 40/100 Gb/s	[802.3ba-2010]
802.11a	Bezprzewodowe sieci LAN 54 Mb/s w paśmie 5 GHz	[802.11-2007]
802.11b	Bezprzewodowe sieci LAN 11 Mb/s w paśmie 2,4 GHz	[802.11-2007]
802.11e	Adaptacja QoS na gruncie 802.11	[802.11-2007]
802.11g	Bezprzewodowe sieci LAN 54 Mb/s w paśmie 2,4 GHz	[802.11-2007]
802.11h	Rozszerzenia zarządzania energią i spektrum	[802.11-2007]
802.11i	Ulepszenia bezpieczeństwa — następcy WEP	[802.11-2007]
802.11j	Wykorzystywanie pasma 4,9 – 5,0 GHz w Japonii	[802.11-2007]
802.11n	Sieci bezprzewodowe 5,6 – 600 Mb/s w pasmach 2,4 GHz i 5 GHz, z opcjonalnym wykorzystaniem MIMO i kanałów o szerokości 40 MHz	[802.11n-2009]
802.11s (szkic)	Sieci kratowe, kontrola przeciążenia	w przygotowaniu
802.11y	Bezprzewodowe sieci LAN 54 Mb/s w paśmie 3,7 GHz (licencjonowanym)	[802.11y-2008]

Tabela 3.1. *Standardy IEEE 802 sieci LAN i MAN pozostające w związku z protokołami grupy TCP/IP (stan z połowy roku 2011) — ciąg dalszy*

Nazwa	Opis	Oficjalna specyfikacja
802.16	Szerokopasmowe systemy dostępu bezprzewodowego (WiMAX)	[802.16-2009]
802.16d	Standard stacjonarnych bezprzewodowych sieci MAN	[802.16-2009]
802.16e	Standard stacjonarnych/mobilnych bezprzewodowych sieci MAN (WiMAX)	[802.16-2009]
802.16h	Poprawiony mechanizm współegzystencji	[802.16h-2010]
802.16j	Przekazywanie wieloskokowe w sieciach 802.16	[802.16j-2009]
802.16k	Mostkowanie w sieciach 802.16	[802.16k-2007]
802.21	Przekazywanie transmisji w sposób niezależny od nośnika	[802.21-2008]

Poza różnymi typami sieci LAN, definiowanymi przez standardy 802.3, 802.11 i 802.16, widzimy w powyższym zestawieniu standardy wspólne dla wszystkich sieci. Należy do nich m.in. standard 802.2 definiujący **sterowanie połączeniem logicznym** (LLC — *Logical Link Control*), w tym nagłówek ramki LLC wspólny dla formatów wszystkich sieci. W terminologii IEEE, LLC i MAC to „podwarstwy" warstwy łącza danych; format ramki LLC jest przy tym generalnie jednakowy dla wszystkich sieci, natomiast podwarstwa MAC jest dla poszczególnych sieci bardziej zróżnicowana — gdy np. oryginalny Ethernet wykorzystuje badanie nośnika z wykrywaniem kolizji (CSMA/CD), w sieciach bezprzewodowych często spotyka się konkurencyjny mechanizm **unikania** kolizji (CSMA/CA — *Carrier Sense, Multiple Access with Collision Avoidance*).

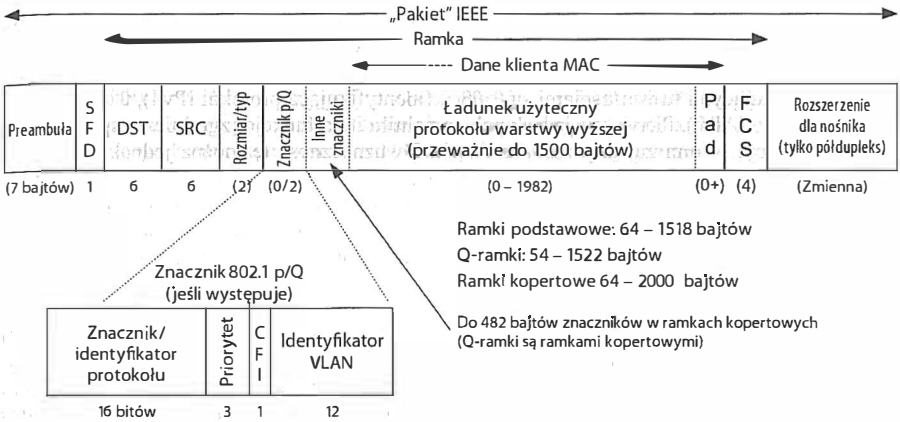
Początkowo kombinacja standardów 802.2 i 802.3 skutkowała definicją formatu ramki różnego od formatu Ethernet II; było tak do momentu ostatecznego unormowania sytuacji przez dokument [802.3-2008]. W świecie TCP/IP enkapsulacja datagramów IP w ramkach ethemetowych definiowana jest w dokumentach [RFC0894] i [RFC2464], choć pozostaje w mocy także dokument [RFC1042] definiujący starszy schemat enkapsulacji LLC/SNAP.

Od dłuższego czasu format ramki ethernetowej pozostaje ustabilizowany; aby lepiej zrozumieć jego ewolucję, przyjrzyjmy się jego szczegółom.

3.2.2. Format ramki ethernetowej

Format ramki ethernetowej, definiowany przez standard 802.3, ukształtował się jako wzbogacenie pierwowzoru o dodatkowe funkcje. Na rysunku 3.3 przedstawiono ów format wraz z ukazaniem, jak wpisuje się w stosunkowo nową koncepcję **pakietu IEEE** (koncepcję trochę niefortunną w świetle tendencji do nadużywania słowa „pakiet" w różnych znaczeniach).

Właściwa ramka poprzedzona jest obszarem **preambuły**, wykorzystywanym przez interfejs urządzenia odbiorczego do wykrywania nadejścia ramki oraz określenia dystansu czasowego między kolejnymi bitami — Ethernet jest technologią asynchroniczną: zegary poszczególnych interfejsów funkcjonują niezależnie od siebie, zatem wspomniany dystans



Rysunek 3.3. *Ramka ethernetowa w standardzie IEEE 802.3 zawiera pola adresów docelowego i źródłowego, dwuznaczne pole Rozmiar/Typ, pole danych i pole kodu kontrolnego CRC32 (FCS* — Frame Check Sequence*). Ten podstawowy format może być rozszerzony o znacznik zawierający identyfikator VLAN i informację o priorytecie (802.1 p/Q), a ostatnio dodatkowo dowolny zestaw innych znaczników. Preambuła i pole SFD służą do synchronizowania zegara w urządzeniu odbierającym ramkę. Przy transmisji półdupleksowej w Ethernecie 100 Mb/s lub szybszym krótkie ramki mogą być uzupełniane tzw. rozszerzeniem dla nośnika zapewniającym prawidłowe działanie mechanizmu wykrywania kolizji*

różni się nieco w poszczególnych interfejsach. Urządzenie odbiorcze musi niejako „odtworzyć" tempo zegara urządzenia nadawczego. W tym celu w preambule umieszcza się łatwo rozpoznawalny wzorzec, zwykle ciąg naprzemiennych bitów 1 i 0 (powtarzany bajt 0xAA). Preambuła oddzielona jest od ramki **separatorem początkowym** (SFD — *Start Frame Delimiter*), który ma postać bajta o ustalonej wartości 0xAB.

Zgodnie z oryginalną specyfikacją Ethernetu, bity ramki kodowane są przy użyciu **manchesterskiego kodowania fazy** (MPE — *Manchester Phase Encoding*) polegającego na przełączaniu między dwoma poziomami sygnału: +0,85 V i –0,85 V: przełączenie z potencjału ujemnego na dodatni oznacza bit 0, przełączenie w kierunku odwrotnym — bit 1. Potencjał 0V oznacza, że współdzielony nośnik jest aktualnie nieużywany. Powtarzany wzorzec preambuły 0xAA (binarnie 10101010) generuje więc falę prostokątną o częstotliwości 10 MHz. Ponieważ MPE wymaga dwóch cykli zegara dla każdego bitu, urządzenia Ethernetu 10 Mb/s muszą być wyposażone w oscylatory 20 MHz. W obecnie używanych wersjach Ethernetu metoda MPE wyparta została przez bardziej wydajne metody kodowania.

Ramka ethernetowa w podstawowym formacie rozpoczyna się od dwóch 48-bitowych (6-bajtowych) pól adresowych — **adresu docelowego** (DST) i **adresu źródłowego** (SRC). Adresy te często opatrywane są nazwami „adresów MAC", „adresów warstwy łącza danych", „adresów 802", „adresów sprzętowych" lub „adresów fizycznych". Adres docelowy może być także adresem identyfikującym grupę stacji (zamiast pojedynczej stacji), czyli adresem broadcast lub multicast (patrz rozdział 9.); adresy broadcast wykorzystywane są przez protokół ARP (piszemy o nim w rozdziale 4.), zaś adresy multicast wykorzystywane są przez protokół ICMPv6 (patrz rozdział 8.) do konwertowania adresów między warstwą sieciową a warstwą łącza danych.

Po dwóch polach adresowych może wystąpić *Znacznik p/Q*, którym za chwilę zajmiemy się szczegółowo. Kolejne, dwubajtowe pole identyfikuje *Typ ramki*, a dokładniej — typ danych stanowiących ładunek użyteczny: w kontekście protokołów TCP/IP najczęściej występującymi tu wartościami są 0x0800 (identyfikująca protokół IPv4), 0x86DD (IPv6) i 0x0806 (ARP). Pierwotnie jednak pole to pełniło inną funkcję: zgodnie ze specyfikacją 802.3 był w nim zawarty *rozmiar ramki*. Dwuznaczność tę można jednak łatwo rozstrzygnąć na podstawie *zawartości* tego pola. Maksymalny rozmiar ramki 802.3 wynosił 1518 bajtów, zatem wartość 1536 lub większa jest ewidentnie oznaczeniem typu; nieprzypadkowo zakres dozwolonych identyfikatorów typu rozpoczyna się właśnie od wartości 1536 (0x0600) (pełna lista zdefiniowanych identyfikatorów dostępna jest pod adresem [ETHERTYPES]).

Jak wspomnieliśmy przed chwilą, między polem adresu źródłowego (*SRC*) a polem *Typ/Rozmiar* może (choć nie musi) pojawić się czterobajtowe pole *Znacznika p/Q*. Opcjonalność tego pola stwarza kolejną niejednoznaczność — czy dwa bajty następujące bezpośrednio po polu *SRC* to pole *Typ/Rozmiar* czy może początkowe bajty *Znacznika p/Q*? Również i tę niejednoznaczność rozstrzyga się na podstawie zawartości wspomnianych bajtów: otóż wartość 0x8100 jest niezawodnie świadectwem początku znacznika, oznacza bowiem tzw. *Q-ramkę* (*Q-tagged Frame*), czyli ramkę zawierającą identyfikator wirtualnej sieci LAN, zgodnie ze standardem 802.1Q; wykluczone jest wystąpienie takiej wartości jako *Typu ramki*.

Reasumując, zawartość rzeczonych dwóch bajtów przesądza o tym, czy są one początkiem znacznika p/Q (0x8100), oznaczeniem typu ramki (wartość większa lub równa 1536), czy też określają rozmiar ramki (wartość różna od wymienionych).

Kolejne pole nagłówka ramki, zdefiniowane w specyfikacji [802.3-2008], zawiera zmienną liczbę znaczników dedykowanych protokołom innych standardów IEEE; znaczniki najczęściej używane związane są z wirtualnymi sieciami LAN i *jakością usługi* (QoS — *Quality of Service*) — omawiamy je w punkcie 3.2.3.

Obecnie obowiązujący standard Ethernetu, zdefiniowany w specyfikacji [802.3-2008], przewiduje do 482 bajtów znaczników obecnych w każdej ramce; rozmiar tak rozszerzonej ramki, zwanej ramką kopertową (*envelope frame*), może dochodzić do 2000 bajtów. Ramka zawierająca znaczniki standardu 802.1p/Q, zwana Q-ramką (*Q-tagged frame*), jest ramką kopertową (jednak nie wszystkie ramki kopertowe są Q-ramkami).

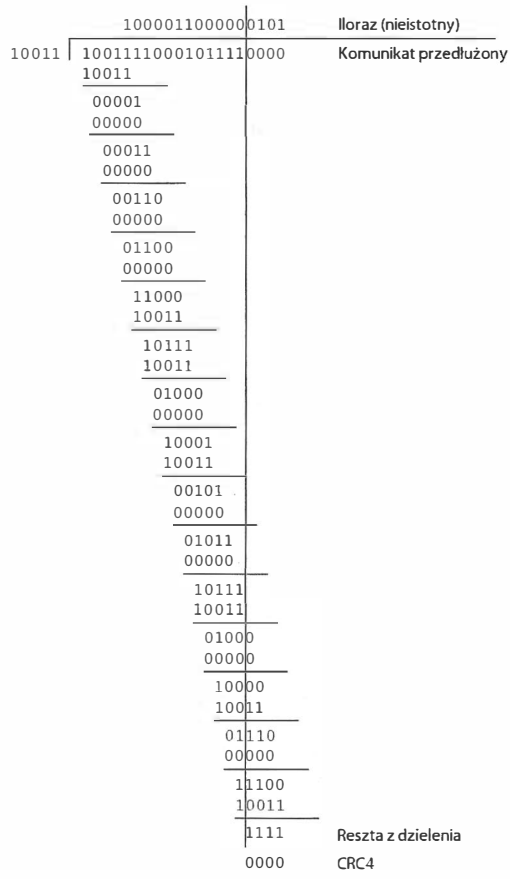
Po opisanych powyżej polach (składających się na umownie pojmowany *nagłówek*) następuje pole *ładunku użytecznego* (*payload*) — w tym polu enkapsulowane są jednostki protokołu warstwy wyższej, np. datagramy IP. Tradycyjnie pole to miało zawsze maksymalną dopuszczalną długość 1500 bajtów (stanowiącą MTU dla Ethernetu), choć — oczywiście — może być (z różnych względów) krótsze — wówczas jest często uzupełniane bajtami zerowymi dla zapewnienia wymaganego minimalnego rozmiaru ramki (powrócimy do tej kwestii w podpunkcie 3.2.2.2.

3.2.2.1. Ciąg kontrolny ramki (FCS) — nadmiarowa kontrola cykliczna (CRC)

Ostatnie pole ramki ethernetowej, następujące po ładunku użytecznym (i jego ew. dopełnieniu), to pole FCS (*Frame Check Sequence* — ciąg kontrolny ramki), którego zadaniem jest kontrola integralności informacji zawartej w ramce. Pole to zawiera 32-bitową

sumę kontrolną, obliczaną dla całej zawartości ramki jako **cykliczny kod nadmiarowy** (CRC — *Cyclic Redundancy Check*). Generalnie, obliczanie n-bitowego kodu CRC dla danego komunikatu danych rozpoczyna się od uzupełnienia tego komunikatu ciągiem n bitów o wartości 0 (otrzymujemy w ten sposób tzw. **komunikat przedłużony** — *augmented message*), po czym następuje obliczenie reszty z dzielenia modulo 2 tegoż przedłużonego komunikatu przez (n+1)-bitowy dzielnik, zwany **generatorem wielomianowym** (*generator polynomial*); iloraz otrzymany w tym dzieleniu nie ma znaczenia. Bitowa negacja wspomnianej reszty daje ostateczni przedmiotowy kod CRC. Zalecane wartości generatora wielomianowego są zestandaryzowane dla wielu wartości n; dla n = 32 jest to 33-bitowy ciąg 100000100110000010001110110110111.

Aby lepiej zrozumieć tę procedurę, na rysunku 3.4 zilustrowaliśmy obliczanie 4-bitowego kodu CRC dla przykładowego, 16-bitowego komunikatu 1001111000101111; dzielnik dla kodu CRC4 ustalony został przez ITU na wartość 10011 w standardzie G.704 (patrz [G704]). Negując otrzymaną resztę 1111, otrzymujemy kod CRC w postaci 0000.



Rysunek 3.4. *Dzielenie modulo 2 ilustrujące sposób obliczania 4-bitowego kodu CRC*

Obliczona wartość CRC wpisana zostaje przez nadawcę ramki do jej pola FCS. Odbiorca ramki przeprowadza obliczenie identyczne z opisanym powyżej i porównuje otrzymany wynik z tym, który jest zawarty w polu FCS ramki; jeżeli porównywane wartości są różne, oznacza to uszkodzenie ramki podczas przesyłania i (zazwyczaj) jej odrzucenie przez odbiorcę. Funkcje CRC obliczane zgodnie z przedstawionym algorytmem znakomicie spisują się w roli weryfikatorów integralności danych, bo każda zmiana wzorca bitowego tychże danych z dużym prawdopodobieństwem prowadzi do zmiany wartości CRC obliczanego na ich podstawie.

3.2.2.2. Rozmiary ramek

Rozmiar ramki ethernetowej jest ograniczony zarówno pod względem wartości minimalnej, jak i wartości maksymalnej. Minimalny rozmiar to 64 bajty, z czego samorzutnie wynika minimalny rozmiar ładunku użytecznego (bez znaczników) — 48 bajtów; jeżeli pole ładunku użytecznego jest krótsze, zostaje prawostronnie dopełnione do tego rozmiaru ciągiem bajtów zerowych.

Powodem ustanowienia minimalnego rozmiaru ramki są względy niezawodności w wykrywaniu kolizji ramek, przy użyciu CSMA/CD w Ethernecie 10 Mb/s. Generalnie chodzi o to, by czas transmisji ramki nie był krótszy niż czas propagacji sygnału od nadawcy do odbiorcy i z powrotem; w razie wystąpienia kolizji wiadomo wówczas, o którą ramkę chodzi — o ramkę aktualnie transmitowaną. Stacja wykrywająca kolizję emituje wtedy impuls blokujący (*jamming signal*) w postaci wysokiego potencjału, co powoduje wejście innych stacji uwikłanych w kolizję w stan oczekiwania (o czym wcześniej pisaliśmy). Wobec górnego ograniczenia 2500 m na długość kabla (5 segmentów, każdy po 500 m, spiętych 4 wzmacniaczami) i orientacyjnej prędkości elektronów w miedzi (0,77 prędkości światła, czyli ok. 231 000 000 m/s) otrzymujemy minimalną wartość czasu transmisji ramki $\frac{2 \cdot 2500 \text{ m}}{231\,000\,000 \text{ m/s}} \approx 21{,}65 \cdot 10^{-6}$ s. Przy częstotliwości 10 Mb/s, czyli 10^{-7} s na jeden bit, jest to czas przesyłania ok. 216 bitów, czyli ok. 27 bajtów. Projektanci Ethernetu dla bezpieczeństwa podwoili tę wartość i zaokrąglili ją w górę do najbliższej potęgi liczby 2, czyli 64 bajtów.

Maksymalny rozmiar ramki konwencjonalnego Ethernetu to 1518 bajtów (maksymalnie 1500 bajtów danych, plus 14-bajtowy nagłówek, plus 4-bajtowy kod FCS). Wartość ta jest wynikiem pewnego kompromisu. Z jednej strony, uszkodzenie ramki podczas transportu wymaga jej ponownego przesłania — mniejsza ramka to ponowna transmisja mniejszej liczby bajtów. Z drugiej jednak strony, przesyłanie większych komunikatów wymaga dzielenia ich między kilka ramek; przykładowo pakiet o wielkości 64 kB (to wielkość typowa dla protokołów TCP/IP) wymaga co najmniej 44 ramek. I tu pojawia się pewien problem.

Kolejne ramki nie mogą być przesyłane w sposób ciągły, lecz muszą być przedzielane krótkimi pauzami — wymaga tego sprzęt w celu właściwego wyodrębniania ramek z nośnika, jest to również konieczne do tego, by poszczególne stacje miały możliwość „przeplatania" nawzajem swych transmisji. Specyfikacja Ethernetu II, oprócz poprzedzenia ramki 7-bajtową preambułą i jednobajtowym znacznikiem SFD, wymaga także tzw. *przerwy międzypakietowej* (IPG — *Inter-Packet Gap*) równoważnej czasowi przesyłania 12 bajtów (9,6 μs przy transmisji 10 Mb/s, 960 ns przy transmisji 100 Mb/s, 96 ns przy

transmisji 1000 Mb/s i 9,6 ns przy transmisji 10 000 Mb/s). Narzut na 1500-bajtowy ładunek użyteczny wynosi więc 38 bajtów (12 bajtów IPG, 7-bajtowa preambuła, bajt SFD, 14 bajtów nagłówka i 4-bajtowe pole FCS), co daje wydajność transmisji maksymalnie $\frac{1500}{1500+38} \approx 98\%$. Wynik ten można poprawić, wykorzystując tzw. *ramki jumbo* jako niestandardowe rozszerzenie Ethernetu, implementowane głównie w przełącznikach 1000 Mb/s (patrz [JF]), zwiększające MTU do wartości 9000 bajtów. W niektórych środowiskach przetwarzane są nawet ramki *superjumbo* — większe niż 9000 bajtów. Należy jednak z rozwagą wykorzystywać tę możliwość, bo owe monstrualne ramki nie są kompatybilne ze standardowymi ramkami 1518-bajtowymi, wykorzystywanymi przez większość starszego wyposażenia Ethernetu.

3.2.3. 802.1p/Q sieci wirtualne i znaczniki QoS

W sieciach LAN opartych na „przełączanym" Ethernecie dwa dowolne komputery należące do tej samej sieci mogą nawiązywać komunikację przy użyciu protokołu IP i innych protokołów warstwy sieciowej, bez żadnych dodatkowych zabiegów konfiguracyjnych (lub przy minimalnej interwencji administratora). Ponadto transmisje broadcast i multicast (patrz rozdział 9.) trafiają do wszystkich komputerów sieci bez potrzeby stosowania specjalnych protokołów. To rozwiązanie wygodne, lecz nie jest wolne od wad. Przede wszystkim transmisje broadcast generują w sieci znaczny ruch, ponadto względy bezpieczeństwa mogą dyktować ograniczenie swobody komunikowania się poszczególnych hostów ze sobą.

Z myślą o rozwiązaniu tych problemów, dotkliwych zwłaszcza w wielodostępnych sieciach LAN z dużą liczbą przełączników i hostów, IEEE rozszerzyła rodzinę standardów 802 LAN o koncepcję wirtualnych sieci LAN (*Virtual LAN*, w skrócie VLAN), zdefiniowaną jako standard 802.1Q (patrz [802.1Q-2005]). Przełączniki zgodne z tym standardem oddzielają od siebie transmisje należące do poszczególnych sieci VLAN. Zauważmy, że w związku z tym dwa komputery należące do różnych VLAN mogą komunikować się ze sobą jedynie za pośrednictwem routera, *nawet jeśli podłączone są do tego samego przełącznika*; fakt ten stał się przyczyną popularności urządzeń łączących obie funkcje — routera i przełącznika. W międzyczasie wydajność samych routerów znacząco wzrosła do stopnia porównywalnego z wydajnością przełączanych sieci VLAN; choć przyćmiło to wyraźnie blask tych ostatnich, nadal są w powszechnym użyciu, warto więc zapoznać się z zasadami ich funkcjonowania.

Różne mogą być *kryteria* przydzielania stacji roboczych do poszczególnych VLAN. Najprostsze i najczęściej stosowane bazuje na portach przełączników: poszczególne porty przyporządkowywane są a priori do poszczególnych VLAN, a stacja przyłączona do portu staje się automatycznie częścią jego VLAN. Innym kryterium przynależności do VLAN może być podział w oparciu o numery MAC poszczególnych stacji i utrzymywanej w przełączniku tablicy ich odwzorowania na identyfikatory VLAN; staje się ono jednak kłopotliwe, gdy zmienić trzeba adres MAC którejś stacji (wskutek zachowania pewnych użytkowników). Inne jeszcze kryterium bazować może na adresach IP stacji.

Gdy do jednego przełącznika podłączonych jest kilka stacji należących do różnych VLAN, implementacja tego przełącznika powinna zapewniać odpowiednią „szczelność", czyli uniemożliwiać przepływ ruchu między komputerami należącymi do różnych VLAN.

W sytuacji gdy sieci VLAN rozciągają się na wiele przełączników (co powszechnie nazywane jest *trunkingiem*), konieczne jest etykietowanie poszczególnych ramek identyfikatorami VLAN, do których ramki te należą — w przeciwnym razie informacja o podziale sieci LAN na poszczególne VLAN byłaby po prostu gubiona na drodze między przełącznikami. Etykietowanie to realizowane jest przy użyciu 12-bitowego pola identyfikacyjnego, zawartego w nagłówku znaczników 802.1p/Q (patrz rysunek 3.3). W wielu przypadkach funkcja trunkingu (czyli właściwa obsługa ramek zawierających wspomniany nagłówek) musi zostać jawnie włączona przez administratora na poszczególnych portach. By miał on nieco łatwiejsze zadanie w tym względzie, wiele przełączników realizuje koncepcję *natywnej sieci VLAN*: do sieci tej zaliczane są automatycznie wszystkie ramki nieposiadające wspomnianych znaczników. Porty trunkingowe przełączników służą do łączenia tychże w kontekście realizacji koncepcji VLAN, podczas gdy do pozostałych portów przyłącza się stacje robocze. Należy w tym miejscu zaznaczyć, że różni producenci przełączników implementować mogą specyficzne metody realizacji trunkingu, czego przykładem może być protokół *Inter-Switch Link* (ISL) firmy Cisco.

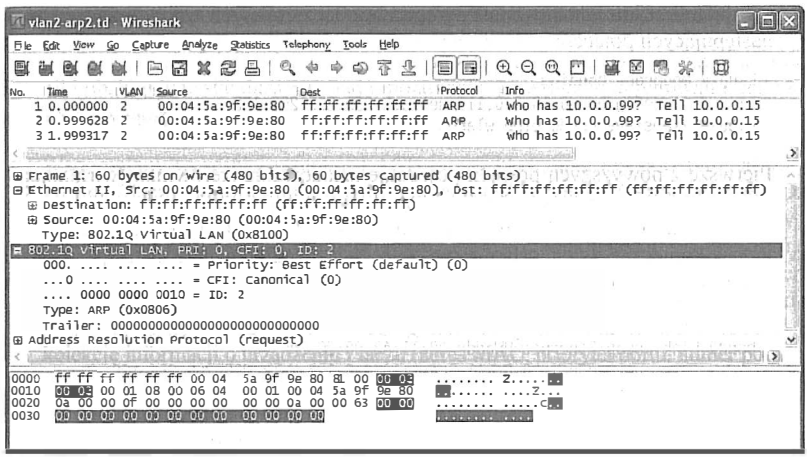
W skład znaczników 802.1p/Q, poza identyfikatorem VLAN, wchodzi m.in. 3-bitowe pole *Priorytet*, określające klasę tzw. jakości usługi (QoS — *Quality of Service*) zgodnie z rozszerzenie standardu 802.1Q; 3 bity pozwalają na zdefiniowanie ośmiu klas, od 0 oznaczającego tzw. ruch niegwarantowany (czyli ruch, w którym podejmowany jest uczciwy wysiłek z zamiarem dostarczenia, lecz bez gwarancji efektu — tzw. *best-effort delivery*), do 7 reprezentującej ramki krytyczne z punktu widzenia zarządzania siecią. Poza umiejscowieniem tego pola w strukturze ramki (widocznej na rysunku 3.3), standard 802.1Q nie precyzuje jednak ani zasad opatrywania ramek poszczególnymi klasami QoS, ani sposobu interpretowania tychże klas w implementacjach Ethernetu — kwestie te pozostawione zostały w gestii producentów i w przypadku dwóch różnych przełączników czy routerów mogą prezentować się dość odmiennie. Zauważmy, że znaczniki QoS są niezależne od mechanizmu VLAN, bo zajmują we wspomnianym nagłówku rozłączne obszary.

W systemie Linux zarządzanie informacją związaną ze standardami 802.1p/Q odbywa się za pomocą polecenia `vconfig`, które umożliwia definiowanie identyfikatorów VLAN dla poszczególnych interfejsów fizycznych oraz anulowanie poczynionych definicji. Przy jego użyciu można także manipulować priorytetami QoS, zmieniać konwencję identyfikowania wirtualnych interfejsów, ustanawiać związki między pakietami zawierającymi różne identyfikatory VLAN oraz przypisywać poszczególnym identyfikatorom priorytety obsługi w implementacjach protokołów w systemie operacyjnym. W poniższym scenariuszu widzimy kolejno zdefiniowanie wirtualnego interfejsu `VLAN ID 2` dla fizycznego interfejsu `eth1`, anulowanie tej definicji, zmianę konwencji nazewniczej dla interfejsów wirtualnych i zdefiniowanie nowego interfejsu wirtualnego:

```
Linux# vconfig add eth1 2
Added VLAN with VID == 2 to IF -:eth1:-
Linux# ifconfig eth1.2
eth1.2 Link encap:Ethernet HWaddr 00:04:5A:9F:9E:80
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
Linux# vconfig rem eth1.2
Removed VLAN -:eth1.2:-
Linux# vconfig set_name_type VLAN_PLUS_VID
```

```
Set name-type for VLAN subsystem. Should be visible in
          /proc/net/vlan/config
Linux# vconfig add eth1 2
Added VLAN with VID == 2 to IF -:eth1:-
Linux# ifconfig vlan0002
vlan0002 Link encap:Ethernet HWaddr 00:04:5A:9F:9E.80
          BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 b) TX bytes:0 (0.0 b)
```

Jak widzimy, domyślna dla Linuksa konwencja tworzenia nazw dla interfejsów wirtualnych polega na konkatenacji nazwy interfejsu fizycznego, kropki i identyfikatora przypisanego interfejsu wirtualnego. Przykładowo interfejs wirtualny o identyfikatorze 2 utworzony dla fizycznego interfejsu `eth1` otrzymuje nazwę `eth1.2`. Konwencja alternatywna polega na enumeracji poszczególnych sieci VLAN za pomocą schematu `vlan<n>`, gdzie *<n>* jest identyfikatorem interfejsu wirtualnego. Od momentu przypisania interfejsowi fizycznemu interfejsu wirtualnego identyfikator tego ostatniego wbudowywany będzie w ramki wysyłane za pośrednictwem interfejsu fizycznego. Można się o tym przekonać, analizując ruch sieciowy z wykorzystaniem programu Wireshark, którego okno widoczne jest na rysunku 3.5.



Rysunek 3.5. *Etykietowanie wysyłanych ramek identyfikatorem interfejsu wirtualnego, obserwowane w oknie programu Wireshark; zmieniono domyślny układ wyświetlania tak, by wyświetlane były identyfikatory VLAN i „surowe" adresy ethernetowe*

Analizując dokładniej postać przesyłanej ramki, odnajdziemy w niej ładunek użyteczny w postaci pakietu ARP (patrz rozdział 4.). Wartość 0x8100 w bajtach następujących bezpośrednio po polu SRC wskazuje na to, iż jest to początek *Znacznika p/Q* (co wyjaśnialiśmy w punkcie 3.2.2), z którego odczytać możemy identyfikator interfejsu wirtualnego VLAN (2) oraz klasę QoS (0). Rozmiar ramki wynosi 60 bajtów (wartość ta nie obejmuje pola *FCS*). Wartość 0x0806 w polu *Rozmiar/Typ* oznacza, że ładunek użyteczny ma postać typową dla enkapsulowanego pakietu ARP.

3.2.4. 802.1AX: agregowanie łączy (dawniej 802.3ad)

Niektóre urządzenia wyposażone w wiele interfejsów sieciowych realizują funkcje *agregowania łączy*, zwaną także ich *spajaniem* (*bonding*). Istotą tej funkcji jest traktowanie zbioru dwóch lub więcej interfejsów jako pojedynczego interfejsu w celu uzyskania większej niezawodności (poprzez redundancję, czyli przesyłanie tych samych danych przez kilka interfejsów równocześnie) lub większej wydajności (poprzez rozdział przesyłanych danych na kilka interfejsów fizycznych). Poprawka *IEEE Amendment 802.1AX* [802.1AX-2008] definiuje w tym celu protokół LACP (*Link Aggregation Control Protocol* — protokół sterowania agregacją łączy); protokół ten wykorzystuje ramki IEEE 802 w specjalnym formacie, określanym akronimem LACPDU (*Link Aggregation Control Protocol Data Unit*).

Wykorzystywanie przełączników realizujących agregację łączy może być tańszą alternatywą dla drogich przełączników wyposażonych w porty o bardzo dużej przepustowości — zbiór zagregowanych łączy zdolny jest zapewnić porównywalną przepustowość. Zazwyczaj wymaga się, by agregowane porty były tego samego typu i działały w tym samym trybie (półdupleksowym albo pełnodupleksowym). Agregację łączy realizować można nie tylko w ramach przełączników, lecz również na bazie wielu **kart sieciowych** (NIC — *Network Interface Cards*) zainstalowanych w tym samym komputerze.

W systemie Linux możliwe jest agregowanie łączy różnych typów urządzeń za pomocą następujących poleceń:

```
Linux# modprobe bonding
Linux# ifconfig bond0 10.0.0.111 netmask 255.255.255.128
Linux# ifenslave bond0 eth0 wlan0
```

Pierwsze z powyższych poleceń powoduje załadowanie sterownika (wirtualnego) urządzenia realizującego agregację. Rezultatem drugiego polecenia jest utworzenie interfejsu `bond0` opatrzonego konkretnym adresem IPv4; użycie adresu IP, choć typowe, nie jest jednak konieczne. Istotą ostatniego z poleceń jest zagregowanie interfejsów `eth0` i `wlan0` w postaci wirtualnego interfejsu `bond0` — ten ostatni etykietowany jest jako nadrzędny (`MASTER`), któremu podporządkowane (`SLAVE`) są oba interfejsy źródłowe:

```
bond0 Link encap:Ethernet HWaddr 00:11:A3:00:2C:2A
          inet addr:10.0.0.111 Bcast:10.0.0.127 Mask:255.255.255.128
          inet6 addr: fe80::211:a3ff:fe00:2c2a/64 Scope:Link
          UP BROADCAST RUNNING MASTER MULTICAST MTU:1500 Metric:1
          RX packets:2146 errors:0 dropped:0 overruns:0 frame:0
          TX packets:985 errors:0 dropped:0 overruns:0 carrier:0
          collisions:18 txqueuelen:0
          RX bytes:281939 (275.3 KiB) TX bytes:141391 (138.0 KiB)
eth0 Link encap:Ethernet HWaddr 00:11:A3:00:2C:2A
          UP BROADCAST RUNNING SLAVE MULTICAST MTU:1500 Metric:1
          RX packets:1882 errors:0 dropped:0 overruns:0 frame:0
          TX packets:961 errors:0 dropped:0 overruns:0 carrier:0
          collisions:18 txqueuelen:1000
          RX bytes:244231 (238.5 KiB) TX bytes:136561 (133.3 KiB)
          Interrupt:20 Base address:0x6c00
wlan0 Link encap:Ethernet HWaddr 00:11:A3:11:13:8A
          UP BROADCAST SLAVE MULTICAST MTU:1500 Metric:1
          RX packets:269 errors:0 dropped:0 overruns:0 frame:0
```

```
TX packets:24 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
RX bytes:38579 (37.6 KiB) TX bytes:4830 (4.7 KiB)
```

Powyższy przykład ilustruje agregowanie przewodowego łącza `eth0` z bezprzewodowym łączem `wlan0`; będący wynikiem tej agregacji interfejs `bond0` otrzymuje adresy IPv4 (które zazwyczaj przypisuje się rzeczywistym, fizycznym interfejsom) oraz adres MAC, domyślnie będący adresem MAC pierwszego z agregowanych łączy. Gdy ruch IPv4 skierowany zostanie do wirtualnego interfejsu `bond0`, istnieje kilka możliwości rozdysponowania go pomiędzy agregowane interfejsy `eth0` i `wlan0`. W Linuksie wybór konkretnej możliwości wykonywany jest na etapie polecenia ładującego sterownik agregacji — i tak do dyspozycji jest m.in. cykliczne (*round-robin*) przełączanie między interfejsami, wykorzystanie jednego interfejsu jako duplikatu (*backup*) drugiego, wybór interfejsu na podstawie wyniku operacji XOR wykonanej na adresie MAC źródłowym i docelowym, kopiowanie ramek do obu interfejsów oraz agregację według standardu 802.3ad. Wariant z duplikowaniem interfejsu stosowany jest w sieciach, których dostępność jest czynnikiem krytycznym: w razie awarii jednego z interfejsów (wykrywanej za pomocą monitorowania MII, szczegóły patrz [BOND]) jego funkcje kontynuuje drugi, redundantny. Istotą wariantu z operacją XOR, stosowanego przy wystarczająco dużej liczbie interfejsów fizycznych, jest dedykowanie konkretnego interfejsu fizycznego dla komunikacji między dwoma konkretnymi hostami: uzyskujemy w ten sposób dość dobre równoważenie obciążenia poszczególnych interfejsów fizycznych za cenę niewielkiego narzutu związanego z dysponowaniem nimi w ramach agregacji. Kopiowanie ramek do wielu interfejsów jednocześnie wydatnie zwiększa odporność sieci na awarie, natomiast przełączniki obsługujące standard 802.3ad umożliwiają dynamiczne agregowanie jednorodnych zbiorów łączy.

Wspomniany wcześniej protokół LACP zaprojektowano z myślą o automatyzacji agregowania łączy, która eliminowałaby konieczność ręcznego konfigurowania. Działanie protokołu opiera się na wysyłaniu przez urządzenie pakietów LACPDU przez wszystkie łącza, na których aktywowano protokół. Para komunikujących się urządzeń, implementujących protokół LACP, może w ten sposób zidentyfikować wszystkie łącza realizujące komunikację między nimi i zagregować te łącza do postaci jednej wiązki zwanej *grupą agregacji* (LAG — *Link Aggregation Group*). Każda ramka LACPDU zawiera kolekcję informacji o urządzeniu i łączu (adres MAC, priorytet portu, numer portu i klucz). Czytelników zainteresowanych szczegółami protokołu odsyłamy do oryginalnej definicji standardu [802.1AX-2008].

3.3. Pełny dupleks, oszczędzanie energii, autonegocjowanie i sterowanie przepływem 802.1X

Funkcjonowanie początkowej wersji Ethernetu opierało się na półdupleksowej transmisji poprzez współdzielone segmenty kabla. W danej chwili dane mogły być przesyłane tylko w jednym kierunku i tylko przez jedną stację. Gdy pojawił się „przełączany" Ethernet, sieci przestały być segmentami kabli i stały się zbiorami łączy; w rezultacie wiele par urządzeń mogło odtąd niezależnie wymieniać informację i to w obu kierunkach (czyli w trybie pełnodupleksowym), zniknął też problem kolizji i ich wykrywania. Zniknięcie

ograniczeń czasowych wynikających z trybu półdupleksowego pozwoliło też na poszerzenie fizycznych rozmiarów sieci.

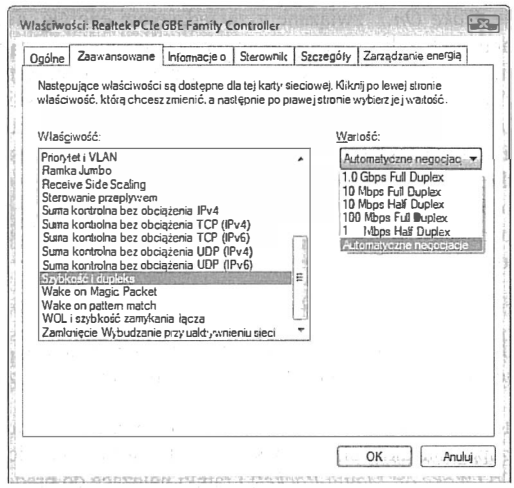
Możliwości urządzeń w zakresie transmisji pełnodupleksowej, fakt wykorzystywania tych możliwości, a także wiele innych cech interfejsu ethernetowego poznać można przy użyciu linuksowego polecenia `ethtool`:

```
Linux# ethtool eth0
Settings for eth0:
          Supported ports: [ TP MII ]
          Supported link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          Supports auto-negotiation: Yes
          Advertised link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          Advertised auto-negotiation: Yes
          Speed: 10Mb/s
          Duplex: Half
          Port: MII
          PHYAD: 24
          Transceiver: internal
          Auto-negotiation: on
          Current message level: 0x00000001 (1)
          Link detected: yes
Linux# ethtool eth1
Settings for eth1:
          Supported ports: [ TP ]
          Supported link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          1000baseT/Full
          Supports auto-negotiation: Yes
          Advertised link modes: 10baseT/Half 10baseT/Full
          100baseT/Half 100baseT/Full
          1000baseT/Full
          Advertised auto-negotiation: Yes
          Speed: 100Mb/s
          Duplex: Full
          Port: Twisted Pair
          PHYAD: 0
          Transceiver: internal
          Auto-negotiation: on
          Supports Wake-on: umbg
          Wake-on: g
          Current message level: 0x00000007 (7)
          Link detected: yes
```

Jak wynika z powyższego raportu, interfejs `eth0` przyłączony jest do półdupleksowej sieci 10 Mb/s. Widzimy także, iż urządzenie obsługuje funkcję *autonegocjacji* — mechanizm ten, wywodzący się ze standardu 802.3u, polega na wymianie między interfejsami informacji o własnych możliwościach: szybkości transmisji, obsługiwanym trybie transmisji (pół- lub pełno dupleksowym) itp. Autonegocjacja realizowana jest na poziomie warstwy fizycznej, przy użyciu sygnałów wymienianych w przerwach między transmisją „właściwych" danych. Z kolei interfejs `eth1` również obsługuje autonegocjację, przyłączony jest jednak do sieci 100 Mb/s operującej w trybie pełnodupleksowym. Pozostałe parametry (`Port`, `PHYAD`, `Transceiver`) identyfikują fizyczny typ portu, jego adres i umiejsco-

wienie obwodów warstwy fizycznej (wewnątrz karty sieciowej albo na zewnątrz niej). Parametr `Current message level` specyfikuje poziom szczegółowości (zapisywanych w dzienniku) komunikatów związanych z operacjami interfejsu. Znaczenie parametru `Wake-on` wyjaśnimy w jednym z następnych punktów.

W systemie Windows analogiczne informacje uzyskać można jako właściwości połączenia sieciowego. Uruchamiając odpowiednie okno dialogowe za pośrednictwem menu kontekstowego konkretnego połączenia, klikamy przycisk *Konfiguruj* i w kolejnym oknie przechodzimy na zakładkę *Zaawansowane* (patrz rysunek 3.6).



Rysunek 3.6. *Konfigurowanie zaawansowanych właściwości sterownika urządzenia sieciowego w MS Windows 7*

Na rysunku widzimy konfigurowanie szybkości i trybu dupleksu łącza; oprócz zestawu ustalonych trybów dostępna jest opcja autonegocjacji. Możliwe jest konfigurowanie także wielu innych aspektów interfejsu, m.in. poszczególnych znaczników standardów 802.1p/Q oraz opcji wybudzania (*Wake-on*), którą omawiamy szczegółowo w punkcie 3.3.2.

3.3.1. Niezgodność dupleksowa

Opcja autonegocjacji może sprawiać pewne problemy ze współdziałaniem urządzeń w sytuacji różnego skonfigurowania połączonych portów: tego w komputerze i tego w przełączniku, np. włączenia autonegocjacji tylko na jednym interfejsie lub ustalenia pełnego dupleksu na jednym porcie i półdupleksu na drugim. Zjawisko takie nazywamy **niezgodnością dupleksową** (*duplex mismatch*); co ciekawe, nie powoduje ono kompletnego załamania połączenia, lecz tylko (i to nie zawsze) degradację jego wydajności. Otóż, gdy sieć obciążona jest w znacznym stopniu (np. przesyłaniem dużego strumienia danych), dotarcie danych do półdupleksowego portu wysyłającego właśnie dane potraktowane zostanie jako kolizja (zgodnie z protokołem CSMA/CD) uruchamiająca wykładniczą procedurę wyczekiwania i skutkująca prawdopodobną utratą ramki; wymaga to ponowienia transmisji ze strony protokołu warstwy wyższej (np. TCP). Notorycznie powtarzające się kolizje mogą powodować drastyczne pogorszenie wydajności. Zauważmy

jednak, że przy małym obciążeniu sieci nawet port półdupleksowy dobrze poradzi sobie z sytuacją, gdyż prawdopodobieństwo wystąpienia kolizji będzie niewielkie. To ciekawe zjawisko jest przedmiotem badań i analiz; w publikacji [SC05] prezentowane są narzędzia służące do jego wykrywania.

3.3.2. Wybudzanie przez sieć (WoL), oszczędzanie energii i magiczne pakiety

W obu przedstawionych przykładach — tym z Linuksa i tym z Windows 7 — widzimy opcje *Wake On...* związane z zarządzaniem energią, a dokładniej — „wybudzaniem" komputera (czyli automatycznym przechodzeniem do stanu aktywności) wykonywanym z inicjatywy interfejsu sieciowego. Istotą tych opcji jest określenie charakterystyki pakietów, których dotarcie do interfejsu skutkuje taką właśnie akcją. W Linuksie charakterystykę tę definiuje się w postaci zbioru oznaczeń literowych — do wyboru mamy: dowolny pakiet otrzymany z warstwy fizycznej (`p`), ramkę unicast adresowaną do interfejsu (`u`), ramkę multicast docierającą do interfejsu (`m`), ramkę rozgłoszeniową (`b`), ramkę ARP (`a`), dowolną „magiczną" ramkę (`g`) lub „magiczną" ramkę zawierającą określone hasło. Wspomniany zbiór oznaczeń jest argumentem polecenia `ethtool`, przykładowo polecenie w postaci:

```
ethtool -s eth0 wol umgb
```

konfiguruje interfejs `eth0` do wybudzania komputera w przypadku nadejścia ramki należącej do którejkolwiek z kategorii identyfikowanych oznaczeniami `u`, `m`, `g` lub `b`.

System Windows oferuje podobną możliwość, jednak standardowy interfejs użytkownika nie zapewnia takiej elastyczności jak w Linuksie, dając do dyspozycji jedynie „magiczne" ramki (*Wake on Magic Packet*) i ramki należące do predefiniowanego zbioru {`u`, `m`, `b`, `a`} (*Wake on pattern match*).

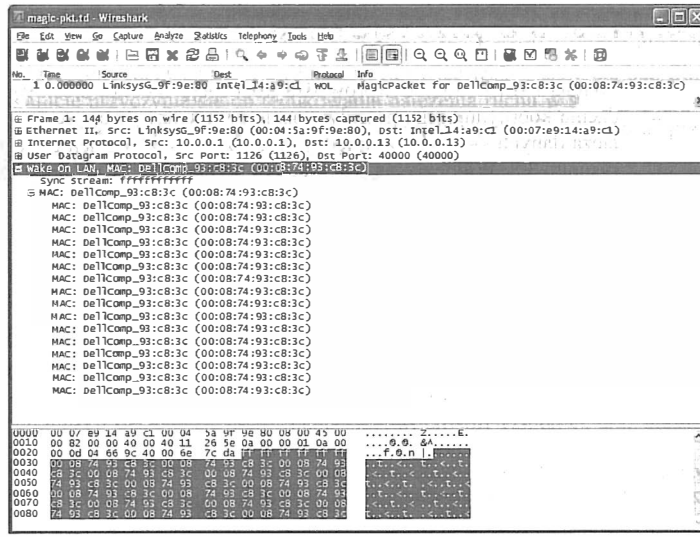
„Magiczną" ramką jest ramka zawierająca specjalny wzorzec powtarzających się bajtów `0xff`; jest to najczęściej ramka rozgłoszeniowa Ethernetu, przenosząca pakiet UDP (patrz rozdział 10.). Ramki takie generować można na różne sposoby, m.in. za pomocą programu `wol` (patrz [WOL]). I tak w poniższym scenariuszu:

```
Linux# wol 00:08:74:93:C8:3C
Waking up 00:08:74:93:C8:3C...
```

generowana jest magiczna ramka, której zawartość podglądnąć możemy za pomocą programu Wireshark (patrz rysunek 3.7). Ładunek użyteczny tej ramki wygląda na typowy pakiet UDP, choć numery portów (1126 i 40000) wybrane są arbitralnie. Specyficzny jest tu wzorzec tworzony przez 6 bajtów `0xff`, po których następuje 16-krotne powielenie docelowego adresu MAC `00:08:74:93:C8:3C`.

3.3.3. Sterowanie przepływem w warstwie łącza danych

Transmisja w trybie pełnodupleksowym między segmentami sieci o różnej prędkości może wiązać się z koniecznością buforowania ramek w przełącznikach; tak dzieje się np. w sytuacji, gdy wiele stacji równocześnie wysyła ramki do tego samego portu przeznaczenia



Rysunek 3.7. *„Magiczna" ramka rozpoczynająca się sześcioma bajtami 0xff, po których następuje 16-krotne powtórzenie adresu MAC*

(co nazywane jest powszechnie „rywalizacją o port" — *port contention*). Gdy tempo przybywania ramek do przełącznika przez dłuższy czas przewyższa możliwości ich przetwarzania przez ten przełącznik, w pewnym momencie pojemność bufora przełącznika wyczerpuje się i przybywające ramki są odrzucane.

Jedynym praktycznym rozwiązaniem tej sytuacji jest wymuszenie spowalniania tempa wysyłania ramek przez nadawcę — ogólnie mechanizm ten nosi nazwę *sterowania przepływem* (*flow control*) i w Ethernecie implementowany jest przez większość przełączników oraz kart w ten sposób, że do nadawcy wysyłane są specjalne ramki kontrolne, zwane **ramkami PAUSE** (*PAUSE frames*), opisane w specyfikacji 802.3 (patrz [802.3-2008]).

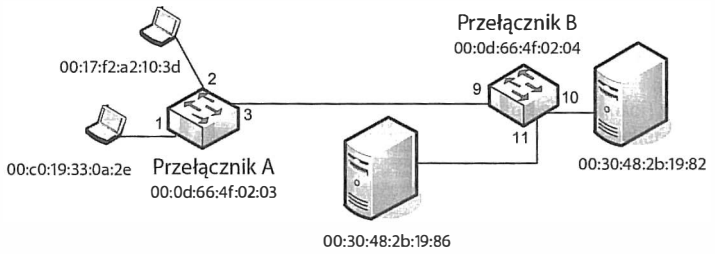
Ramki *PAUSE* identyfikowane są przez wartość 0x8808 w polu *Rozmiar/Typ*; w następujących bezpośrednio dalej dwóch bajtach znajduje się kod kontrolny 0x001, zaś dwa następne bajty zawierają wartość żądanego czasu powstrzymania transmisji, wyrażonego w jednostkach zwanych *Pause Quanta* — jednostka taka odpowiada czasowi przesłania 512 bitów przez łącze. W polu adresu docelowego (DST) znajduje się ustalona wartość 01:80:C2:00:00:01.

Ramki kontrolne (*control frames*) wpisują się w ogólny format ramki ethernetowej, przedstawiony na rysunku 3.3. Znajdująca się w polu *Rozmiar/Typ* wartość 0x8808 oznacza, że w dwóch następnych bajtach znajduje się kod kontrolny narzucający sposób interpretowania kolejnych bajtów.

Wykorzystywanie sterowania przepływem w warstwie Ethernetu prowadzi do pewnego niepożądanego efektu ubocznego (i z tego powodu nie jest powszechnie stosowane). Gdy transmisja z wielu stacji jednocześnie doprowadza do przeciążenia pamięci przełącznika, ten zaczyna wysyłać ramki *PAUSE* do *wszystkich* nadawców — także do tych, które w spowodowaniu rzeczonego przeciążenia mają udział znikomy lub żaden i które — tym samym — są niesłusznie penalizowane.

3.4. Mostki a przełączniki

Standard IEEE 802.1d wprowadza koncepcję **mostka sieciowego** (*bridge*) jako urządzenia koordynującego przepływ pakietów między dwom segmentami sieci na poziomie łącza danych — czyli np. między parą fizycznych segmentów ethernetowych, tak jak na rysunku 3.8. Widoczne na tym rysunku przełączniki A i B pełnią właśnie rolę mostków.



Rysunek 3.8. *Proste rozszerzenie sieci LAN za pomocą dwóch przełączników. Każdy port przełącznika posiada numer referencyjny, każde urządzenie posiada unikatowy adres MAC*

W konfiguracji z rysunku 3.8 przełącznik A, do którego przyłączone są komputery „segmentu klienckiego", połączony jest z przełącznikiem B, łączącym komputery „segmentu serwerowego"; każde urządzenie w tej konfiguracji posiada unikatowy adres MAC, każdy port identyfikowany jest (na rysunku) unikatowym numerem referencyjnym. Pełnienie funkcji mostka przez przełącznik polega m.in. na tym, że ten „uczy się" z upływem czasu sposobów osiągania urządzeń „nielokalnych", czyli przyłączonych do innych przełączników, a zdobywana w ten sposób wiedza gromadzona jest w tzw. bazie filtracyjnej (*filtering databases*). Dokładniej mówiąc, każdy rekord tej bazy jest parą „adres MAC-port" co oznacza, że ruch kierowany do określonego adresu MAC powinien wychodzić z przełącznika przez określony port. Wspomnianą „naukę" realizują przełączniki na podstawie obserwacji ramek krążących w medium transmisyjnym, z których to ramek odczytują adresy źródłowe MAC. Gdy przełączniki A i B z rysunku 3.8 zakończą swą edukację, informacja zawarta w ich bazach filtracyjnych prezentować się będzie tak, jak na rysunku 3.9.

Rysunek 3.9. *Bazy filtracyjne przełączników A i B z rysunku 3.8 po zakończeniu skanowania topologii sieci*

Stacja	Port
00:17:f2:a2:10:3d	2
00:c0:19:33:0a:2e	1
00:0d:66:4f:02:03	
00:0d:66:4f:02:04	3
00:30:48:2b:19:82	3
00:30:48:2b:19:86	3

Baza filtracyjna przełącznika A

Stacja	Port
00:17:f2:a2:10:3d	9
00:c0:19:33:0a:2e	9
00:0d:66:4f:02:03	9
00:0d:66:4f:02:04	
00:30:48:2b:19:82	10
00:30:48:2b:19:86	11

Baza filtracyjna przełącznika B

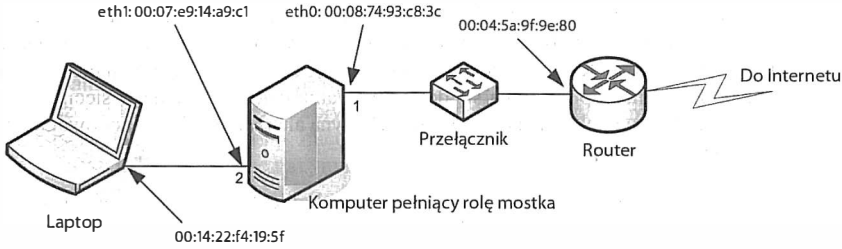
Bezpośrednio po włączeniu przełącznika jego baza filtracyjna jest pusta — przełącznik dysponuje informacjami o lokalizacjach jedynie swych własnych stacji. Gdy do przełącznika tego dotrze przez określony port ramka adresowana do „obcej" stacji, przełącznik

wyśle kopię tej ramki do wszystkich innych portów. Jeżeli jednak zawarty w ramce adres docelowy figurować będzie w bazie filtracyjnej, przełącznik wyśle kopię ramki tylko przez jeden port — ten odczytany z rekordu bazy. Jak więc widać, istnienie baz filtracyjnych znakomicie przyczynia się do redukowania natężenia ruchu w sieci: wysyłanie ramki przez jeden konkretny port staje się efektywną alternatywą dla „floodowania" przez wszystkie możliwe porty.

Obecnie większość systemów operacyjnych realizuje funkcję połączeń mostkowych między interfejsami sieciowymi, dzięki czemu dowolny komputer wyposażony w kilka kart sieciowych może realizować funkcje mostka. W Windows 7 w celu skonfigurowania połączenia mostkowego należy otworzyć *Centrum sieci i udostępniania*, przejść na stronę *Zmień ustawienia karty sieciowej*, zaznaczyć (trzymając naciśnięty klawisz *Ctrl*) dwa interfejsy przeznaczone do mostkowania i z menu kontekstowego dowolnego z nich wybrać opcję *Połączenia mostkowe*. Spowoduje to pojawienie się nowej ikony, reprezentującej urządzenie mostkujące, które przejmuje od swych źródłowych interfejsów większość typowych funkcji sieciowych, m.in. klienta sieci Microsoft Windows oraz udostępnianie plików i drukarek (patrz rysunek 3.10).

Rysunek 3.10. *Okno właściwości mostka sieciowego w Windows 7. Mostek tworzy się, zaznaczając ikony reprezentujące interfejsy źródłowe i wybierając z menu kontekstowego opcję Połączenia mostkowe. Utworzony mostek przejmuje z interfejsów źródłowych większość właściwości sieciowych, które tym samym stają się niedostępne na poziomie oryginalnych interfejsów*

W systemie Linux konfigurowanie mostka odbywa się przy użyciu zestawu poleceń, co zilustrujemy na przykładzie topologii przedstawionej na rysunku 3.11. Rolę mostka pełni komputer PC wyposażony w dwie karty sieciowe: do karty numer 2 podłączona jest pojedyncza stacja (laptop), przez port numer 1 osiągalna jest „reszta" sieci.



Rysunek 3.11. *Prosta topologia, w ramach której komputer linuksowy skonfigurowany jest jako mostek łączący dwa segmenty ethernetowe. Jako mostek uczący się, gromadzi on informację na temat osiągalności różnych systemów zewnętrznych za pośrednictwem poszczególnych portów*

W celu zdefiniowania mostka należy zrealizować następujący ciąg poleceń:

```
Linux# brctl addbr br0
Linux# brctl addif br0 eth0
Linux# brctl addif br0 eth1
Linux# ifconfig eth0 up
Linux# ifconfig eth1 up
Linux# ifconfig br0 up
```

Po utworzeniu urządzenia mostkującego `br0` kojarzone są z nim interfejsy `eth0` i `eth1`. Interfejs źródłowy można odłączyć od mostka za pomocą polecenia `brctl delif`. Listę funkcjonujących mostków można otrzymać za pomocą polecenia `brctl show`:

```
Linux# brctl show
bridge name bridge id STP enabled interfaces
br0 8000.0007e914a9c1 no eth0 eth1
```

Zawartość bazy filtracyjnej konkretnego mostka (zwanej w terminologii linuksowej **bazą forwardingową** — *forwarding database*, w skrócie *fdb*) można obejrzeć za pomocą polecenia `brctl showmacs`:

```
Linux# brctl showmacs br0
port no mac addr is local? ageing timer
  1 00:04:5a:9f:9e:80 no 0.79
  2 00:07:e9:14:a9:c1 yes 0.00
  1 00:08:74:93:c8:3c yes 0.00
  2 00:14:22:f4:19:5f no 0.81
  1 00:17:f2:e7:6d:91 no 2.53
  1 00:90:f8:00:90:b7 no 17.13
```

Powyższy raport ujawnia przy okazji pewien interesujący szczegół funkcjonowania mostka. Ponieważ struktura sieci może się zmieniać — stacje są dołączane i odłączane, wymieniane są karty sieciowe, zmieniają się adresy MAC itp. — zawartość bazy filtracyjnej może stać się w każdej chwili nieaktualna, musi więc podlegać okresowej weryfikacji. Problem ten rozwiązano, związując z każdym rekordem bazy tzw. *okres przedawnienia* o konfigurowalnej wartości (zazwyczaj domyślnie 5 minut): jeżeli w trakcie tego okresu nie nadejdzie do mostka ramka spod adresu specyfikowanego w rekordzie, rekord jest usuwany z bazy. W Linuksie wspomniany okres przedawnienia definiowany jest globalnie dla całego mostka za pomocą polecenia `brctl setageing`; w poniższym przykładzie, dla celów ilustracyjnych, specjalnie przyjęliśmy nienaturalnie małą wartość 1 sekundy.

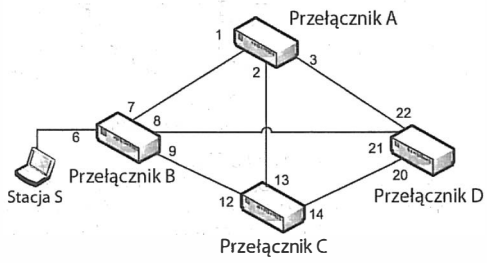
W kolumnie ageing timer widoczny jest interwał, jaki upłynął od czasu utworzenia rekordu lub ostatniego zarejestrowania ramki przybyłej spod adresu specyfikowanego w rekordzie.

```
Linux# brctl setageing br0 1
Linux# brctl showmacs br0
port no mac addr is local? ageing timer
  1 00:04:5a:9f:9e:80 no 0.76
  2 00:07:e9:14:a9:c1 yes 0.00
  1 00:08:74:93:c8:3c yes 0.00
  2 00:14:22:f4:19:5f no 0.78
  1 00:17:f2:e7:6d:91 no 0.00
```

Oczywiście, opisana strategia prowadzić może do pochopnego usuwania rekordów z bazy filtracyjnej — adres MAC może pozostawać aktualny, mimo iż nie nadeszła z jego strony żadna ramka. To jednak żadna katastrofa, bo ponowne pojawienie się ramki ze wspomnianym adresem znowu spowoduje jego umieszczenie w bazie. Znacznie poważniejszym problemem, występującym w sytuacji połączenia wielu mostków redundantnymi łączami, jest możliwość zapętlenia ramek, prowadząca do katastrofalnego przeciążenia sieci. Przyjrzyjmy się dokładniej temu zagrożeniu i zobaczmy, jak można mu przeciwdziałać.

3.4.1. Protokół drzewa rozpinającego (STP)

Mostki mogą działać samodzielnie lub w połączeniu z innymi mostkami. Gdy połączonych jest kilka (więcej niż dwa) mostków — i generalnie, gdy porty przełącznika są skrosowane — istnieje niebezpieczeństwo zapętlenia dystrybucji ramek i lawinowego ich namnażania. Zilustrujemy to zjawisko na przykładzie sieci pokazanej na rysunku 3.12.

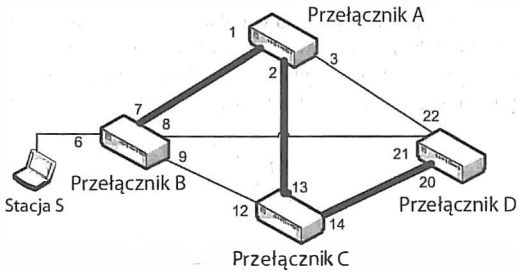


Rysunek 3.12. *Rozbudowana sieć ethernetowa z czterema przełącznikami i wieloma łączami redundantnymi. Wynikająca z tej redundancji możliwość zapętlenia ramek może być przyczyną lawinowego narastania obciążenia sieci. Antidotum na to zagrożenie jest protokół STP*

Załóżmy, że wszystkie cztery przełączniki (A, B, C i D) zostały dopiero co włączone i ich bazy filtracyjne nie zawierają żadnych rekordów. Gdy stacja S wyśle ramkę, w przełączniku B ramka ta zostanie powielona trzykrotnie, po jednym egzemplarzu dla każdego z portów 7, 8 i 9. Każda z tych trzech kopii zostanie następnie powielona dwukrotnie w następnym przełączniku A, C albo D. Przełączniki A, C i D, nawzajem przerzucając między sobą sześć kopii pierwotnej ramki, nieustannie modyfikować będą swe bazy filtracyjne z zamiarem (niestety, bezskutecznym) ustalenia, który z portów jest naprawdę tym prowadzącym do stacji S. Zdecydowanie nie tego oczekiwaliśmy po funkcji mostkowania i podjąć należy działania zmierzające do zniwelowania opisanego zjawiska.

Po bliższej analizie tego zjawiska nietrudno wywnioskować, że przyczyną całego zamieszania jest *cykliczny* charakter połączeń między przełącznikami. Przykładowo ramka, która opuściła przełącznik A przez port 3, po przejściu przez przełączniki C i D trafia z powrotem do przełącznika A przez port 2. Usunięcie cykli z topologii sieci jest więc naturalnym sposobem na przerwanie błędnego koła; wykonanie tego zadania powierzono **protokołowi drzewa rozpinającego** (STP — *Spanning Tree Protocol*), którego podstawy działania i realizację teraz opiszemy. Oryginalna wersja tego protokołu zastąpiona została w standardzie [802.1D-2004] przez jego udoskonaloną wersję, czyli **błyskawiczny protokół drzewa rozpinającego** (RSTP — *Rapid Spanning Tree Protocol*), którym zajmiemy się później.

W teorii grafów **grafem spójnym** nazywamy graf, w którym między dowolną parą wierzchołków istnieje przynajmniej jedna ścieżka. Graf, w którym nie występują cykle, nazywamy **drzewem**. Biorąc dowolny graf spójny i pozbawiając go cykli przez usuwanie krawędzi, zachowując jednak spójność, redukujemy go do struktury zwanej **drzewem rozpinającym**. W strukturze tej między *dowolną parą* wierzchołków istnieje *dokładnie jedna* droga. Wyróżniony wierzchołek drzewa rozpinającego nazywamy **korzeniem**. Zazwyczaj dla danego grafu można utworzyć wiele drzew rozpinających, a pełnienie roli korzenia można powierzyć dowolnemu wierzchołkowi. Wynik redukcji grafu połączeń z rysunku 3.12 do drzewa rozpinającego widzimy na rysunku 3.13 — krawędzie drzewa oznaczono pogrubionymi liniami.



Rysunek 3.13. *W rezultacie zastosowania protokołu STP aktywne pozostają tylko łącza B–A, A–C i C–D tworzące drzewo rozpinające. Porty 6, 7, 1, 2, 13, 14 i 20 normalnie forwardują ramki, w pozostałych portach funkcja forwardowania jest zablokowana. Eliminuje to możliwość cyklicznej transmisji ramek. W przypadku awarii któregoś z portów odblokowane zostają wszystkie porty i protokół STP konstruuje nowe drzewo rozpinające*

Protokół STP realizuje ideę drzewa rozpinającego poprzez *blokowanie* wybranych portów — w tym przypadku portów 8, 9, 12, 21, 22 i 3, jedynie porty 7, 1, 2, 13, 14 i 21 uczestniczą w przesyłaniu ramek między przełącznikami. Między dowolną parą przełączników istnieje teraz dokładnie jedna ścieżka dla ramek. Wyznaczanie drzewa rozpinającego i w konsekwencji blokowanie odpowiednich portów realizowane jest przez rozproszony algorytm, którego instancje implementowane są we wszystkich mostkach.

Podobnie jak w przypadku baz filtracyjnych, również konstruowane przez protokół STP drzewa rozpinające mogą stawać się nieadekwatne do bieżącej sytuacji z tych samych powodów: wyłączenia mostka, wymiany karty sieciowej, zmiany adresu MAC itp. W związku z tym, protokół ten implementuje odpowiednie mechanizmy adaptacyjne, bazujące na wymianie informacji między poszczególnymi instancjami algorytmu; wymiana ta or-

ganizowana jest w postaci specjalnych ramek zwanych „jednostkami danych protokołu mostków" — *bridge protocol data units*, w skrócie BPDU — a budowanie drzewa rozpinającego rozpoczyna się od jego korzenia, którym staje się mostek wybierany w drodze specyficznego „głosowania".

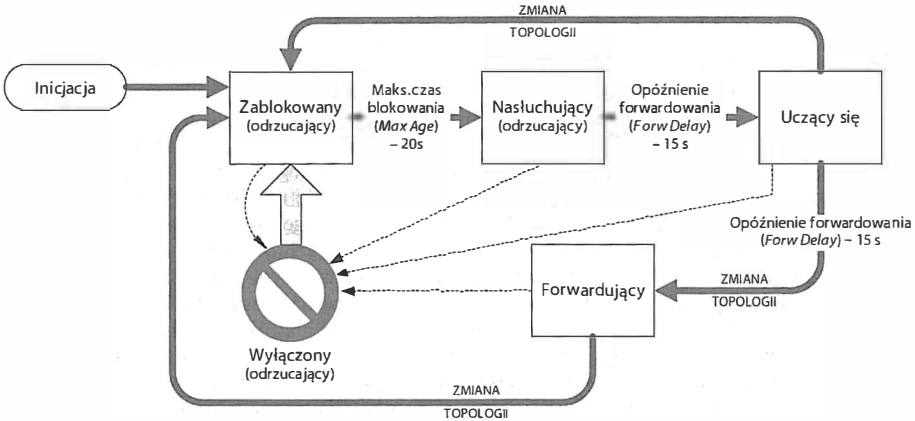
Jak wspominaliśmy, dla danego grafu może istnieć wiele drzew rozpinających. Jeżeli każdej krawędzi grafu przypiszemy umownie liczbę (w ogólności — liczbę rzeczywistą) rozumianą jako *koszt* tej krawędzi i jeżeli za koszt drogi między wierzchołkami uważać będziemy sumę kosztów krawędzi tworzących tę drogę, a za koszt drzewa — sumę kosztów jego krawędzi, to spośród wszystkich możliwych drzew rozpinających rzeczony graf można wybrać to, które charakteryzuje się najmniejszym kosztem. W takim drzewie sumaryczny koszt wszystkich dróg łączących korzeń z pozostałymi wierzchołkami jest mniejszy niż w innych drzewach rozpinających ten sam graf. W przypadku grafu połączeń, którego krawędzie odpowiadają łączom między przełącznikami, rozsądnie jest zdefiniować koszt każdej krawędzi jako liczbę całkowitą, w przybliżeniu odwrotnie proporcjonalną do przepustowości łącza reprezentowanego przez tę krawędź. I tak np. łączom o przepustowości (odpowiednio) 10 Mb/s, 100 Mb/s i 1000 Mb/s można przyporządkować koszty (odpowiednio) 100, 19 i 4. Oczywiście, z punktu widzenia transmisji ramek w sieci optymalne jest wówczas drzewo rozpinające o najmniejszym koszcie — konstruowanie takich właśnie drzew jest istotą protokołu STP. W takim optymalnym drzewie ramki docierają do korzenia (z poszczególnych wierzchołków) tak szybko, jak jest to możliwe.

3.4.1.1. Stany i role portów

Centralną częścią protokołu STP są automaty skończone (zwane również maszynami stanów) odzwierciedlające stany (i przejścia między nimi) wszystkich portów we wszystkich mostkach. W danej chwili konkretny port znajdować się może w jednym z pięciu stanów: **zablokowany** (*blocking*), **nasłuchujący** (*listening*), **uczący się** (*learning*), **forwardujący** (*forwarding*) albo **wyłączony** (*disabled*). Możliwe przejścia między stanami definiuje diagram widoczny na rysunku 3.14; liniami ciągłymi oznaczono przejścia wynikające z naturalnego funkcjonowania protokołu STP, linie przerywane reprezentują przejścia wymuszane w rezultacie interwencji administratora sieci.

Bezpośrednio po zainicjowaniu port znajduje się w stanie „zablokowany". W tym stanie nie uczestniczy w procesie „uczenia się" mostka (czyli uaktualniania jego bazy filtracyjnej), nie realizuje forwardowania ramek i nie wysyła ramek BPDU, monitoruje jednak otrzymywane ramki BPDU, na wypadek gdyby w przyszłości miał zostać portem prowadzącym po najkrótszej ścieżce do korzenia drzewa rozpinającego. W stanie „nasłuchujący" port może odbierać i wysyłać ramki BPDU, w dalszym ciągu jednak nie uczestniczy w procesie uczenia się mostka ani nie realizuje forwardowania ramek. Po upływie określonego interwału czasowego (zazwyczaj 15 sekund) port przechodzi do stanu „uczący się", co oznacza uruchomienie funkcji uczenia; po upływie kolejnego interwału czasowego port wchodzi w stan „forwardujący", oznaczający pełną funkcjonalność (czyli również forwardowanie ramek).

Portowi, stosownie do maszyny stanów, przypisuje się określone **role** — szczególnie istotne w przypadku protokołu RSTP (patrz podpunkt 3.4.1.6): dany port może być portem **głównym** (*root*), **wyznaczonym** (*designated*), **alternatywnym** (*alternate*) lub **zapasowym** (*backup*). Portami głównymi są te porty na końcach krawędzi drzewa rozpinającego,



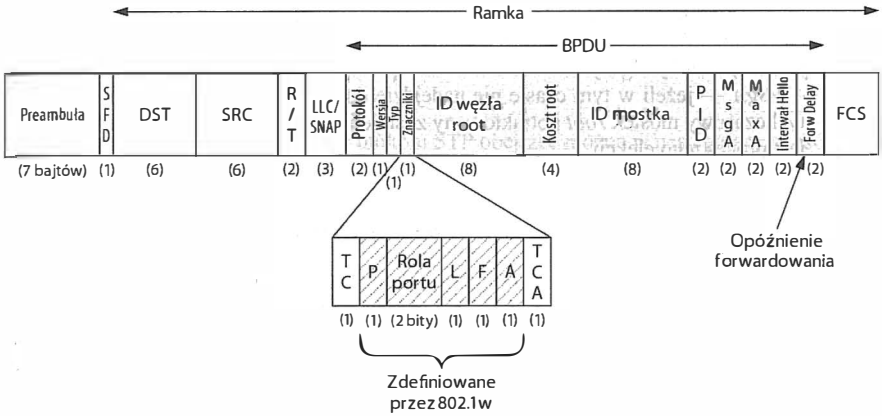
Rysunek 3.14. *Maszyna stanów portu w kontekście protokołu STP. Port zablokowany nie forwarduje ramek, jednak zmiana topologii może spowodować jego przejście w stan nasłuchiwania. Port forwardujący to port wykonujący swą zasadniczą pracę, czyli forwardowanie ramek. Nazwy w nawiasach oznaczają stan portu w terminologii RSTP*

które prowadzą do korzenia tego drzewa po ścieżce „najkrótszej", czyli o najmniejszym koszcie. Porty wyznaczone to porty w stanie „forwardujący" będące portami na ścieżce o najmniejszym koszcie prowadzącej w kierunku korzenia drzewa rozpinającego. Portami alternatywnymi są porty także prowadzące do korzenia drzewa rozpinającego, lecz po ścieżce o koszcie większym niż minimalny; porty te nie forwardują ramek. Portem zapasowym jest port dołączony do tego samego segmentu, co port wyznaczony *w tym samym mostku*, może go więc zastąpić w przypadku jego awarii, lecz nie zapewnia alternatywnej ścieżki w czasie awarii całego mostka.

3.4.1.2. Struktura BPDU

Na rysunku 3.15 przedstawiono format ramek BPDU, wymienianych między mostkami w ramach budowania drzewa rozpinającego przez protokół STP; jest to format wspólny dla obu wersji protokołów: klasycznej (STP) i ulepszonej (RSTP). Ramki BPDU wysyłane są na adres grupowy `01:80:C2:00:00:00` (patrz rozdział 9.), a przejście ramki przez mostek zawsze wiąże się z jej modyfikacją. Znaczenie pól `DST`, `SRC` i `R/T` jest takie samo jak w klasycznym formacie ramki ethernetowej 802.3 (patrz rysunek 3.3) — pole `R/T` zawiera wartość 38 (dziesiętnie), czyli rozmiar ładunku użytecznego (rozpoczynającego się od pola `LLC/SNAP`). W 3-bajtowym polu `LLC/SNAP` znajduje się stała `0x424203`, zdefiniowana dla ramek BPDU przez standard 802.1 (niektóre ramki BPDU nie wykorzystują tego pola).

Pole *Protokół* ma wartość 0, pole *Wersja* zawiera 0 albo 2, zależnie od wersji protokołu (odpowiednio STP albo RSTP), na podobnej zasadzie ustalana jest wartość pola *Typ*. W polu znaczników obecne są bity związane ze zmianą topologii sieci (zgodnie ze standardem 802.1d: `TC` — *Topology Change* i `TCA` — *Topology Change Acknowledgment*), propozycją (`P` — *Proposal*), rolą portu (`00` — nieznana, `01` — alternatywny, `10` — główny, `11` — wyznaczony), uczeniem mostka (`L` — *Learning*), forwardowaniem ramek (`F` — *Forwarding*) i negocjowaniem (`A` — *Agreement*); ich znaczenie opisujemy w podpunkcie 3.4.1.6



Rysunek 3.15. *BPDU jako ładunek użyteczny przenoszony w ramkach 802, wymienianych między mostkami w procesie budowania drzewa rozpinającego. Najważniejsze pola BPDU określają adres źródłowy, węzeł root, koszt ścieżki prowadzącej do węzła root i wskaźnik zmiany topologii. Specyfikacje 802.1w i [802.1D-2004] definiują dodatkowe pola określające m.in. stan portu*

w kontekście protokołu RSTP. *ID węzła root* to identyfikator domniemywany przez nadawcę ramki, zaś analogiczny identyfikator nadawcy znajduje się w polu *ID mostka root*; oba identyfikatory stanowią konkatenację 2-bajtowego priorytetu i adresu MAC. Ponieważ, jak wkrótce zobaczymy, identyfikator mostka ma kluczowe znaczenie w procesie wyboru mostka *root*, manipulowanie wspomnianym priorytetem (stanowiącym bardziej znaczącą część identyfikatora) jest podstawowym sposobem preferowania konkretnych mostków w tej roli przez oprogramowanie zarządzające; przykładowo w przełącznikach Catalyst firmy Cisco priorytet ustawiany jest domyślnie na wartość 0x8000.

Wartość w polu *Koszt root* wyliczana jest jako koszt drogi prowadzącej od mostka nadawcy do mostka *root* (koszt rozumiany jako suma kosztu łączy leżących na tej drodze). W polu *PID* znajduje się identyfikator (numer) portu, przez który mostek nadawca wysłał ramkę, skonkatenowany z 1-bajtowym polem priorytetu (o domyślnej wartości 0x80).

Następne pola związane są z czasowymi uwarunkowaniami transmisji ramek. Pole *MsgA* zawiera wartość odzwierciedlającą upływ „czasu życia" ramki (o tym dokładniej za chwilę), zaś w polu *MaxA* definiowany jest interwał przeterminowania (domyślnie 20 sekund). Wartość pola *Interwał Hello* określa interwał czasowy ponawiania okresowej transmisji ramek konfiguracyjnych, zaś wartość pola *Opóźnienie forwardowania* (*Forw Delay*) określa długość pozostawania portu w stanie „uczący się" lub „nasłuchujący". Wszystkie wartości czasowe wyrażane są w jednostkach równych $\frac{1}{256}$ sekundy.

Pole *MsgA*, w odróżnieniu od innych pól czasowych, nie ma ustalonej wartości. W ramce BPDU wysyłanej przez mostek *root* pole ma to wartość 0; przy przejściu ramki przez każdy kolejny mostek wartość tego pola jest zwiększana o 1 (z wyjątkiem przypadku, gdy jej wysłanie następuje przez port uważany za prowadzący do mostka *root*). W rzeczywistości zatem pole to jest licznikiem przeskoków, czyli mostków pośrednich, przez

które przeszła dotychczas ramka od momentu wygenerowania jej przez mostek *root*. Gdy ramka BPDU odbierana jest przez mostek, zawarta w niej informacja gromadzona jest na potrzeby algorytmu STP i usuwana po okresie przeterminowania równym *MaxA – MsgA* — jeżeli w tym czasie nie nadejdzie do mostka ramka przez port główny, dotychczasowy mostek *root* potraktowany zostaje jako „martwy" i ponownie uruchamiany jest proces jego elekcji.

3.4.1.3. Budowanie drzewa rozpinającego

Pierwszym krokiem w procesie budowania drzewa rozpinającego przez protokół STP jest ustanowienie mostka *root*. W tej roli obsadzany jest ten z (funkcjonujących) mostków, któremu przypisano najmniejszy identyfikator (czyli konkatenację priorytetu i adresu MAC). Początkowo (po włączeniu) każdy mostek domniemywa, iż to on jest mostkiem *root* i rozsyła ramki BPDU z własnym identyfikatorem w polu *ID węzła root*. Tak jednak dzieje się tylko do momentu, gdy mostek ten odbierze ramkę BPDU z mniejszą wartością identyfikatora węzła root; port, przez który ramka ta zostanie odebrana, zostaje wówczas oznaczony przez mostek jako port główny (czyli port prowadzący do poprzedniego mostka leżącego na drodze do mostka *root*), sama zaś ramka rozsyłana jest w miejsce poprzedniej. Gdy nadejdzie ramka z jeszcze mniejszą wartością identyfikatora węzła *root*, sytuacja się powtarza.

3.4.1.4. Zmiany topologii

Kolejnym wyzwaniem, jakiemu stawić musi czoło protokół STP, są możliwe zmiany w topologii sieci. Można by w tym względzie zdać się na opisany wcześniej mechanizm samoczynnego uaktualniania baz filtracyjnych, gdyby nie zbyt mała częstotliwość tego uaktualniania — standardowo równa 5 minut. W STP zastosowano więc oddzielny, znacznie szybszy mechanizm, polegający na wykrywaniu zmian topologii w momencie przechodzenia portu w stan „zablokowany" lub „forwardujący". Gdy mianowicie mostek wykryje zmiany w funkcjonowaniu łącza (np. jego uszkodzenie), powiadamia o tym mostek macierzysty (czyli poprzedni na trasie łączącej go z mostkiem *root*), wysyłając ramkę BPDU zawierającą *powiadomienie o zmianie topologii* (TCN — *Topology Change Notification*). Mostek odbierający tę ramkę propaguje ją do swego mostka macierzystego, jednocześnie wysyłając do nadawcy komunikat potwierdzenia informacji o zmianie topologii (TA — *Topology Change Acknowledgment*). Ostatecznie komunikat TCN dociera do mostka *root*, który w związku z zaistniałą sytuacją zaczyna rozsyłanie po sieci ramek z ustawionym znacznikiem TC. Reakcją każdego mostka na odebranie takiego komunikatu jest natychmiastowe usunięcie nieaktualnych rekordów z baz filtracyjnych, już po kilku sekundach od wystąpienia zmian w topologii; warto również zauważyć, że stacje pozostające bez związku ze zmianą topologii nie muszą niepotrzebnie usuwać zawartości swych baz filtracyjnych.

3.4.1.5. Przykład

W systemie Linux protokół STP jest domyślnie wyłączony, zakłada się bowiem prostą topologię sieci, z opcjonalnym wykorzystywaniem zwykłego komputera jako mostka.

Poniższe polecenie powoduje włączenie protokołu STP w mostku br0, który utworzyliśmy w jednym z wcześniejszych przykładów:

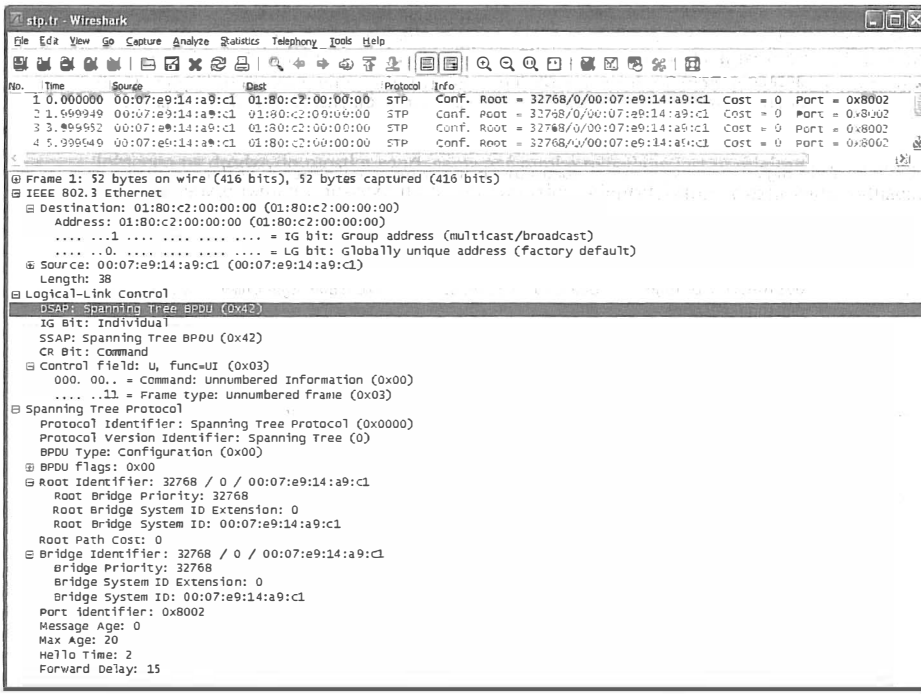
```
Linux# brctl stp br0 on
```

Szczegóły funkcjonowania protokołu STP obejrzeć można w następujący sposób:

```
Linux# brctl showstp br0

br0

 bridge id              8000.0007e914a9c1
 designated root        8000.0007e914a9c1
 root port                 0                path cost              0
 max age                  19.99             bridge max age        19.99
 hello time                1.99             bridge hello time      1.99
 forward delay            14.99             bridge forward delay  14.99
 ageing time               0.99
 hello timer               1.26             tcn timer              0.00
 topology change timer     3.37             gc timer               3.26

 flags                   TOPOLOGY_CHANGE TOPOLOGY_CHANGE_DETECTED

eth0 (0)
 port id                0000                    state         forwarding
 designated root        8000.0007e914a9c1       path cost          100
 designated bridge      8000.0007e914a9c1       message age timer   0.00
 designated port        8001                    forward delay timer 0.00

 designated cost        0                       hold timer          0.26

 flags

eth1 (0)
 port id                0000                    state         forwarding
 designated root        8000.0007e914a9c1       path cost           19
 designated bridge      8000.0007e914a9c1       message age timer   0.00
 designated port        8002                    forward delay timer 0.00

 designated cost        0                       hold timer          0.26

 flags
```

Powyższy raport rozpoczyna się od właściwości mostka br0 jako całości: widzimy identyfikator mostka (8000.0007e914a9c1) utworzony na bazie najmniejszego adresu MAC interfejsu składowego eth1 (patrz rysunek 3.11), widzimy też wyrażone w sekundach parametry czasowe (*Interwal Hello*, *Opóźnienie forwardowania*, okres przedawnienia itp.). Znaczniki (*flags*) wskazują na niedawną zmianę topologii, co nie powinno dziwić wobec faktu, że urządzenia sieci zostały dopiero przed chwilą włączone. Pozostałe części raportu dotyczą interfejsów składowych eth0 (port 1) i eth1 (port 2). Zauważmy, że koszt ścieżki (*path cost*) jest dla eth0 ok. 10 razy większy niż dla eth1; jest to prostą konsekwencją faktu, że eth0 to interfejs 10 Mb/s, zaś eth1 to interfejs 100 Mb/s pracujący w trybie pełnodupleksowym.

Na rysunku 3.16 widoczne jest okno programu Wireshark ukazujące zawartość ramki BPDU. Raportowana długość ramki (52 bajty, a więc mniej niż wymagane minimum 64 bajty) nie uwzględnia dopełnienia, a jedynie 14-bajtowy nagłówek i 38-bajtowy ładunek użyteczny. Adres docelowy jest adresem grupowym `01:80:C2:00:00:00`, co wcześniej wyjaśnialiśmy. Rozmiar ładunku użytecznego wynosi 38 bajtów, zgodnie z polem `R/T`, w polu `SNAP/LLC` znajduje się predefiniowana stała `0x424243`, a hermetyzowana ramka związana jest z klasyczną wersją protokołu STP (wartość `0` w polu *Wersja*). Z reszty pól protokołu możemy wyczytać, że aktualnie domniemanym mostkiem *root* jest stacja o adresie MAC `00:07:e9:14:a9:c1`, z priorytetem `32768` (niskim), ramka zaś wysłana została przez port 2 z wartością priorytetu `0x80`. Widoczne są także parametry czasowe: *MaxA* = 20 sekund, *Interwał Hello* = 2 sekundy i *Opóźnienie forwardowania* = 15 s.



Rysunek 3.16. *Ramka BPDU w oknie programu Wireshark. Adres docelowy jest adresem grupowym dla mostków (01:80:c2:00:00:00)*

3.4.1.6. Błyskawiczny protokół drzewa rozpinającego (RSTP), dawniej 802.1w

Jednym z mankamentów konwencjonalnego protokołu STP jest stosunkowo duża inercja reagowania na czynniki powodujące zmianę topologii sieci: czas zbieżności, czyli przystosowania baz filtracyjnych do nowych warunków, jest dość długi — w wielu przypadkach zbyt długi. Okoliczność ta stała się fundamentem nowej, ulepszonej wersji protokołu, opisanej w specyfikacji IEEE 802.1w (stanowiącej obecnie część specyfikacji

[802.1D-2004]) i opatrzonej nową nazwą *Rapid Spanning Tree Protocol* („błyskawiczny protokół drzewa rozpinającego"), w skrócie RSTP. Podstawowym ulepszeniem RSTP w stosunku do STP jest monitorowanie statusu każdego portu i w przypadku stwierdzenia przesłanek awarii *natychmiastowe* wyzwalanie procedury adaptującej do nowej topologii. RSTP wykorzystuje ponadto wszystkie 6 bitów znaczników w ramce BPDU w celu komunikowania dodatkowej informacji między mostkami, co w wielu przypadkach pozwala na uniknięcie niepotrzebnego inicjowania pewnych operacji przez procedury zegarowe protokołu. Zredukowana została także maszyna stanów portu: zamiast pięciu stanów mamy teraz trzy (ich nazwy ujęte są w nawiasy na rysunku 3.14) — „odrzucający", „uczący się" i „forwardujący"; nowy stan „odrzucający" jest wynikiem scalenia dotychczasowych stanów „wyłączony", „zablokowany" i „nasłuchujący". RSTP definiuje także nową rolę portu — *alternatywny* — polegającą na natychmiastowym zastępowaniu portu głównego w przypadku jego awarii.

RSTP wykorzystuje tylko jeden typ ramki BPDU, nie ma więc np. specjalnej ramki informującej o zmianie topologii. Ramki protokołu RSTP mają wartość 2 w polu *Wersja* (ramki klasycznego protokołu STP mają w tym polu wartość 0). Każdy przełącznik, który wykryje warunki mogące skutkować zmianą topologii, może natychmiast wysłać odpowiednią ramkę informującą o tym fakcie, zaś każdy przełącznik, który taką ramkę odbierze, powinien obowiązkowo usunąć wszystkie rekordy ze swej bazy filtracyjnej. Takie postawienie sprawy znacząco skraca czas konwergencji: zamiast dotychczasowej wędrówki ramki TCN w górę drzewa (czyli do mostka *root*) i następnie rozgłaszania ramki TC z opóźnieniem o interwał *Opóźnienie forwardowania* w każdym węźle mamy teraz natychmiastowy broadcasting z poziomu dowolnego mostka. Efektem netto jest skrócenie czasu konwergencji z (nawet) kilkudziesięciu sekund do (w większości przypadków) ułamka sekundy.

Protokół RSTP rozróżnia **porty brzegowe** (*edge ports*) — czyli porty bezpośrednio podłączone do stacji końcowych — i „zwykłe" porty, uczestniczące w algorytmie STP; odróżnia także połączenia „punkt-punkt" od połączeń współdzielonych (*shared links*). Porty brzegowe i łącza „punkt-punkt" zazwyczaj nie tworzą cykli w grafie połączeń, dzięki czemu pominąć można stany „nasłuchujący" oraz „uczący się" i przejść od razu do stanu „forwardujący". Gdy zdarzy się tak, że na porcie uznanym aktualnie za brzegowy pojawi się ramka BPDU, port ten natychmiast zmienia status na „zwykły" — nie może on być przyłączony do stacji końcowej, ponieważ stacje końcowe nie wysyłają ramek BPDU; stając się portem zwykłym, rozpoczyna jednocześnie swe uczestnictwo w budowaniu drzewa rozpinającego. Typ łącza jest określany na podstawie trybu operacyjnego interfejsu: dla trybu „pełny dupleks" jest to łącze „punkt-punkt", dla trybu „półdupleks" — łącze współdzielone.

W klasycznym STP ramki BPDU generowane są bądź to przez mostki powiadamiające o awarii, bądź też przez mostek *root* sygnalizujący zmianę topologii. RSTP usprawnia przepływ informacji w tym względzie, tworząc nowy mechanizm diagnozujący „żywotność" sieci, notabene wzorowany na analogicznym zachowaniu wielu protokołów warstw wyższych: każdy z mostków wymienia okresowo określone ramki BPDU (zwane w oryginale *keepalives*) ze swymi sąsiadami, testuje tym samym poprawność połączenia z nimi. Odstęp czasowy między emisją kolejnych ramek określony jest w polu *Interwał Hello*. Jeżeli któryś z mostków nie otrzyma żadnej ramki *keepalive* od któregoś z sąsiadów na przestrzeni trzykrotności tego interwału, uznaje połączenie z tym sąsiadem za zerwane

i rozpoczyna wysyłanie ramek TC nie tylko do mostka *root*, lecz także bezpośrednio do innych mostków; dzięki temu procedury adaptacyjne mogą zostać zapoczątkowane znacznie szybciej niż w klasycznym STP — powiadomione mostki usuwają zawartość swych baz filtracyjnych i przystępują do ponownego „uczenia się" topologii. Z mechanizmu tego wyłączone są jednak porty brzegowe, bo awaria takiego portu pozostaje bez związku z topologią sieci (w klasycznej wersji STP były one angażowane w sygnalizowanie awarii na równi z pozostałymi portami).

Zanim protokół RSTP zyskał rangę oficjalnego standardu, wiele z jego cech zostało opracowanych przez producentów sprzętu (głównie Cisco Systems, lecz także inne firmy) jako „specyficzne dla twórcy" (*proprietary*) rozszerzenia protokołu STP. Większość z tych rozszerzeń włączona została przez IEEE do poprawionego standardu 802.1d (patrz [802.1D-2004]), obejmującego zarówno klasyczną (STP), jak i nowszą (RSTP) wersję protokołu. W rezultacie wielosegmentowe sieci LAN mogą wykorzystywać STP w niektórych segmentach, a RSTP w innych (choć traci się wówczas wiele korzyści udostępnianych przez RSTP). W kolejnej edycji standardu ([8021Q-2005]) protokół RSTP rozszerzony został o obsługę wirtualnych sieci LAN; zachowany został dotychczasowy format ramki BPDU, co zapewnia kompatybilność wstecz, jednocześnie umożliwiono równoczesne budowanie i utrzymywanie wielu drzew rozpinających, po jednym dla każdej sieci VLAN.

3.4.2. 802.1ak: protokół wielorejestracyjny (MRP)

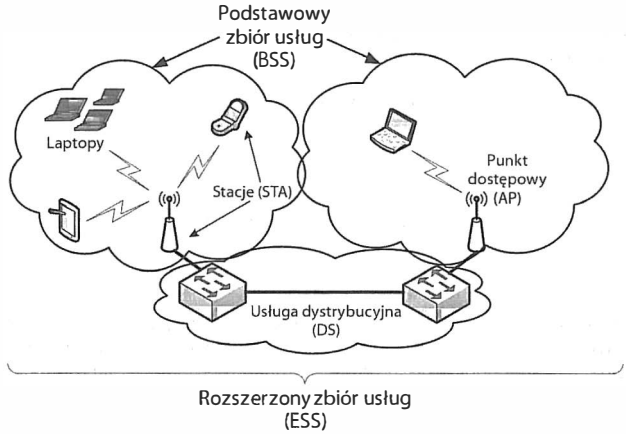
Protokół wielorejestracyjny (*Multiple Registration Protocol*, w skrócie MRP) to generalne narzędzie rejestrowania atrybutów pomiędzy stacjami w środowisku mostkowanej sieci LAN. W specyfikacji [802.1ak-2007] zdefiniowano dwa zastosowania („aplikacje") tego protokołu: MVRP (rejestrator sieci VLAN) i MMRP (rejestrator grup adresów MAC). Protokół MRP jest następnikiem frameworku GARP (*Generic Attribute Registration Protocol*) definiowanego w standardzie 802.1Q — MVRP i MMRP zastępują jego aplikacje GVRP i GMRP.

Zgodnie z protokołem MVRP, gdy tylko stacja końcowa skonfigurowana zostanie jako uczestnik sieci VLAN, informacja ta komunikowana jest przełącznikowi, do którego stacja ta jest podłączona; przełącznik z kolei propaguje tę informację do innych przełączników. Te z kolei wykorzystują otrzymaną informację do uaktualnienia swych baz filtracyjnych. Zauważmy, że zapisywany jest jedynie identyfikator sieci VLAN, jej topologia jest ukryta przed przełącznikami, a więc zmiana tejże topologii nie wymaga kosztownego „przeliczania" drzewa rozpinającego, jak to miało miejsce w przypadku protokołu GVRP; fakt ten stanowi najważniejszą przesłankę na rzecz migracji z GVRP do MVRP.

Analogicznie protokół MMRP jest narzędziem komunikowania przełącznikom informacji na temat zarejestrowania stacji końcowej w grupie adresów multicast. Przełączniki mogą wykorzystywać tę informację do wyznaczania portów, przez które kierować będą transmisje opatrzone określonymi adresami docelowymi multicast — bez tej informacji każdy ruch musiałby być rozprowadzany w trybie rozgłoszeniowym (broadcast), co skutkowałoby znacznie większym obciążeniem sieci. MMRP jest protokołem warstwy 2., podobnym do protokołów IGMP i MLD rezydujących w warstwie 3. i ich funkcji „inwigilowania" (*snooping*) — opisujemy je dokładniej w rozdziale 9.

3.5. Bezprzewodowe sieci LAN — IEEE 802.11 (Wi-Fi)

Jedną z najpopularniejszych obecnie technologii dostępu do Internetu jest Wi-Fi (od *Wireless Fidelity* — „wierność bezprzewodowa"), faktycznie bezprzewodowa wersja Ethernetu, utożsamiana z akronimem „802.11" pochodzącym — oczywiście — od przedrostka rozpoczynającego tytuły specyfikacji IEEE. Technologię tę zaprojektowano z myślą o zapewnieniu łączności taniej i wygodnej, a jednocześnie wydajnej w stopniu akceptowalnym dla większości aplikacji. Sieci Wi-Fi są łatwe do skonfigurowania, a większość urządzeń elektroniki użytkowej — takich jak komputery, smartfony, a nawet domowe urządzenia multimedialne — wyposażona jest w sprzęt umożliwiający dostęp do infrastruktury Wi-Fi. W wielu kafejkach, hotelach, portach lotniczych, centrach dużych miast itp. dostępne są publicznie hotspoty udostępniające Wi-Fi, technologia Wi-Fi znakomicie też toruje sobie drogę w krajach rozwijających się, które z różnych przyczyn (głównie ekonomicznych) nie mogą sobie pozwolić na wystarczające tempo rozwoju innych technologii dostępowych. Architektura przykładowej sieci zgodnej ze standardem IEEE 802.11 przedstawiona jest na rysunku 3.17.



Rysunek 3.17. *Terminologia standardu 802.11 definiującego bezprzewodowe sieci LAN (WLAN). Punkty dostępowe (AP) mogą być połączone za pomocą usługi dystrybucyjnej (DS), przewodowej lub bezprzewodowej; tworzą wówczas rozbudowaną formę sieci WLAN zwaną rozszerzonym zbiorem usług (ESS). Punkt dostępowy i skojarzone z nim stacje tworzą podstawowy zbiór usług (BSS). Zwykle ESS opatrzony jest identyfikatorem (ESSID) pełniącym rolę nazwy sieci*

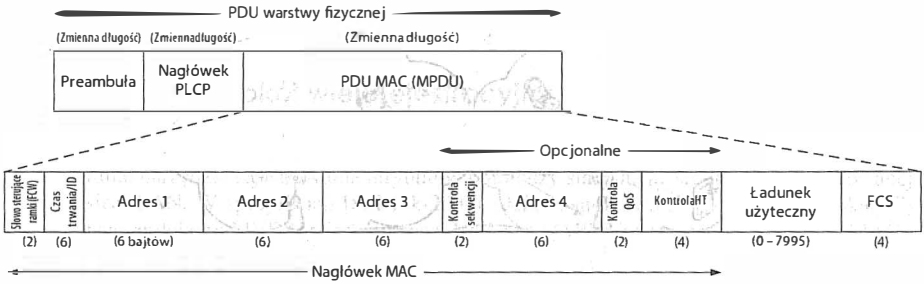
Konfiguracja ta obejmuje pewną liczbę *stacji* (STA), współdziałających z *punktami dostępowymi* (AP)[1]. Punkt dostępowy wraz ze skojarzonymi z nim stacjami określany jest mianem **podstawowego zbioru usług** (*Basic Service Set*, w skrócie BSS). Punkty dostępowe połączone są za pomocą **usługi dystrybucyjnej** (*Distribution Service* — DS), realizowanej zwykle w postaci magistrali przewodowej. Zespół połączonych ze sobą BSS-ów nazywamy **rozszerzonym zbiorem usług** (*Extended Service Set* — ESS). Funkcjonowanie sieci zgodnie z takim hierarchicznym schematem (Stacje -> Punkty dostępowe ->

[1] Zgodnie z terminologią [802.11-2007] punkt dostępowy także jest stacją (jej szczególnym rodzajem), co widoczne jest również na rysunku 3.17 — *przyp. tłum.*

ESS-y) nazywamy popularnie **trybem infrastrukturalnym** (*infrastructure mode*); standard 802.11 definiuje także inny tryb, zwany **trybem ad hoc** (*ad hoc mode*), w którym stacje komunikują się bezpośrednio ze sobą (*peer to peer*), bez udziału punktów dostępowych i usług dystrybucyjnych — zbiór tak połączonych stacji nazywany jest w ramach wspomnianego standardu **niezależnym podstawowym zbiorem usług** (*Independent Basic Service Set* — IBSS). Lokalna sieć bezprzewodowa (WLAN) stanowiąca kolekcję BSS-ów i (lub) IBSS-ów traktowana jest jako *zbiór usług* (*service set*) opatrzony identyfikatorem SSID (*Service Set IDentifier*) stanowiącym ciąg maksymalnie 32 znaków alfanumerycznych. Identyfikator rozszerzonego zbioru usług oznaczany jest akronimem ESSID. Identyfikatory poszczególnych BSS-ów przypisuje się poszczególnym punktom dostępowym zwykle podczas pierwszego konfigurowania sieci WLAN.

3.5.1. Ramki standardu 802.11

Wszystkie ramki sieci 802.11 dają się generalnie opisać za pomocą jednego głównego formatu, w ramach którego istnieje kilka różnych typów. Na rysunku 3.18 przedstawiono ów zasadniczy format oraz szczegółowy format ramki danych o maksymalnym rozmiarze.



Rysunek 3.18. *Podstawowy format ramki standardu 802.11 (według specyfikacji [802.11n-2009]). Format MPDU przypomina format znany z Ethernetu, posiada jednak dodatkowe pola zależne od typu usługi dystrybucyjnej łączącej punkty dostępowe, kierunku wysyłania ramki (do usługi dystrybucyjnej lub z niej), agregowania ramek itd. Pole Kontrola QoS wykorzystywane jest do sterowania wydajnością transmisji, natomiast pole Kontrola HT związane jest z cechą „dużej przepustowości" (high throughput) definiowanej w specyfikacji 802.11n*

Widoczna na rysunku ramka rozpoczyna się preambułą, pełniącą funkcje synchronizacyjne; szczegółowa postać preambuły zależna jest od stosowanego wariantu 802.11. Następne pole zawiera nagłówek **procedury zbieżności warstwy fizycznej** (PLCP — *Physical Layer Convergence Procedure*), dostarczający informacji na temat wykorzystywanej warstwy fizycznej, jednak w formacie niezależnym od specyfiki tej warstwy. Nagłówek ten transmitowany jest zazwyczaj z mniejszą szybkością niż pozostałe pola. Powody tego są dwa: zwiększenie prawdopodobieństwa poprawnego dostarczenia (mniejsza szybkość to generalnie mniejsza podatność na błędy) oraz zapewnienie zgodności ze starszymi urządzeniami (działającymi z mniejszą prędkością) i unikanie zakłóceń z ich strony.

Jednostka danych MAC (MAC PDU) ma format podobny do formatu w tradycyjnej ramce ethernetowej, zawiera jednak kilka dodatkowych pól. Dwubajtowe *Słowo sterujące ramki* (FCW — *Frame Control Word*) obejmuje m.in. dwubitowy znacznik identyfi-

kujący jeden z trzech typów ramki: **zarządczą** (*management*), **sterującą** (*control*) i **danych** (*data*). W ramach każdego z tych typów istnieje kilka podtypów; szczegółowy ich wykaz znajduje się w specyfikacji [802.1n-2009], w tabeli 7.1. Zestaw i znaczenie pozostałych pól są specyficzne dla poszczególnych typów ramek, które teraz dokładnie omówimy.

3.5.1.1. Ramki zarządcze

Ramki zarządcze wykorzystywane są do ustanawiania, utrzymywania i likwidowania skojarzeń między stacjami i punktami dostępowymi. Ich zawartość określa różnorodne aspekty tych skojarzeń, m.in. typ identyfikatora (SSID albo ESSID), szybkość transmisji, wspólną bazę czasową oraz fakt wykorzystywania szyfrowania. Informacja ta jest niezbędna w procesie „skanowania", czyli poszukiwania przez interfejs Wi-Fi najbliższych punktów dostępowych.

Skanowanie to jest procedurą polegającą na przełączaniu się kolejno na dostępne częstotliwości i biernym nasłuchiwaniu sygnałów identyfikujących osiągalne punkty dostępowe; oprócz biernego nasłuchiwania stacja może również prowadzić czynne próbkowanie, czyli wysyłanie ramek zarządczych typu *probe request*. Zgodnie ze specyfikacją 802.11 w procesie tym omijane są częstotliwości wykorzystywane do celów innych niż bezprzewodowy Ethernet (np. w aparaturze medycznej). Poniżej widoczny jest raport z prostego skanowania prowadzonego przy użyciu karty bezprzewodowej w komputerze z systemem Linux.

```
Linux# iwlist wlan0 scan
wlan0 Scan completed :
          Cell 01 - Address: 00:02:6F:20:B5:84
                    ESSID:"Grizzly-5354-Aries-802.11b/g"
                    Mode:Master
                    Channel:4
                    Frequency:2.427 GHz (Channel 4)
                    Quality=5/100 Signal level=47/100
                    Encryption key:on
                    IE: WPA Version 1
                       Group Cipher : TKIP
                       Pairwise Ciphers (2) : CCMP TKIP
                       Authentication Suites (1) : PSK
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s;
                          6 Mb/s; 12 Mb/s; 24 Mb/s; 36 Mb/s; 9 Mb/s;
                          18 Mb/s; 48 Mb/s; 54 Mb/s
                    Extra:tsf=0000009d832ff037
```

Rezultatem zainicjowanego skanowania jest wykrycie punktu dostępowego o adresie MAC `00:02:6F:20:B5:84` działającego jako *master*, czyli w trybie infrastrukturalnym, anonsującego się pod identyfikatorem ESSID `Grizzly-5354-Aries-802.11b/g` w kanale 4., czyli na częstotliwości 2,427 (więcej szczegółów na temat kanałów i częstotliwości przedstawiamy w punkcie 3.5.4). Wskaźniki jakości (`Quality`) i poziomu sygnału (`Signal Level`) zawierają informację o tym, jak dobrze skanująca stacja odbiera sygnał z punktu dostępowego; konkretne znaczenie tych wskaźników różni się nieco u poszczególnych producentów. Widzimy także użycie szyfrowania WPA na łączu (patrz punkt 3.5.5) oraz dostępność różnych szybkości transmisji, od 1 Mb/s do 54 Mb/s. Wartość `tsf` (*time synchronization function* — „funkcja synchronizacji czasowej") odzwierciedla sposób

pojmowania przez punkt dostępowy mechanizmów uwarunkowanych czasowo, takich jak tryb oszczędzania energii (patrz punkt 3.5.2).

Gdy punkt dostępowy anonsuje swój SSID, dowolna stacja może próbować nawiązania skojarzenia z nim; po udanej próbie większość obecnych sieci Wi-Fi wykonuje zabiegi konfiguracyjne niezbędne do zapewnienia stacji dostępu do Internetu (patrz rozdział 6.). Zwykle jednak operatorzy punktów dostępowych reglamentują w różny sposób (ze względów bezpieczeństwa) ich dostępność, np. wyłączają rozgłaszanie identyfikatora SSID; jednak to zabieg mało skuteczny, bo przy odrobinie wysiłku SSID można zgadnąć. Znacznie pewniejsze jest zabezpieczenie łącza hasłem oraz szyfrowanie transmisji —do tego tematu powrócimy w punkcie 3.5.5.

3.5.1.2. Ramki sterujące — RTS/CTS i potwierdzenia

Zadaniem ramek sterujących jest realizowanie pewnej formy sterowania przepływem oraz potwierdzanie otrzymywania ramek. Sterowanie przepływem to funkcja umożliwiająca odbiorcy zasygnalizowanie nadawcy, że powinien spowolnić tempo wysyłania ramek. Potwierdzanie otrzymywania ramek przez odbiorcę pozwala nadawcy upewnić się, że wysłane przez niego ramki dotarły poprawnie do celu (obie te koncepcje wykorzystywane są również przez protokół TCP funkcjonujący w warstwie transportowej, o czym będziemy pisać w rozdziale 15.). Sieci standardu 802.11 udostępniają opcjonalny mechanizm RTS/CTS (*Request-To-Send/Clear To Send* — żądanie pozwolenia na wysłanie/udzielenie pozwolenia na wysłanie) pozwalający na moderowanie transmisji ramek w związku ze sterowaniem przepływem: gdy mechanizm ten jest aktywny, stacja zamierzająca wysłać ramkę danych wysyła najpierw do adresata ramkę sterującą RTS; gdy adresat gotowy jest na przyjęcie ramki danych, potwierdza to przez odpowiedź w postaci ramki sterującej CTS. Po wymianie sygnałów RTS/CTS stacja nadawcza otrzymuje kwant czasu (określony w ramce CTS), w tym czasie może wysłać jedną lub więcej ramek, których otrzymywanie będzie potwierdzane przez odbiorcę. Schemat ten jest typowy dla sieci bezprzewodowych i wzorowany na podobnym mechanizmie, realizowanym od dawna (sprzętowo) w kablowej transmisji szeregowej.

Wymiana sygnałów CTS/RTS pomaga w unikaniu **problemu ukrytego terminala** (*hidden terminal problem*), polegającego na tym, że transmisje wykonywane do stacji odbiorczej równocześnie przez dwie (lub więcej) stacje sąsiednie nawzajem zakłócają się i żadna z tych transmisji nie może zostać poprawnie odebrana. Ponieważ ramki RTS i CTS są stosunkowo niewielkie, ich wymiana nie wpływa znacząco na obciążenie kanału. Generalnie punkt dostępowy inicjuje wymianę RTS/CTS tylko dla pakietów, których rozmiar przekracza ustaloną wartość, zwaną (w opcjach konfiguracyjnych) rozmiarem progowym (*packet size threshold* lub podobnie). Większość producentów ustala tę wartość domyślnie na (w przybliżeniu) 500 bajtów, o ile mechanizm RTS/CTS w ogóle ma być używany. W systemie Linux można ją ustawić w następujący sposób:

```
Linux# iwconfig wlan0 rts 250
wlan0 IEEE 802.11g ESSID:"Grizzly-5354-Aries-802.11b/g"
          Mode:Managed
          Frequency:2.427 GH
          Access Point: 00:02:6F:20:B5:84
          Bit Rate=24 Mb/s Tx-Power=0 dBm
          Retry min limit:7 RTS thr=250 B Fragment thr=2346 B
```

```
Encryption key:xxxx- ... -xxxx [3]
Link Quality=100/100 Signal level=46/100
Rx invalid nwid:0 Rx invalid crypt:0 Rx invalid frag:0
Tx excessive retries:0 Invalid misc:0 Missed beacon:0
```

Za pomocą polecenia `iwconfig` można ustawiać wartości wielu innych parametrów, m.in. progu fragmentacji (patrz podpunkt 3.5.1.3). Polecenie to umożliwia również obserwację statystyki sieci — liczbę błędów spowodowanych przez niepoprawny identyfikator ESSID lub błędny klucz szyfrowania, liczbę wykonywanych retransmisji itp. — co stanowić może podstawę ogólnej oceny kondycji łącza i wynikających z niej decyzji dotyczących trasowania (patrz [ETX]). W prostych sieciach WLAN, w których wystąpienie problemu ukrytego terminala jest mało prawdopodobne, ramki RTS/CTS niepotrzebnie generowałyby dodatkowy ruch, celowe więc może być wyłączenie mechanizmu RTS/CTS w ogóle, co uzyskuje się przez ustawienie rozmiaru progowego pakietu na „wysoką" wartość (1500 bajtów lub więcej).

W przewodowych sieciach Ethernet niewystąpienie kolizji oznacza z dużym prawdopodobieństwem poprawne dostarczenie ramki do odbiorcy. W sieciach bezprzewodowych sprawa nie przedstawia się aż tak prosto, ze względu na obecność wielu czynników warunkujących poprawność transmisji, takich jak niewystarczająca moc sygnału czy interferencja sygnałów. W związku z tym, w standardzie 802.11 rozszerzono oryginalną koncepcję retransmisji (sformułowaną w standardzie 802.3) do schematu retransmisji z potwierdzaniem (*acknowledgment*, w skrócie ACK). Nadawca po wysłaniu ramki lub grupy ramek na adres unicast oczekuje na potwierdzenie jej (ich) odebrania przez adresata, przez ograniczony interwał czasowy (potwierdzanie odebrania pojedynczej ramki jest opisane w standardach 802.11a/b/g, potwierdzanie grupy ramek — tzw. blokowe ACK — w standardach 802.11n i 802.11e). Nie stosuje się potwierdzania w stosunku do ramek wysyłanych na adresy multicast i broadcast w celu uniknięcia objawu „implozji ACK" (patrz rozdział 9.). Nieotrzymanie przez nadawcę potwierdzenia w założonym interwale czasowym traktowane jest jako objaw zagubienia ramki i skutkuje jej retransmisją.

Ponieważ jednak nieodebranie potwierdzenia wcale nie musi oznaczać zagubienia wysłanej ramki — bo równie dobrze mogła np. zaginąć ramka niosąca potwierdzenie albo odbiorca wysłał potwierdzenie zbyt późno — z retransmisją wiąże się niebezpieczeństwo „rozmnożenia" ramki (czyli wielokrotnego odebrania tej samej ramki). Oznacza to, że ramki retransmitowane muszą być jawnie odróżniane od oryginału wysyłanego po raz pierwszy. Zadanie to spełnia bit Retry w słowie sterującym ramki (FCW). Od stacji odbierającej oczekuje się obecności niewielkiej pamięci cache, w której gromadzona jest informacja o adresach, numerach sekwencyjnych i numerach fragmentu (patrz następny podpunkt) ostatnio odebranych ramek. Po otrzymaniu ramki retransmitowanej (czyli ramki z ustawionym bitem Retry) stacja sprawdza, czy oryginał ramki nie znajduje się już we wspomnianej pamięci cache — jeśli tak, retransmitowany egzemplarz jest odrzucany.

Wielkość interwału czasowego, przez który nadawca oczekuje na potwierdzenie odebrania ramki, zależna jest od długości łącza i wynikającej z niej wielkości tzw. **szczeliny czasu** (*slot time*) stanowiącej podstawową jednostkę pomiaru czasu w protokołach MAC standardu 802.11 (patrz punkt 3.5.3). Zarówno wartość samej szczeliny czasu, jak i limit oczekiwania na potwierdzenie odbioru ramki są konfigurowalne w większości systemów, choć sposób ich konfigurowania jest zróżnicowany. Dla niewielkich sieci,

instalowanych w gospodarstwach domowych lub małych biurach, najczęściej wystarczające okazują się wartości domyślne; dla sieci Wi-Fi operujących na dużych odległościach (patrz np. [MWLD]) konieczna jest jednak odpowiednia adjustacja tych wartości.

3.5.1.3. Ramki danych, fragmentacja i agregowanie ramek

Większość ramek krążących po (normalnie obciążonej) sieci Ethernet to ramki danych, spełniające oczywistą funkcję — przenoszenie użytecznej informacji. Zazwyczaj istnieje przełożenie „jeden do jednego" między pakietami odbieranymi z warstwy sieciowej a ramkami standardu 802.11 — jeden pakiet obsługiwany jest przez jedną ramkę — jednak standard ten przewiduje odstępstwo od tej zasady w postaci **fragmentacji**, czyli podziału dużych ramek na kilka mniejszych; w standardzie 802.11n zdefiniowano także **agregację** polegającą na komasacji wielu małych ramek do postaci jednej dużej ramki.

Gdy ramka podlega fragmentacji, każdy fragment posiada własny nagłówek MAC, własne pole kontrolne CRC i obsługiwany jest niezależnie od pozostałych fragmentów — możliwe jest też przeplatanie się transmisji fragmentów należących do różnych ramek. Fragmentowanie ramek przyczynia się do poprawy efektywności kanału obarczonego dużymi zakłóceniami: każdy fragment jest przesyłany i potwierdzany oddzielnie (z wyjątkiem przypadku, gdy stosowane jest potwierdzanie blokowe), a zagubienie fragmentu wymaga retransmisji mniejszej porcji danych niż pełna ramka. Fragmentacja stosowana jest jedynie do ramek z adresem docelowym unicast, ramki z adresami multicast i broadcast nie są fragmentowane.

Z fragmentacją ramek związane jest ich pole *Kontrola sekwencji*, podzielone na podpola *Numer fragmentu* (4 bity) i *Numer sekwencji* (12 bitów). Numer sekwencji jest jednakowy dla wszystkich fragmentów tej samej ramki, a kolejne fragmenty numerowane są kolejnymi liczbami naturalnymi (począwszy od 1) — 4-bitowe pole dopuszcza podział na maksymalnie 15 fragmentów. Liczba fragmentów danej ramki może być — oczywiście — różna i nie jest nigdzie zapisywana w sposób jawny, zamiast tego *ostatni* fragment rozpoznawany jest po zerowej wartości bitu *Więcej fragmentów* (w pozostałych fragmentach bit ten ma wartość 1). Te trzy elementy wystarczają do „poskładania" fragmentów w oryginalną ramkę (co formalnie określa się mianem **defragmentacji**) po dotarciu wszystkich do stacji odbierającej.

Podstawową przesłanką stosowania fragmentacji jest fakt, że mniejszy rozmiar przesyłanych danych oznacza mniejsze prawdopodobieństwo błędu (o tym dokładniej za chwilę). Mimo to, stosuje się ją raczej rzadko, ponieważ wymaga zwykle zaawansowanych zabiegów konfiguracyjnych; pozostawienie jej z ustawieniami domyślnymi prawdopodobnie tylko (nieznacznie) pogorszy wydajność transmisji, zamiast ją wyraźnie poprawić. Jako rozmiar fragmentu (stanowiący jednocześnie **próg fragmentacji**, czyli maksymalny rozmiar ramek, które fragmentowane nie są) przyjmuje się zazwyczaj wartości od 256 bajtów do 2 kilobajtów. Ustawienie tego progu na dużą wartość (w punktach dostępowych produkcji Linksys jest to domyślnie 2437 bajtów) oznacza w praktyce rezygnację z fragmentowania.

Aby zrozumieć korzyści możliwe do uzyskania dzięki fragmentacji, wykonajmy proste obliczenie. Jeśli prawdopodobieństwo przekłamania jednego bitu (BER, od *Bit Error Rate*) wynosi P, to prawdopodobieństwo poprawnego dostarczenia *N*-bitowej ramki równe jest

$(1-P)^N$; wartość ta maleje wraz ze wzrostem N, zatem skracanie ramki przyczynia się do jej bezbłędnego dostarczenia. Jeżeli więc podzielimy N-bitową ramkę na K-bitowe fragmenty, otrzymamy $Q = \lceil N/K \rceil$ fragmentów [2], z których każdy zostanie poprawnie dostarczony z prawdopodobieństwem $(1-P)^K$. Dla przykładowej ramki o rozmiarze 1500 bajtów (12 000 bitów) i (stosunkowo dużej) wartości $P = 10^{-4}$ otrzymujemy prawdopodobieństwo $(1-10^{-4})^{12\,000} \approx 30\%$ bezbłędnego dostarczenia ramki za pierwszym razem, średnio więc będziemy musieli ponowić transmisję trzy lub cztery razy.

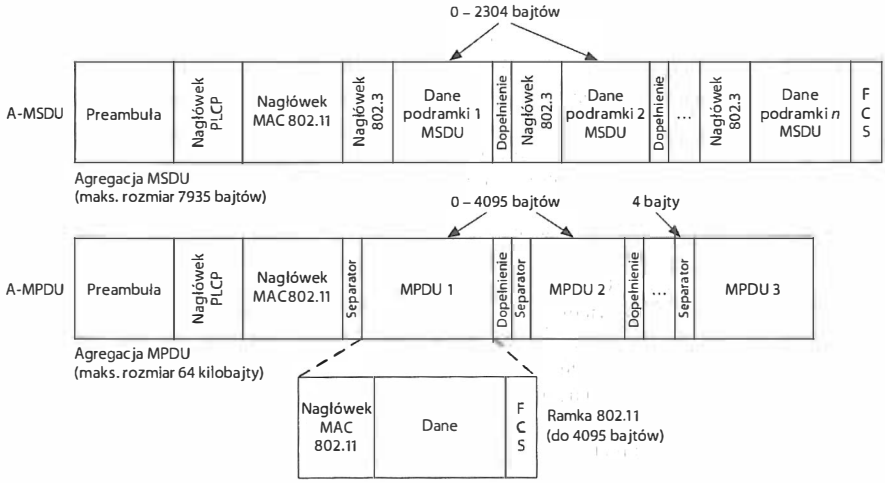
Jeżeli podzielimy wspomnianą ramkę na trzy fragmenty po 4000 bitów każdy, prawdopodobieństwo bezbłędnego dostarczenia każdego fragmentu z osobna za pierwszym razem wynosić będzie $(1-10^{-4})^{4000} \approx 67\%$. Prawdopodobieństwo, że za pierwszym razem dostarczone zostaną (odpowiednio) 3 fragmenty, 2 fragmenty, 1 fragment lub żaden fragment, wynosi (odpowiednio) $0{,}67^3 \approx 0{,}3$, $0{,}67^2 \cdot 0{,}33 \approx 0{,}44$, $0{,}67 \cdot 0{,}33^2 \approx 0{,}22$ i $0{,}33^3 \approx 0{,}04$. Chociaż prawdopodobieństwo bezbłędnego dostarczenia za pierwszym razem wszystkich fragmentów pozostaje na poziomie tym samym, co dla niepofragmentowanej ramki, to jednak całkiem niemałe są prawdopodobieństwa bezbłędnego dostarczenia za pierwszym razem dwóch fragmentów lub jednego. Retransmisja jednego lub dwóch fragmentów wymagać będzie mniej czasu niż retransmisja pełnej ramki. Oczywiście, z retransmisją każdego fragmentu wiążą się pewne dodatkowe narzuty czasu, jeśli więc wartość BER jest praktycznie zerowa, fragmentacja jedynie pogarsza efektywność kanału wskutek zwiększenia liczby przesyłanych ramek.

Jak wcześniej wspomnieliśmy, w standardzie 802.11n zdefiniowano możliwość *agregowania* ramek i to w dwojaki sposób. Pierwszy z nich, określany akronimem A-MSDU (od *Aggregated MAC Service Data Unit*), polega na upakowaniu wielu kompletnych ramek standardu 802.3 w pojedynczej ramce 802.11. Sposób drugi, oznaczany akronimem A-MPDU (od *Aggregated MAC Protocol Data Unit*), oznacza upakowanie wielu jednostek MPDU z identycznymi wartościami adresu źródłowego, adresu docelowego i ustawień QoS. Oba sposoby zilustrowano na rysunku 3.19.

Podstawową zaletą wariantu A-MSDU jest oszczędność miejsca wynikająca z różnicy długości nagłówków 802.11 i 802.3 — na każdej składowej ramce zyskujemy 36–14 = 22 bajty. Maksymalna wielkość ramki agregatu — 7935 bajtów — oznacza miejsce dla 100 niewielkich (np. 50-bajtowych) albo pięciu dużych (po 1500 bajtów każdy) pakietów danych. Oczywiście, agregowanie zwiększa prawdopodobieństwo wystąpienia błędu transmisji — jest on bardziej prawdopodobny przy rozmiarze 7935 bajtów niż przy rozmiarze 50 czy 1500 bajtów. Ponadto, ponieważ integralność agregatu kontrolowana jest za pomocą pojedynczej sumy kontrolnej (FCS), w przypadku błędu transmisji musi on być retransmitowany w całości.

Z kolei agregat A-MPDU to grupa kilku (maksymalnie 64) ramek 802.11 (zwanych *podramkami — subframes*), każda ma własny nagłówek MAC i własną sumą kontrolną. Maksymalna wielkość całego agregatu wynosi 64 kB, czyli ponad 1000 mniejszych pakietów albo 40 dużych. Ze względu na większy potencjalnie rozmiar agregatu, bardziej prawdopodobne jest jego uszkodzenie podczas transmisji, ponieważ jednak każda podramka posiada własną sumę kontrolną, konieczna jest retransmisja tylko uszkodzonych podramek. Jest to możliwe dzięki *potwierdzaniu blokowemu*, wywodzącemu się ze

[2] Zapis $\lceil x \rceil$ oznacza wynik zaokrąglenia wartości x w górę do najbliższej liczby całkowitej — *przyp. tłum.*



Rysunek 3.19. *Dwa warianty agregowania ramek według specyfikacji 802.11n. W agregacie A-MSDU integralność całości kontrolowana jest za pomocą pojedynczej sumy kontrolnej, w agregacie A-MPDU poszczególne ramki składowe rozdzielone są 4-bajtowym separatorem, każda z nich zawiera własną sumę kontrolną i może podlegać niezależnemu potwierdzaniu odebrania (i niezależnej retransmisji w przypadku odrzucenia lub zagubienia)*

standardu 802.11e i ostatecznie zdefiniowanemu w 802.11n, a polegającemu na osobnym potwierdzaniu otrzymania każdej podramki. Jest to mechanizm podobny (ale tylko koncepcyjnie, nie w szczegółach) do selektywnego potwierdzania wykonywanego w ramach protokołu TCP (patrz rozdział 14.). Reasumując, mimo iż agregacja A-MSDU może być korzystna w działających bezbłędnie sieciach przenoszących duże ilości małych pakietów, w rzeczywistych warunkach lepsze wyniki daje agregacja A-MPDU (patrz np. [S08]).

3.5.2. Tryb oszczędzania energii i funkcja synchronizacji czasu (TSF)

Specyfikacja 802.11 definiuje także szczegóły zachowania stacji w stanie ograniczonego poboru mocy, nazywanym **trybem oszczędzania energii** (*power save mode*, w skrócie PSM). Przebywanie stacji w tym stanie sygnalizowane jest przez ustawienie odpowiedniego bitu w słowie sterującym (FCW) ramek wysyłanych przez tę stację. Współpracujący ze stacją punkt dostępowy (AP) reaguje na tę informację w ten sposób, że powstrzymuje się od wysyłania ramek do stacji, buforując je do momentu, aż stacja sama zażąda ich dostarczenia. Punkt dostępowy okresowo wysyła specjalny typ ramek zarządczych, zwanych **ramkami beacon**, zawierających informacje na temat m.in. SSID, kanału i dane uwierzytelniające; jeżeli punkt dostępowy honoruje mechanizm PSM u współpracujących stacji i w buforze oczekują niewysłane ramki, w słowie FCW ramki *beacon* ustawiony zostaje bit informujący o tym fakcie. Gdy stacja wchodzi w tryb PSM, oczekuje w tym stanie do momentu odebrania ramki *beacon* — wówczas przechodzi w tryb pełnego poboru energii i sprawdza, czy wśród oczekujących w buforze AP ramek są przeznaczone dla niej.

Tryb PSM należy stosować ostrożnie i ze zrozumieniem. To prawda, że opóźnia on wyczerpywanie się baterii, jednakże karta sieciowa to tylko jeden z elementów stacji (komputera), wcale nie najważniejszy w bilansie zużycia energii, w porównaniu z takimi konsumentami jak dysk twardy czy monitor. Ceną płaconą za ewentualne oszczędności jest znaczący spadek wydajności (spowodowany długimi okresami bezczynności przeplatającymi się z transmisją ramek) oraz dodatkowy czas potrzebny na przełączanie trybów (patrz [SHK07]).

Aby stacja mogła wychodzić z trybu PSM w najbardziej odpowiednim do tego momencie, czyli tuż przed pojawianiem się ramki *beacon*, konieczne jest uzgodnienie wskazania czasu między tą stacją a punktem dostępowym. Na gruncie Wi-Fi uzgodnienie takie wykonywane jest za pomocą **funkcji synchronizacyjnej** (*Time Synchronization Function*, w skrócie TSF). Każda ze stacji utrzymuje 64-bitowy licznik, inkrementowany co mikrosekundę, synchronizowany z licznikami innych stacji w sieci; każda stacja, otrzymująca komunikat TSF (którego sednem jest wartość wspomnianego licznika stacji wysyłającej) powinna dostosować swój własny licznik do tej wartości, o ile ten ma wartość mniejszą. Procedura ta umożliwia synchronizację liczników z dokładnością do 4μs plus maksymalny czas przechodzenia sygnału przez warstwę fizyczną. Jak łatwo zauważyć, liczniki nigdy nie są „cofane", choć stacje z trochę wolniejszymi zegarami są konsekwentnie przyspieszane przez szybszych partnerów.

Wraz z włączeniem tematu jakości usługi (QoS — *Quality of Service*) do standardu 802.11 (co nastąpiło w specyfikacji 802.11e) rozszerzono koncepcję obsługi trybu PSM, likwidując konieczność „budzenia się" stacji przy każdej ramce *beacon* na rzecz dłuższego okresu bezczynności, liczonego wielokrotnością pojawiania się takich ramek. Mechanizm ten, zwany **przesyłaniem z automatycznym oszczędzaniem energii** (*Automatic Power Save Delivery*, w skrócie APSD) jest przydatny dla urządzeń szczególnie wymagających pod względem oszczędzania energii.

Specyfikacja 802.11n rozszerza także tradycyjne możliwości trybu PSM w kontekście stacji wyposażonych w kilka instalacji radiowych (patrz opis technologii MIMO w podpunkcie 3.5.4.2). W nowym trybie, zwanym **oszczędzaniem energii w trybie multipleksowania przestrzennego** (*spatial multiplexing power save mode*) w stan oszczędzania energii wprowadzone zostają wszystkie układy radiowe z wyjątkiem jednego, czuwającego w trybie pełnego poboru mocy. Specyfikacja ta definiuje także rozszerzenie APSD o nazwie *Power Save Multi-Poll* (w skrócie PSMP), zgodnie z którym punkt dostępowy przesyła do stacji informację o tym, jak długo może ona pozostawać w trybie niskiego poboru energii, kiedy rozpocznie się transmisja przeznaczonych dla niej danych oraz kiedy może ona rozpocząć wysyłanie. Umożliwia to zaplanowanie transmisji ramek w obu kierunkach między stacją a punktem dostępowym.

3.5.3. Sterowanie dostępem do nośnika w sieciach 802.11

W sieciach bezprzewodowych pojęcie „kolizji" i ich wykrywania nabiera nowego znaczenia w porównaniu z sieciami przewodowymi opisywanymi przez standard 802.3. Fale radiowe są nośnikiem innej natury niż wiązka przewodów, a zamiast wykrywania kolizji post factum bardziej klarowna wydaje się koncepcja ich unikania — realizowana zarówno w formie scentralizowanej, jak i rozproszonej. W standardzie 802.11 definiowane

są trzy podejścia do kontroli współdzielenia nośnika bezprzewodowego: **funkcja koordynacyjna punktu** (*Point Coordination Function* — PCF), **rozproszona funkcja koordynacyjna** (*Distributed Coordinating Function* — DCF) i **mieszana funkcja koordynacyjna** (*Hybrid Coordination Function* — HCF). Definicja HCF wprowadzona została do standard 802.11 w specyfikacji [802.11-2007] wraz z dodaniem obsługi QoS w standardzie 802.11e i wykorzystywana jest także w standardzie 802.11n. Implementacja DSF jest obowiązkowa w stacji każdego typu, również w punktach dostępowych, implementacja PCF jest opcjonalna i rzadko wykonywana (nie będziemy więc wnikać w szczegóły tego mechanizmu). Implementacje HCF pojawiają się w stosunkowo nowych urządzeniach Wi-Fi, obsługujących koncepcję QoS, takich jak punkty dostępowe zgodne z 802.11n i wcześniejszą specyfikacją 802.11e. Zajmiemy się najpierw szczegółami DCF, po czym opiszemy HCF w kontekście QoS. DCF jest realizacją mechanizmu **wielodostępu z badaniem nośnika i unikaniem kolizji** (*Carrier Sense, Multiple Access with Collision Avoidation*, w skrócie CSMA/CA) — mechanizmu stosowanego zarówno w trybie infrastrukturalnym, jak i w trybie ad hoc. Zgodnie z założeniami tego mechanizmu, stacja zamierzająca wysyłać dane obserwuje nośnik w celu stwierdzenia, czy jest on wolny od jakichkolwiek transmisji i jeżeli tak, wykorzystuje okazję do zainicjowania własnej transmisji; jeżeli nie, odczekuje pewien losowy odcinek czasu i powraca do obserwacji. Zachowanie to podobne jest do wielodostępu CSMA/CD w sieciach przewodowych. Arbitraż kanałów w 802.11 oparty jest jednak na mechanizmie CSMA/CA poszerzonym o uwzględnianie priorytetów dostępu określonych stacji i określonych typów ramek.

Badanie nośnika bezprzewodowego 802.11 wykonywane jest zarówno fizycznie, jak i wirtualnie. Stwierdzenie przez stację, że nośnik nie jest zajęty, nie skutkuje natychmiastowym rozpoczęciem przez nią transmisji: stacja wchodzi najpierw w stan oczekiwania przez okres czasu określony przez parametr DIFS (*Distributed Inter-Frame Space*), aby dać szansę rozpoczęcia transmisji stacjom o wyższym priorytecie. Gdy w tym czasie w kanale pojawi się jakakolwiek transmisja, oczekiwanie to jest ponawiane przez stację; gdy nośnik przez cały ten czas pozostanie wolny, stacja wykonuje procedurę biernego oczekiwania (opisywaną w podpunkcie 3.5.3.3) konieczną z punktu widzenia unikania kolizji; procedura ta inicjowana jest także po udanej lub nieudanej transmisji, sygnalizowanej otrzymaniem (odpowiednio: brakiem) potwierdzenia ze strony odbiorcy. W przypadku nieudanej transmisji czas oczekiwania jest inny, określony przez parametr EIFS (*Extended Inter-Frame Space*). Przyjrzyjmy się szczegółom mechanizmu DCF, zarówno w jego wirtualnym, jak i fizycznym aspekcie.

3.5.3.1. Wirtualne badanie nośnika, RTS/CTS i wektor alokacji sieci (NAV)

W protokole MAC 802.11 mechanizm wirtualnego badania nośnika realizowany jest na bazie pola *Czas trwania* ramki i opiera się na nasłuchiwaniu przez stację transmisji adresowanych do innych stacji. Pole to znajduje się zarówno w ramkach RTS i CTS, jak i w „regularnych" ramkach danych, a służy do szacowania czasu, przez który nośnik pozostanie zajęty transmitowaniem wysłanej ramki. Stacja wysyłająca ramkę umieszcza w tym polu wartość wyliczaną w oparciu o rozmiar tej ramki, szybkość transmisji oraz charakterystykę warstwy fizycznej (głównie szybkość transmisji w tej warstwie). Każda stacja utrzymuje lokalny licznik, określany akronimem NAV, od *Network Allocation Vector*; jest to wektor alokacji sieci wykorzystywany przez nią do określania czasu, jaki

zajmie transmisja wysyłanej właśnie ramki, czyli czasu niezbędnego oczekiwania przed próbą wysłania ramki następnej. Gdy stacja — w wyniku nasłuchiwania ramek krążących między innymi stacjami — napotka ramkę o wartości pola *Czas trwania* większej niż wartość aktualna na jej prywatnym liczniku NAV, powiększa ów licznik do wartości zaobserwowanej we wspomnianym polu. Licznik NAV wyrażony jest w jednostkach czasowych i konsekwentnie dekrementowany przez mechanizm zegarowy stacji oraz zerowany po odebrania ramki potwierdzenia (ACK). Niezerowa wartość tego licznika jest dla stacji tożsama z założeniem, że nośnik jest aktualnie zajęty.

3.5.3.2. Fizyczne badanie nośnika

W standardzie 802.11 nieodłącznym elementem definicji każdej warstwy fizycznej (czyli kombinacji „technologia radiowa-częstotliwość") jest określenie funkcji służącej do badania stanu zajętości kanału na podstawie transmitowanej energii oraz rozpoznawania kształtu fali (najczęściej dobrze ukształtowanej PLCP). Funkcja ta oznaczana jest akronimem CCA, od *Clear Channel Assessment*, dosł. „ocena czystości kanału", jej implementacja zależna jest od specyfiki odnośnej warstwy fizycznej. To właśnie CCA jest wspomnianym wcześniej fizycznym aspektem badania zajętości nośnika, który w połączeniu z aspektem wirtualnym jest podstawą mechanizmu CSMA/CA.

3.5.3.3. Unikanie kolizji dzięki DCF i procedura wyczekiwania

Wydawałoby się, że stwierdzenie przez stację, iż nośnik jest prawdopodobnie wolny — bo wyzerował się licznik NAV, a funkcja CCA nie raportuje zajętości kanału — uprawnia tę stację do natychmiastowego rozpoczęcia transmisji. Tak jednak nie jest: jeżeli weźmiemy pod uwagę, że na zwolnienie kanału mogło czekać wiele stacji, to gdyby wszystkie one rozpoczęły zmasowany atak na zwolniony właśnie nośnik, uruchomione transmisje nawzajem kolidowałyby ze sobą i prawdopodobnie żadna z nich nie zakończyłaby się powodzeniem. Dla zapobieżenia takiemu stanowi rzeczy, każda stacja przed rozpoczęciem transmisji przez zwolniony właśnie nośnik wchodzi w krótki stan oczekiwania, co określane jest mianem **procedury wyczekiwania** (*backoff procedure*). Czas tego oczekiwania równy jest losowej ilości jednostek zwanych „szczelinami czasu" (*slot time*); wartość takiej jednostki jest specyficzna dla warstwy fizycznej, zwykle nie przekracza kilkudziesięciu mikrosekund. Ilość jednostek oczekiwania jest liczbą losową o rozkładzie równomiernym wybieraną z przedziału (0 ; CW), gdzie CW (skrót od *contention window* — „okno rywalizacji") jest wartością zawartą między wartościami aCWmin i aCWmax, specyficznymi dla warstwy fizycznej. Dokładniej mówiąc, wartość CW równa jest 2^k-1 dla pewnego k; początkowo k ma największą wartość spełniającą warunek $2^k-1 \geq$ aCWmin (czyli w przybliżeniu $\log_2$ aCwMin) i zwiększane jest o 1 po każdej udanej transmisji, aż do osiągnięcia warunku $2^{k+1}-1 >$ aCWmax (wartość CW rośnie więc wykładniczo, dlatego wspomniane określenie *backoff procedure* opatruje się często przymiotnikiem *exponential*). Czytelnicy natychmiast odnajdą tu analogię do *backoff procedure* wykonywanej w związku z unikaniem kolizji w protokole CSMA/CD.

W środowisku transmisji bezprzewodowej *wykrywanie* kolizji byłoby działaniem niepraktycznym — trudno byłoby zarówno nadajnikowi, jak i odbiornikowi śledzić obecność transmisji innych niż ich własna, dlatego bardziej rozsądne okazuje się postępowanie ukierunkowane na *unikanie* kolizji, opisane powyżej. O tym, czy kolizji faktycznie

udało się uniknąć, przekonuje się nadawca, odbierając potwierdzenie od odbiorcy (albo doświadczając braku takiego potwierdzenia). Po pozytywnym zweryfikowaniu poprawności odebranej ramki odbiorca odczekuje pewien interwał czasu (określony przez parametr SIFS — *Short Interframe Space*), po czym wysyła ramkę ACK niezależnie od stanu zajętości nośnika. Ponieważ wartość SIFS zawsze jest mniejsza niż DIFS, więc w praktyce stacje generujące ramki ACK faworyzowane są w uzyskiwaniu dostępu do nośnika. Nieotrzymanie ramki ACK przez nadawcę w trakcie założonego limitu czasowego traktowane jest jako zagubienie ramki; stacja ponawia opisane wcześniej postępowanie w celu ponownego jej wysłania. W podobny sposób traktowane jest nieotrzymanie ramki CTS w odpowiedzi na wysłaną ramkę RTS, inny jest tylko wspomniany limit czasowy oczekiwania (określony przez stałą CTStimeout).

3.5.3.4. HCF i jakość usługi w specyfikacjach 802.11e/n

Klauzule 5, 6, 7 i 9 standardu 802.11 ([802.11-2007]), bazujące częściowo na pracach grupy 802.11e w ramach IEEE, poświęcone są wprowadzeniu mechanizmu **jakości usługi** (QoS — *Quality of Service*) do standardu 802.11 i jego interfejsów, na potrzeby aplikacji multimedialnych, m.in. VoIP (*Voice over IP*, dosł. „głos przez IP") i strumieniowanego wideo (z tego względu akronim WMM — od *Wi-Fi multimedia* — jest często używanym synonimem dla 802.11e). Kwestia użyteczności QoS w danej sieci powiązana jest bezpośrednio ze stopniem jej obciążenia i wykorzystywanymi w niej aplikacjami multimedialnymi. Dla sieci o małym nasyceniu ruchem mechanizm QoS jest w zasadzie niepotrzebny, choć niezależnie od niego dostępne są inne mechanizmy standardu 802.11e (np. potwierdzanie blokowe czy APSD); w sieciach wysoce obciążonych (i przeciążonych) QoS jest niezbędne do zapewnienia dobrej jakości (czyli możliwie małego obarczenia błędami) takich usług jak np. VoIP. Standard 802.11e jest specyfikacją stosunkowo nową, można więc oczekiwać, że implementujące ją urządzenia będą droższe i bardziej skomplikowane od swych poprzedników.

W związku z QoS pojawiło się w 802.11e kilka nowych terminów, m.in. **stacja QoS** (w skrócie QSTA), **punkt dostępowy QoS** (QAP) i **BSS z obsługą QoS** (QBSS). Zdefiniowane w 802.11n „stacje wysokoprzepustowe" (*high-throughput stations* — HT STA) także są stacjami QSTA. Wszystkie nowsze urządzenia, wyposażone w obsługę QoS, z zasady zapewniają zgodność ze starszymi mechanizmami pozbawionymi QoS.

Funkcja koordynacyjna w nowej postaci, czyli HCF (od *Hybrid Coordination Function* — mieszana funkcja koordynacyjna), łączy dwie metody sterowania dostępem do kanału: rywalizacyjną, zwaną EDCA (*Enhanced DCF Channel Access*) oraz (stosowaną rzadziej) rezerwacyjną, oznaczaną akronimem HCCA (*HFCA Controlled Channel Access*). Metody te wspomagane są elementami mechanizmu *kontroli dopuszczenia* (*admission control*), zabraniającego nawiązywania nowych połączeń w sytuacji przeciążenia sieci.

ECDA stanowi rozszerzenie opisanej wcześniej funkcji DCF o różnicowanie usług na bazie ich priorytetów. Zdefiniowano osiem wartości priorytetu użytkownika (UP — *User Priority*), od najniższego (1) do najwyższego (7); priorytet identyfikowany wartością 0 ma rangę pośrednią między priorytetami 2 i 3. Wartości te odwzorowane są w cztery **kategorie dostępu** (AC — *Access Categories*): usługom **działającym w tle** (*background*) przyporządkowano (nominalnie) priorytety 1 i 2, priorytety 0 i 3 przyporządkowane są do tzw. **ruchu niegwarantowanego** (*best-effort delivery*), priorytetami 4 i 5 objęte są

usługi *wideo*, zaś priorytetami 6 i 7 — usługi *głosowe i audio*. W rywalizacji o dostęp do kanału (na bazie rozproszonego algorytmu DCF) stacje wspomagane są różnym poziomem tzw. **preferencji transmisyjnych** (*transmit opportunities*, w skrócie TXOP). Istotą takiej preferencji jest dedykowanie danej stacji pewnego odcinka czasu, w którym może ona nadać kolejno kilka ramek z danymi, bez konieczności ubiegania się o dostęp do nośnika osobno przy transmisji każdej z nich. W tym celu wartości wielu parametrów odzwierciedlających specyfikę warstwy fizycznej (m.in. DIFS, `aCWmin` i `aCWmax`) stają się elementami konfiguracji; wartości te komunikowane są stacjom QSTA za pomocą ramek zarządczych.

Metoda HCCA jest luźno związana z algorytmem PCF i opiera się na koncepcji „przepytywania" (*polling*) nośnika. Zaprojektowano ją z myślą o synchronicznym szeregowaniu dostępu do nośnika i pierwszeństwie przed realizowanym przez EDCA dostępem opierającym się na rywalizacji. Synchronizowaniem dostępu do nośnika zajmuje się koordynator hybrydowy (HC — *Hybrid Coordinator*) działający w ramach punktu dostępowego (AP). Stacja QSTA przed właściwą transmisją może wysłać do AP *specyfikację ruchu* (TSPEC — *Traffic SPECification*) opatrzoną wartością priorytetu UP od 8 do 15. W odpowiedzi na tę specyfikację HC może przydzielić stacji preferencję TXOP umożliwiającą jej zrealizowanie transmisji w pierwszej fazie dystansu czasowego między kolejnymi ramkami *beacon* (CFP — *Contention-Free Phase*), wolnej od rywalizacji o nośnik w ramach EDCA (która to rywalizacja rozgrywa się w fazie drugiej, oznaczanej CP — od *Contention Phase*). Koordynator hybrydowy może jednak zabronić przydzielania TXOP w odpowiedzi na TSPEC, na podstawie założeń polityki dostępu, sformułowanych przez administratora sieci.

Funkcja HCF wykorzystuje więc mechanizm wirtualnego badania zajętości nośnika (czyli wirtualny aspekt DCF) w celu ochrony stacji korzystających z przywilejów TXOP przed interferencją ze strony stacji rywalizujących o dostęp do nośnika. Zauważmy, że w sieci zawierającej zarówno QSTA, jak i klasyczne stacje nieimplementujące QoS, funkcje HCF i DCF mogą egzystować równolegle, choć — oczywiście — klasyczne stacje nie będą wykorzystać możliwości QoS. Co się tyczy sieci ad hoc, to brak punktu dostępowego oznacza brak koordynatora hybrydowego, czyli niemożność obsługi żądań TSPEC; w takich sieciach można wykorzystywać HCF, lecz przywileje TXOP urzeczywistniane mogą być tylko w granicach wyznaczonych przez funkcjonowanie EDCA.

3.5.4. Parametry warstwy fizycznej: szybkości, kanały i częstotliwości

Standard [802.11-2007] komasuje poprawki będące przedmiotem specyfikacji 802.11a, 802.11b, 802.11d, 802.11g, 802.11h, 802.11i, 802.11j i 802.11e. Specyfikacja 802.11n została uwzględniona w wersji standardu z roku 2009 ([802.11n-2009]). Większość wymienionych poprawek dotyczy dodatkowych modulacji, kodowania i częstotliwości operacyjnych w sieciach 802.11, jednakże 802.11n definiuje dodatkowo równoległe strumienie danych i agregowanie ramek (opisane w podpunkcie 3.5.1.3). Nie będziemy w tym miejscu poświęcać zbyt wiele uwagi warstwie fizycznej, by jednak dać czytelnikom pojęcie o ogromie oferowanych przez nią opcji, zestawiamy w tabeli 3.2 specyfikacje precyzujące jej szczegóły.

Tabela 3.2. *Specyfikacje standardu 802.11 opisujące warstwę fizyczną*

Standard (klauzula)	Szybkość (w Mb/s)	Zakres częstotliwości, modulacja	Zestaw kanałów
802.11a (klauzula 17)	6, 9, 12, 18, 24, 36, 48, 54	5,16 – 5,35 GHz, 5,725 – 5,825 GHz OFDM	34 – 165 (zależnie od kraju) opcje szerokości kanału 20 MHz/10 MHz/ 5 MHz
802.11b (klauzula 18)	1, 2, 5.5, 11	2,401 – 2,495 GHz DSSS	1 – 14 (zależnie od kraju)
802.11g (klauzula 19)	1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48, 54 (plus 22, 33)	2,401 – 2,495 GHz OFDM	1 – 14 (zależnie od kraju)
802.11n	6,5 – 600 z wieloma opcjami (do 4 strumieni MIMO)	Tryby 2,4 GHz i 5 GHz z kanałami o szerokości 20 MHz lub 40 MHz OFDM	1 – 13 (2,4 GHz band); 36 – 196 (pasmo 5 GHz) (zależnie od kraju)
802.11y	(taka sama jak w 802.11-2007)	3,650 – 3,700 GHz (wymagana licencja) OFDM	1 – 25, 36 – 64, 100 – 161 (zależnie od kraju)

W pierwszej kolumnie wymienione są oryginalne nazwy poszczególnych specyfikacji i ew. umiejscowienie tychże specyfikacji w zbiorczej wersji [802.11-2007]. Wart uwagi jest fakt, że urządzenia standardu 802.11b/g operują w paśmie ISM (*Industrial, Scientific, and Medical* — przemysłowo-naukowo-medycznym) o częstotliwości 2,4 GHz, podczas gdy urządzenia standardu 802.11b/a operują w 5 GHz paśmie U-NII (*Unlicensed National Information Infrastructure* — dosł. „nielicencjonowana narodowa infrastruktura informacyjna"), natomiast urządzenia 802.11n mogą wykorzystywać oba te pasma. Poprawka 802.11y przeznacza pasmo 3,65 – 3,70 GHz do licencjonowanego użytku na obszarze USA. Z tabeli 3.2 wynika praktyczny wniosek, iż urządzenia 802.11b/g nie mogą interferować z urządzeniami 802.11a (i vice versa), natomiast urządzenia obu tych standardów mogą być zakłócane przez urządzenia 802.11n, jeśli te nie są właściwie skonfigurowane.

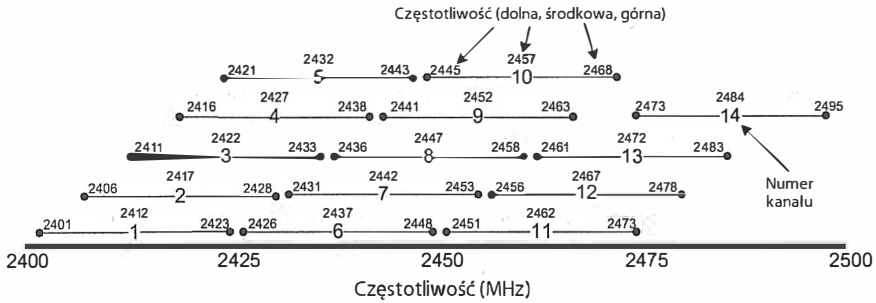
3.5.4.1. Kanały i częstotliwości

W rezultacie porozumienia różnych gremiów regulacyjnych (w USA gremium takim jest Federalna Komisja Łączności — *Federal Communications Commission*, FCC, a w Polsce Urząd Komunikacji Elektronicznej) spektrum dostępnych częstotliwości radiowych podzielone zostało na zakresy przydzielone do wykorzystywania w określonych celach na całym świecie. Wykorzystywanie określonego pasma do konkretnych celów może — choć nie musi — być obwarowane licencją, zależnie od lokalnych uregulowań prawnych. Jak to wynika z tabeli 3.2, każdy ze standardów definiuje określony zestaw kanałów, których wykorzystywanie do określonych zastosowań, przy ustalonych poziomach mocy, regulowane jest polityką telekomunikacyjną kraju lub domeny.

Kolejne numery kanałów Wi-Fi odpowiadają kolejnym wielokrotnościom 5 MHz dodawanym do pewnej bazowej częstotliwości środkowej; przykładowo przy bazowej częstotliwości środkowej 5 GHz częstotliwość środkowa kanału o numerze 36 wynosi 5 GHz + 36 · 5 MHz = 5,18 GHz. Mimo iż częstotliwości środkowe sąsiednich kanałów różnią

się o 5 MHz, kanał może być *szerszy* niż 5 MHz — specyfikacja 802.11n dopuszcza kanały o szerokości do 40 MHz. Tak więc różne kanały dla tej samej częstotliwości bazowej mogą nakładać się na siebie, czyli transmisja prowadzona przez jeden kanał może interferować z transmisjami prowadzonymi w kanałach sąsiednich.

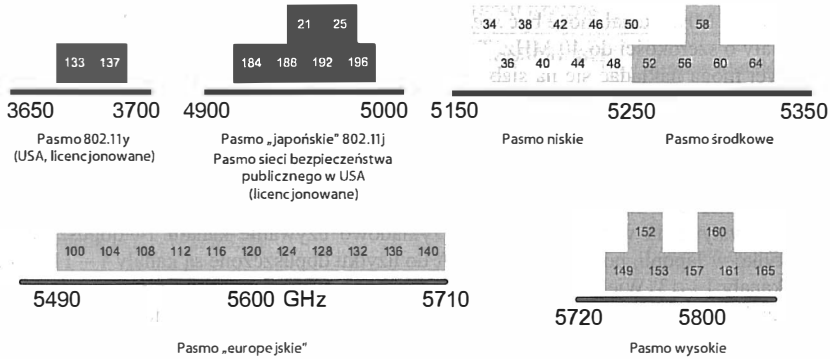
Na rysunku 3.20 przedstawiliśmy schemat odwzorowania kanałów na częstotliwości przy paśmie podstawowym 2,4 GHz (czyli paśmie ISM standardu 802.11b/g). Szerokość każdego kanału wynosi 22 MHz. Nie każdy kanał dostępny jest do legalnego wykorzystywania w każdym kraju, przykładowo używanie kanału 14 dopuszczalne jest tylko w Japonii, podczas gdy w USA do użytku dopuszczone są kanały 1 – 11, w Polsce kanały 1 – 13. Wiele innych krajów jest pod tym względem bardziej restrykcyjnych (patrz aneks J do standardu 802.11 i jego poprawek). Nie zapominajmy jednak, że polityka telekomunikacyjna i związane z nią licencje mogą zmieniać się z upływem czasu.



Rysunek 3.20. *Specyfikacje 802.11b i 802.11g dotyczą częstotliwości z przedziału od 2,4 GHz do ok. 2,5 GHz. Zakres ten podzielony został na 14 nakładających się kanałów o szerokości 22 MHz każdy, których określony podzbiór dopuszczony jest do użytku przez uregulowania prawne danego kraju lub regionu. W USA możliwe jest więc bezkolizyjne funkcjonowanie na tym samym obszarze trzech AP nastrojonych na kanały (odpowiednio) 1., 6. i 11. Tylko jeden kanał o szerokości 40 MHz (wynikającej ze specyfikacji 802.11n) może funkcjonować bez nakładania się na inne kanały*

Na rysunku 3.20 zilustrowano wyraźnie zjawisko nakładania się kanałów: przykładowo transmisja w kanale 1. może oddziaływać na kanały 2., 3., 4. i 5., ale nie wyższe. Nakładanie się kanałów stanowi poważne wyzwanie w sytuacji, gdy na danym obszarze funkcjonować ma kilka punktów dostępowych, być może obsługujących wiele różnych sieci. Praktyką powszechnie przyjętą w USA, gdzie do dyspozycji (bez licencji) mamy 11 kanałów, jest skonfigurowanie trzech punktów dostępowych do pracy w kanałach (odpowiednio) 1., 6. i 11. Gdy na tym samym obszarze funkcjonować ma wiele różnych sieci WLAN, może być konieczne porozumienie się ich administratorów w kwestii wykorzystywania poszczególnych kanałów.

Na rysunku 3.21 widoczna jest cokolwiek bardziej skomplikowana struktura definiowana przez specyfikacje 802.11a/n/y, oferująca jednak większą liczbę nienakładających się kanałów (w USA 12 nielicencjonowanych kanałów o szerokości 20 MHz).



Rysunek 3.21. *Kanały definiowane przez 802.11 i ich częstotliwości środkowe dla szerokości 20 MHz. Jak widać, kanały dopuszczone do używania bez licencji plasują się w paśmie powyżej 5 GHz; dolna część tego pasma zalegalizowana jest w większości krajów, w jego części środkowej znajdują się kanały dozwolone w większości krajów Europy, najwyższe kanały dopuszczone są do użytku w USA i Chinach. Zgodnie ze specyfikacjami 802.11a/y kanały mają szerokość 20 MHz, lecz specyfikacja 40 MHz dopuszcza łączenie sąsiednich kanałów w jeden o szerokości 40 MHz. Na rysunku nie pokazano węższych kanałów oraz niektórych kanałów dopuszczonych do użytku w Japonii*

Kanały na rysunku 3.21 numerowane są według przyrostów 5 MHz, lecz mają zróżnicowaną szerokość — 5 MHz, 10 MHz, 20 MHz i 40 MHz (ostatnia z wymienionych jest wynikiem agregowania dwóch sąsiednich kanałów o szerokości 20 MHz, opisywanego w specyfikacji 802.11n i implementowanego przez niektórych producentów — patrz podpunkt 3.5.4.2).

W typowych sieciach Wi-Fi wybór kanału odbywa się na etapie instalowania punktu dostępowego; stacja kliencka w momencie skojarzenia z punktem dostępowym przełącza się na przypisany mu kanał. W sieciach ad hoc, gdzie nie ma punktów dostępowych, użytkownik stacji ręcznie wybiera kanał, w granicach narzuconych przez możliwości sprzętu, oprogramowanie sterujące i uregulowania prawne.

3.5.4.2. 802.11n — wysoka przepustowość

U schyłku 2009 roku IEEE opublikowało specyfikację 802.11n (patrz [802.11n-2009]) jako aktualizację specyfikacji [802.11-2007]. Specyfikacja ta wprowadza do standardu 802.11 kilka istotnych zmian. Po pierwsze, z myślą o zapewnieniu większej przepustowości zdefiniowano równoległą obsługę strumieni danych, transmitowanych za pomocą wielu anten. Jeden kanał może obsługiwać do czterech takich strumieni — strumienie te nazywane są **strumieniami rozprzestrzenionymi** (*spatial streams*), cały zaś mechanizm ich obsługi oznaczany jest akronimem MIMO, od *multiple input, multiple output* — wielokrotne wejście, wielokrotne wyjście. Po drugie, zdefiniowano mechanizm agregowania dwóch sąsiadujących kanałów, w wyniku czego otrzymuje się kanał o szerokości 40 MHz — dwukrotnie większej niż szerokość konwencjonalnych kanałów 802.11a/b/g/y. Jak łatwo policzyć, połączenie obu tych mechanizmów skutkuje możliwością ośmiokrotnego zwiększenia przepustowości — od standardowej dla 802.11a/g 54 Mb/s do 432 Mb/s. Po trzecie, nawet w przypadku pojedynczego strumienia 802.11n umożliwia to zwiększenie wydajności transmisji przez wykorzystanie schematu modulacji

OFDM (*Orthogonal Frequency Division Multiplexing* — multipleksowanie z ortogonalnym podziałem częstotliwości) pozwalającego na wydzielenie do 52 subnośników w kanale 20 MHz i 108 subnośników w kanale 40 MHz (zamiast 48 w 802.11a/g). Po czwarte, bardziej wydajny kod korekcji błędów forwardowania (ze stosunkiem użytecznym $\frac{5}{6}$ zamiast $\frac{3}{4}$) daje efektywną szybkość transmisji 65 Mb/s w kanale 20 MHz i 135 Mb/s w kanale 40 MHz, a dodatkowo przez skrócenie interwału ochronnego między symbolami (GI — *Guard Interval*) z 800 ns do 400 ns szybkość ta wzrosnąć może do ok. (odpowiednio) 72,2 Mb/s i 150 Mb/s. Przy prawidłowo funkcjonujących czterech strumieniach rozprzestrzenionych daje to szybkość maksymalną prawie 600 Mb/s.

W specyfikacji 802.11n zdefiniowano 77 kombinacji modulacji i kodowania (oznaczanych akronimem MCS, od *Modulation and Coding Scheme*), w tym 8 dla pojedynczego strumienia, 24 dla wielu strumieni wykorzystujących te same parametry (EQM — *EQual Modulation*) i 43 dla zróżnicowanych parametrów na poszczególnych strumieniach (UEQM — *UnEQual Modulation*). W tabeli 3.3 przedstawiono znaczenie wybranych wartości MCS. Pierwsze kombinacje 0 – 32 dotyczą pojedynczego kanału, kombinacje 33 – 38 dwóch, kombinacje 39 – 52 trzech, a kombinacje 53 – 76 czterech kanałów. Kombinacja 32. ma specjalne znaczenie: sygnały w obu połówkach kanału 40 MHz przenoszą tę samą informację. Dla każdej kombinacji podano dwie wartości efektywnej szybkości transmisji, odpowiadające dwu wartościom interwału GI: tradycyjnej 800 ns i obecnej 400 ns. Podkreślone liczby oznaczają najmniejszą (6 Mb/s) i największą (600 Mb/s) szybkość.

W tabeli 3.3 widoczne są różne schematy kodowania: **kluczowanie dwufazowe** (BPSK — *Binary Phase Shift Keying*), **kluczowanie czterofazowe** (QPSK — *Quadrature Phase Shift Keying*) i **kwadraturowa modulacja amplitudowo-fazowa** (QAM — *Quadrature Amplitude Modulation*) w wariancie 4-bitowym (16-QAM) i 6-bitowym (64-QAM). Schematy te zapewniają zwiększoną szybkość transmisji w danym paśmie kanału, im jednak schemat modulacji wydajniejszy i bardziej złożony, tym bardziej podatny na zakłócenia i interferencje. Konieczne jest więc wprowadzenie do transmitowanych danych dodatkowych, redundantnych bitów, umożliwiających wykrywanie przekłamań i ich niwelowanie — jest to zadanie **kodów korygujących błędy forwardowania** (FEC — *Forwarding Error Correction*). Różne kody korekcyjne charakteryzują się odmiennym tzw. **stosunkiem użytecznym** (*code rate*), czyli ilorazem liczby bitów niosących użyteczną informację do ogólnej liczby transmitowanych bitów, przykładowo w kodzie o stosunku użytecznym $\frac{5}{6}$ na każde pięć bitów informacji użytecznej przypada jeden redundantny bit kontrolny, czyli ogółem transmitowanych jest sześć bitów.

Urządzenia standardu 802.11n mogą pracować w jednym z trzech trybów. Charakterystyczny dla tej grupy tzw. **tryb zielony** (*greenfield mode*) charakteryzuje się specyficznym wzorcem bitowym PLCP (nazywanym **ciągiem treningowym** — *training sequence*), nierozpoznawanym przez starsze urządzenia. Współdziałanie z tymi ostatnimi umożliwiają dwa tryby kompatybilności, nieuchronnie pogarszające natywną wydajność sprzętu 802.11n. Pierwszy z tych trybów, zwany *non-HT mode*, oznacza w praktyce pełną kompatybilność wstecz, czyli rezygnację z udogodnień oferowanych przez 802.11n, i jako taki nie jest specjalnie atrakcyjny. Alternatywą dla niego jest „mieszany" tryb *HT-mixed mode*, czyli (mówiąc ogólnie) wsparcie punktu dostępowego (AP) dla obu kategorii

Tabela 3.3. *Wybrane kombinacje MCS zdefiniowane w specyfikacji 802.11n*

MCS	Typ modulacji	Kod korekcyjny	Strumienie rozprzestrzenione	Szybkość (w Mb/s) dla kanału 20 MHz (800 ns/400 ns)	Szybkość (w Mb/s) dla kanału 40 MHz (800 ns/400 ns)
0	BPSK	1/2	1	6,5/7,2	13,5/15
1	QPSK	1/2	1	13/14,4	27/30
2	QPSK	3/4	1	19,5/21,7	40,5/45
3	16-QAM	1/2	1	26/28,9	54/60
4	16-QAM	3/4	1	39/43,3	81/90
5	64-QAM	2/3	1	52/57,8	108/120
6	64-QAM	3/4	1	58,5/65	121,5/135
7	64-QAM	5/6	1	65/72,2	135/150
8	BPSK	1/2	2	13/14,4	27/30
...	...	...	...	...	...
15	64-QAM	5/6	2	130/144,4	270/300
16	BPSK	1/2	3	19,5/21,7	40,5/45
...	...	...	...	...	...
31	64-QAM	5/6	4	260/288,9	540/600
32	BPSK	1/2	1	(nie dotyczy)	6/6,7
...	...	...	...	...	...
76	64×3/ 16×1-QAM	3/4	4	214,5/238,3	445,5/495

operacji (tradycyjnych i 802.11n) zależnie od typu stacji wykonującej skojarzenie z tymże AP. Zachowanie kompatybilności wiąże się z koniecznością wydłużenia nagłówka PLCP, w którym muszą zostać zawarte obie kategorie informacji, a także z koniecznością spowolnienia transmisji, stosownie do możliwości starszego sprzętu. Konieczność spowolnienia transmisji dotyczy także ramek CTS wysyłanych przez AP (i generalnie ruchu RTS/CTS) do starszych urządzeń; mimo niewielkiego rozmiaru samych ramek RTS/CTS konieczność komunikowania ich z klasyczną prędkością 6 Mb/s skutkuje znaczącym spowolnieniem sieci WLAN.

Konfigurując punkty dostępowe 802.11n, należy szczególnie starannie przyporządkowywać kanały. Kanały o szerokości 40 MHz powinny być przyporządkowywane w paśmie U-NII (powyżej 5 GHz) z tej prostej przyczyny, że w paśmie ISM (2,4 GHz) brakuje dla nich wystarczającego spektrum. Opcjonalny mechanizm BSS-u, zwany **współistnieniem cyklicznych operacji** (*Phased Coexistence Operations* — PCO), polegający na cyklicznym przełączaniu się punktu dostępowego między szerokościami kanałów 40 MHz i 20 MHz, ułatwia współegzystencję sprzętu 802.11n i starszego, ale za cenę zapotrzebowania na dodatkową przepustowość.

Kolejną specyficzną cechą urządzeń 802.11n, wynikającą bezpośrednio z ich nowych funkcji, jest zwiększone zapotrzebowanie na moc zasilającą. Zapotrzebowanie to przekracza granicę 15 W wyznaczoną przez standard 802.3 af (*Power-over-Ethernet*, w skrócie PoE) dla zasilania urządzeń Ethernetu bezpośrednio przez skrętkę. Oznacza to konieczność stosowania okablowania zgodnego ze standardem PoE+ (802.3at) wyznaczającym maksymalną moc 30 W, chyba że energia zasilająca dostarczana będzie urządzeniom za pomocą niezależnych zasilaczy.

3.5.5. Bezpieczeństwo Wi-Fi

Model bezpieczeństwa sieci bezprzewodowych, poza oczywistym aspektem praktycznym, interesujący jest sam w sobie także ze względu na ewolucję, jaką przeszedł od zarania standardu 802.11.

Początkowo bezpieczeństwo sieci 802.11 zapewnić miał protokół WEP — to skrót od *Wired-Equivalent Privacy*, dosł. „prywatność równoważna [osiągalnej w] sieci przewodowej". Gdy po pewnym czasie okazało się, że nie jest wystarczająco silny i konieczne jest stworzenie bardziej solidnego następnika, odpowiedzią ze strony przemysłu IT stał się protokół WPA (*Wi-Fi Protected Access*) zmieniający sposób używania kluczy w stosunku do szyfrowanych bloków (podstawy kryptografii przedstawiamy w rozdziale 18.)[1]: schemat o nazwie TKIP (*Temporal Key Integrity Protocol* — protokół integralności kluczy tymczasowych) wymuszał okresową zmianę kluczy szyfrowania, a ponadto wprowadzał ochronę integralności przesyłanych komunikatów za pomocą algorytmu o nazwie Michael. Co ciekawe, urządzenia używające dotychczas szyfrowania WEP można było łatwo przystosowywać do WPA, przeprogramowując ich firmware. Niedługo potem grupa robocza IEEE 802.11i przystąpiła do opracowywania kolejnego, jeszcze silniejszego protokołu ochronnego, którego finalna wersja stałą się przedmiotem klauzuli numer 8 standardu [802.11-2007] i zyskała przemysłową nazwę WPA2. Podczas gdy WEP i WPA oparte były na szyfrowaniu RC4 (patrz [S96]), w WPA2 zastosowano znacznie silniejszy algorytm AES (*Advanced Encryption Standard*, patrz [AES01]).

Wymienione algorytmy służą do ochrony prywatności informacji wymienianej między stacją a punktem dostępowym, przy założeniu że wspomniana stacja jest autoryzowana dla dostępu do sieci. W sieci wykorzystującej WEP oraz niewielkich środowiskach stosujących WPA i WPA2 autoryzację tę realizuje się, umieszczając a priori ustalony klucz (zwany **kluczem wstępnie współdzielonym** — *pre-shared key*, w skrócie PSK) w każdej stacji i samym punkcie dostępowym. Klucz ten, znany wyłącznie uprawnionym użytkownikom, wykorzystywany bywa również jako punkt startowy do generowania tymczasowych kluczy szyfrowania. Nietrudno zauważyć, że scenariusz ten skaluje się raczej kiepsko: wykluczenie użytkownika z grona uprawnionych jest dość kłopotliwe dla administratora, bo musi wówczas wymienić wspomniany klucz współdzielony i podać jego nową postać wszystkim (aktualnie) uprawnionym użytkownikom. W dużych środowiskach — zwłaszcza tych, gdzie zespół uprawnionych użytkowników podlega

[1] Czytelnikom zainteresowanym szczegółami technik kryptograficznych i ich zastosowań polecamy dwutomową monografię *Kryptografia i bezpieczeństwo sieci komputerowych*, wyd. Helion w latach 2011 i 2012. W pierwszym tomie opisywane są szczegóły algorytmów kryptograficznych i ich realizacji, natomiast rozdział 4. drugiego tomu poświęcony jest bezpieczeństwu sieci bezprzewodowych — *przyp. tłum.*

naturalnej rotacji — może stać się to istnym koszmarem. W związku z tym, WPA i późniejsze standardy definiują **kontrolę dostępu bazującą na portach** (*port-based network access control*) opisaną w specyfikacji 802.1x (patrz [802.1X-2010]. Metoda ta opiera się na protokole EAP (*Extensible Authentication Protocol* — elastyczny protokół uwierzytelniania) opisanym w dokumencie [RFC3748], przeniesionym na grunt sieci lokalnych standardów 802.3 i 802.11 (patrz [RFC4017]) i stąd opatrzonym akronimem EAPOL (od *EAP over LAN*). Protokół EAP może być wykorzystywany jako podstawa realizacji wielu innych (standardowych i niestandardowych) protokołów uwierzytelniania, a także do generowania kluczy, w tym kluczy WEP. Szczegółowy opis protokołu EAP odkładamy do rozdziału 18., w międzyczasie jednak powrócimy do niego w kontekście omawiania PPP w podrozdziale 3.6.

W rezultacie prac grupy roboczej 802.11i obowiązująca w ramach WEP kombinacja RC4/TKIP rozszerzona została o nowy algorytm szyfrowania o nazwie CCMP, stanowiący integralny element WPA2. Algorytm ten bazuje na **trybie licznikowym** (*counter mode*) szyfrowania AES (patrz [RFC3610]) w celu zapewnienia poufności danych oraz na **kodzie uwierzytelniającym w trybie łańcuchowania bloków szyfrogramu** (CBC-MAC)[2]. Szyfrowanie AES odbywa się przez podział komunikatu na 128-bitowe bloki, przy użyciu 128-bitowego klucza. CCMP i TKIP to dwa fundamenty architektury bezpieczeństwa Wi-Fi, zwanej **solidnym zabezpieczeniem dostępu do sieci** (*Robust Security Network Access*, w skrócie RSNA); wcześniejsze metody, takie jak WEP, określane są często mianem **metod preRSNA**. Zgodność sprzętu i oprogramowania z RSNA wymaga implementacji CCMP, implementacja TKIP jest opcjonalna — zresztą specyfikacja 802.11n usuwa TKIP całkowicie w cień. Tabela 3.4 stanowi próbę rozjaśnienia tej cokolwiek skomplikowanej sytuacji.

Tabela 3.4. *Ewolucja zabezpieczeń sieci bezprzewodowych*

Standard	Szyfr	Zarządzanie strumieniem kluczy	Uwierzytelnianie
WEP (preRSNA)	RC4	(WEP)	PSK, (802.1X/EAP)
WPA	RC4	TKIP	PSK, 802.1X/EAP
WPA2/802.11(i)	CCMP	CCMP, (TKIP)	PSK, 802.1X/EAP

Jak widać, w każdym przypadku zarówno klucze wstępnie współdzielone (PSK), jak i mechanizmy 802.1X mogą być wykorzystywane do uwierzytelniania i podczas generowania kluczy tymczasowych. 802.1X ma jednak tę zaletę, że dostęp do AP może być kontrolowany na poziomie poszczególnych użytkowników, a to za sprawą odpowiednio skonfigurowanego serwera uwierzytelniającego; z tego względu mechanizmom 802.1X często towarzyszy przymiotnik „Enterprise" (vide *WPA-Enterprise*). Sam protokół EAP może służyć do hermetyzowania różnych innych protokołów uwierzytelniania —do tego tematu powrócimy w rozdziale 18.

[2] Niefortunnie akronim MAC występuje w literaturze przedmiotu w dwóch znaczeniach, niemających ze sobą wiele wspólnego: w kontekście łącza danych oznacza *Media Access Control*, czyli „sterowanie dostępem do nośnika", zaś w kontekście kryptograficznym jest skrótem od *Message Authentication Code* — „kod uwierzytelniania komunikatu". Często w celu uniknięcia nieporozumień w tym względzie określenie „kod uwierzytelniania komunikatu" zastępuje się synonimicznym określeniem „kod integralności komunikatu" i oznacza akronimem MIC, od *Message Integrity Code — przyp. tłum.*

3.5.6. 802.11s — sieci kratowe Wi-Fi

IEEE pracuje obecnie nad standardem 802.11s definiującym **operacje kratowe** (*mesh operations*) w sieciach Wi-Fi. Istotą takich sieci jest możliwość komunikowania się urządzeń bez angażowania centralnych węzłów w postaci punktów dostępowych, czyli bądź to bezpośrednio, bądź za pośrednictwem innych stacji. Obecnie (wiosna 2012 roku) standard ten nie został jeszcze zatwierdzony. W aktualnej postaci definiuje on protokół o nazwie **hybrydowy protokół kraty bezprzewodowej** (HWMP —*Hybrid Wireless Mesh Protocol* —HWRP), oparty po części na protokole AODV (*Ad-Hoc On-Demand Distance Vector* — patrz [RFC3561]), po części zaś na protokole OLSR (*Optimized Link State Routing* — patrz [RFC3626]); oba protokoły stanowią standardy IETF. Stacje sieci kratowej (zwane **stacjami kratowymi** — *mesh stastions*, w skrócie *mesh STA*) są odmianą stacji QoS i mogą wykorzystywać HWMP lub inny protokół trasowania, lecz warunkiem zgodności węzła ze standardem 802.11s jest implementowanie HWRP i związanej z nim **metryki łączy radiowych** (*airtime link metric*). Węzły sieci mogą współdziałać ze sobą w oparciu o mechanizm EDCA lub za pomocą opcjonalnej funkcji koordynującej zwanej **deterministycznym dostępem do kraty** (*mesh deterministic access*). **Punktami kratowymi** (*Mesh Points*, w skrócie MP) są węzły organizujące innym stacjom łączność z węzłami sąsiednimi. Punkty kratowe, implementujące również funkcjonalność punktu dostępowego, nazywane są **kratowymi punktami dostępowymi** (*mesh AP*, w skrócie MAP). Konwencjonalne stacje standardu 802.11 mogą korzystać z usług zarówno klasycznych AP, jak i MAP w celu uzyskania dostępu do innych węzłów w sieci WLAN.

Szkic standardu 802.11s definiuje także nową opcjonalną formę zabezpieczeń dla RSNA, zwaną **równoczesnym równoprawnym uwierzytelnianiem** (*Simultaneous Authentication of Equals*, w skrócie SAE — patrz [SAE]). Jest to protokół o tyle odmienny od tradycyjnego protokołu uwierzytelniania, że nie opiera się na rutynowej wymianie komunikatów między specjalnie wyznaczonymi inicjatorem i responderem, lecz na równorzędnym traktowaniu wszystkich stacji: dowolna stacja, rozpoznająca inną, może sama zainicjować dialog uwierzytelniający (mogą go również zainicjować równocześnie obie stacje wchodzące w skojarzenie).

3.6. Protokół „punkt-punkt" (PPP)

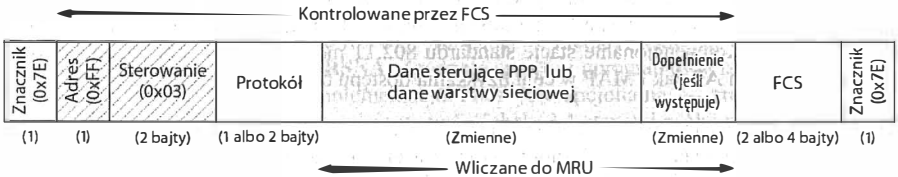
Protokół „punkt-punkt", oznaczamy powszechnie akronimem PPP (od *Point-to-Point Protocol* — patrz [RFC1661] [RFC1662] [RFC2153]), jest popularnym środkiem transmisji datagramów IP przez łącza szeregowe różnego rodzaju — od modemów „dodzwaniających" do szybkich łączy światłowodowych ([RFC2615]). Stosowany jest masowo przez wielu dostawców usługi DSL, którzy wykorzystują go również do konfigurowania parametrów internetowych (adresu IP i serwera DNS — patrz rozdział 6.).

PPP to nie pojedynczy protokół, lecz raczej kolekcja protokołów. Obejmuje ona m.in. **protokół sterowania łączem** (*Link Control Protocol*, w skrócie LCP) oraz rodzinę protokołów NCP, wykorzystywanych do ustanawiania łączy warstwy sieciowej dla IPv4, IPv6 oraz wielu innych protokołów spoza grupy IP. Całości dopełniają powiązane protokoły dotyczące kompresji i szyfrowania transmisji PPP oraz różnych metod uwierzytelniania.

3.6.1. Protokół sterowania łączem (LCP)

Protokół LCP służy do nawiązywania i utrzymywania niskopoziomowego połączenia między uczestnikami komunikacji w postaci łącza dwupunktowego. Operacje PPP mogą się więc koncentrować jedynie na „końcówkach" łącza, nie występuje problem negocjowania dostępu do współdzielonego nośnika, jak to miało miejsce w Ethernecie i Wi-Fi.

PPP ogólnie — a LCP szczególnie — stawiają łączu stosunkowo małe wymagania: musi ono obsługiwać transmisję dwukierunkową (LCP wykorzystuje mechanizm potwierdzania) i funkcjonować w trybie synchronicznym lub asynchronicznym. LCP realizuje nawiązywanie połączenia przy użyciu ramki w prostym formacie bitowym bazującym na protokole HDLC (*High-Level Data Link Control* — sterowanie łączem danym wysokiego poziomu), dobrze sprawdzonym i wykorzystywanym w czasie projektowania PPP (patrz [ISO3309] i [ISO4335]). Modyfikując protokół HDLC, firma IBM stworzyła na potrzeby swej architektury SNA jego odmianę o nazwie SDLC (*Synchronous Data Link Control* — sterowanie synchronicznym łączem danych); HDLC posłużył też jako wzorzec dla standardu sterowania łączem logicznym (LLC) w standardzie 802.2, a w konsekwencji dla PPP. Format jego ramki przedstawiony jest na rysunku 3.22.



Rysunek 3.22. *Podstawowy format ramki PPP, zapożyczony z protokołu HDLC. Obejmuje identyfikator protokołu, obszar ładunku użytecznego i sumę kontrolną (FCS) o rozmiarze 2 lub 4 bajtów. Pozostałe pola mogą mieć znaczenie w przypadku zastosowania kompresji*

Jak łatwo zauważyć, początek i koniec ramki sygnalizowany jest przez 1-bajtowy znacznik o ustalonej wartości 0x7E — w ten sposób granice ramki rozpoznawane są przez komunikujące się urządzenia. Oczywiście, konwencja ta stwarza pewien problem, w sytuacji gdy bajt o wartości 0x7E występuje gdziekolwiek wewnątrz ramki; przed jej wysłaniem należy poddać ją transformacji likwidującej takie wystąpienia, a po odebraniu (gdy znaczniki 0x7E zostaną już rozpoznane) — transformacji odwrotnej przywracającej oryginalną postać.

W trybie asynchronicznym problem ten rozwiązywany jest przez kodowanie każdego napotkanego bajta 0x7E do 2-bajtowej sekwencji 0x7D5E; bajt o wartości 0x7D jest „znakiem unikowym" (*escape character*) dla PPP, jeżeli więc występuje gdziekolwiek wewnątrz ramki, kodowany jest do postaci 0x7D5D. Urządzenie odbiorcze, otrzymując tak zakodowaną ramkę, poszukuje w niej 2-bajtowych sekwencji 0x7D5E i 0x7D5D i zamienia każdą z nich na pojedynczy bajt, o wartości (odpowiednio) 0x7E i 0x7D. Ponieważ operacja ta zwiększa strumień transmisyjny o kompletne bajty, nazywana jest potocznie „faszerowaniem bajtami" (*byte stuffing*).

W łączach synchronicznych (np. T1 lub T3) PPP radzi sobie z opisanym problemem w sposób bardziej oszczędny, wykonując na poziomie pojedynczych bitów analogiczną transformację, czyli „faszerowanie bitami" (*bit stuffing*). Zwróćmy uwagę, że binarna

postać znacznika 0x7E zawiera sześć sąsiadujących ze sobą bitów jedynkowych (01111110); przed wysłaniem ramki każdy napotkany ciąg pięciu bitów jedynkowych kodowany jest do postaci sześciobitowego ciągu 111110, w rezultacie czego każdy napotkany znacznik (czyli ciąg 01111110) konwertowany zostaje do ciągu 011111010. Urządzenie odbiorcze poszukuje w otrzymanej ramce ciągów bitowych 111110 i każdy z nich zamienia na ciąg 11111 (odrzucając po prostu końcowy zerowy bit)[3].

Po początkowym znaczniku napotykamy pola znane z protokołu HDLC. Pole *Adres* w oryginalnej wersji służyło do identyfikowania konkretnej stacji docelowej; ponieważ w przypadku PPP jest tylko jedna stacja docelowa, w pole to wpisywana jest wartość 0xFF (zapewne jako analogia do adresu broadcast, reprezentującego wszystkie stacje). Zadaniem pola *Sterowanie* była oryginalnie obsługa retransmisji ramek i związanego z nim ich sekwencjonowania; ponieważ funkcje tej kategorii nie są zwykle implementowane jako część protokołu PPP, w polu tym znajduje się ustalona wartość 0x03. Jako że wspomniane pola mają ustaloną a priori zawartość, ich transmitowanie oznacza po prostu marnotrawienie trzech bajtów na każdą przesyłaną ramkę, dlatego często są one eliminowane z transmitowanego strumienia za pomocą opcji zwanej *Address and Control Field Compression* (ACFC).

Od wielu lat toczą się nieustanne dyskusje w kwestii, czy — i w jakim zakresie — protokoły warstwy łącza danych powinny realizować mechanizmy zwiększające niezawodność sieci, a szczególnie funkcje retransmisji. Przykładowo Ethernet, jeśli trzeba, ponawia próbę retransmisji 16-krotnie, zanim ostatecznie da za wygraną, natomiast PPP z reguły pozbawiony jest mechanizmu retransmisji w ogóle, chociaż istnieją specyfikacje wzbogacające go o tę funkcję (m.in. [RFC1663]). Wydaje się, że jest to kwestia subtelna, zależna głównie od charakteru (i stopnia krytyczności) ruchu przenoszonego przez warstwę łącza danych. Czytelnikom zainteresowanym wspomnianą dyskusją proponujemy lekturę dokumentu [RFC3366]).

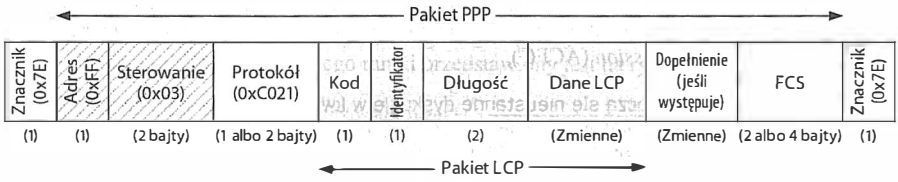
W polu *Protokół* znajduje się identyfikator protokołu warstwy wyższej, którego dane przenoszone są w ramkach PPP. Oficjalna lista identyfikatorów protokołów obsługiwanych przez PPP znajduje się na stronie [PPPn]. Zgodnie ze specyfikacją HDLC, identyfikator protokołu jest dwubajtowym słowem, w którym bardziej znaczący bajt ma wartość parzystą, a mniej znaczący — nieparzystą (każdy identyfikator ma więc postać bitową *xxxx xxx0 yyyy yyy1*). Identyfikatory z przedziału od 0x0000 do 0x3FFF przyporządkowane są protokołom warstwy sieciowej, wartości z przedziału od 0x8000 do 0x8FFF identyfikują dane należące do skojarzonych protokołów NCP, identyfikatory z przedziału od 0x4000 do 0x7FFF używane są przy „niskoprzepustowych" protokołach bez skojarzonych NCP, natomiast identyfikatory z przedziału od 0xC000 do 0xEFFF zarezerwowane są dla protokołów sterujących, takich jak LCP. Jeśli bardziej znaczący bajt identyfikatora jest zerowy (co ma miejsce w protokołach warstwy sieciowej), można zrezygnować z jego przesyłania, o ile urządzenia nawiązujące połączenie wynegocjują opcję PFC — *Protocol Field Compression*). Zauważmy, że kompresja taka nie dotyczy protokołów sterujących, szczególnie LCP, bo ich identyfikatory mają bardziej znaczący bajt o wartości 0xC0 lub większej.

[3] Rzecz jasna bajt o wartości 0x7E nie jest jedynym zawierającym sąsiadującą grupę pięciu jedynek, własność tę ma jeszcze mnóstwo innych bajtów; wspomniana grupa może ponadto pojawić się *na styku* dwóch bajtów. Nie przeszkadza to nijak w realizacji celu, jakim jest wyeliminowanie możliwości pojawienia się bajta 0x7E w przesyłanym strumieniu, nie ma też znaczenia z punktu widzenia jednoznaczności przesyłanej informacji, bo każdy ciąg sąsiadujących pięciu jedynek jest poprawnie kodowany i dekodowany, niezależnie od pochodzenia — *przyp. tłum.*

Ostatnim elementem ramki jest suma kontrolna FCS, mająca postać kodu CRC16 obliczanego przy użyciu generatora 10001000000100001 dla całej ramki, z wyjątkiem znaczników granicznych oraz — oczywiście — pola FCS. Należy w tym miejsc zaznaczyć, że obliczanie tego kodu odbywa się na podstawie oryginalnej zawartości ramki, przed wykonaniem opisanych wcześniej operacji „faszerowania" bajtowego lub bitowego. Przy użyciu odpowiedniej opcji LCP (patrz podpunkt 3.6.1.2) pole FCS może zostać rozszerzone do 4 bajtów, jego zawartość jest wówczas kodem CRC32 obliczanym tak samo jak dla ramki ethernetowej, czyli przy użyciu 33-bitowego generatora 100000100110000010001110110110111 (patrz podpunkt 3.2.2.1).

3.6.1.1. Operacje protokołu LCP

Protokół LCP wykorzystuje prostą hermetyzację w pakiecie PPP, co pokazujemy na rysunku 3.23.



Rysunek 3.23. *Format pakietu LCP obejmujący m.in. identyfikację enkapsulowanych danych i ich długość. Ramki LCP używane są przede wszystkim w procesie ustanawiania połączenia PPP, lecz ich format wykorzystywany jest także w wielu innych protokołach sterowania siecią*

W polu *Protokół* znajduje się wartość 0xC021 identyfikująca protokół LCP; jej bardziej znaczący bajt ma niezerową wartość, nie można więc skompresować pola za pomocą opcji PFC — dzięki temu minimalne jest prawdopodobieństwo powstania niejednoznaczności. Pole *Identyfikator* zawiera numer sekwencyjny nadany ramce przez nadawcę — nadawca nadaje kolejne numery kolejno wysyłanym ramkom żądań, a gdy nadchodzi ramka odpowiedzi na żądanie (ACK, NACK lub REJECT — o tym za chwilę), zawiera w polu *Identyfikator* numer odnośnego żądania. Właśnie zawartość pola *Identyfikator* umożliwia stronie wysyłającej żądania kojarzenie ich z odpowiedziami. W polu *Kod* znajduje się kod żądania lub odpowiedzi, oznaczający szczegółowo:

- żądanie konfiguracyjne (configure-request — 0x01),
- potwierdzenie żądania konfiguracyjnego (configure-ACK — 0x02),
- zakwestionowanie żądania konfiguracyjnego (configure-NACK — 0x03),
- odrzucenie żądania konfiguracyjnego (configure-REJECT — 0x04),
- żądanie zakończenia (terminate-request — 0x05),
- potwierdzenie zakończenia (terminate-ACK— 0x06),
- błędną postać pola *Kod* (code-REJECT— 0x07),
- błędną postać pola *Protokół* (protocol-REJECT — 0x08),
- żądanie ECHO (echo-request — 0x09),

- odpowiedź na żądanie ECHO (`echo-reply` — 0x0A),
- żądanie odrzucenia (`discard-request` — 0x0B),
- identyfikację (`identification` — 0x0C),
- pozostały czas sesji (`time-remaining` — 0x0D).

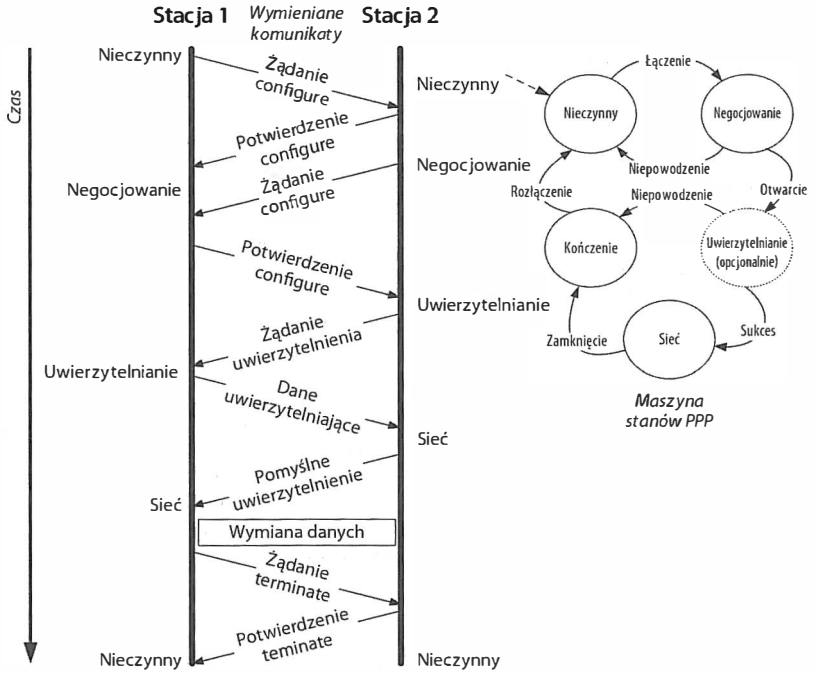
Ogólnie rzecz biorąc, odpowiedź `ACK` oznacza zaakceptowanie zbioru opcji, odpowiedź `NACK` — zakwestionowanie niektórych opcji, zwykle ze wskazaniem alternatywy, a odpowiedź `REJECT` — bezwzględne odrzucenie jednej lub kilku opcji. Pole *Długość* określa rozmiar pakietu w bajtach; rozmiar ten nie może przekraczać maksymalnej wartości, oznaczanej MRU (*Maximum Received Unit*) — jej znaczeniem zajmiemy się za chwilę. Zauważmy, że pole *Długość* jest częścią pakietu LCP, ogólnie protokół PPP nie korzysta z podobnego pola.

Podstawowym zadaniem protokołu LCP jest ustanowienie łącza PPP w podstawowym zakresie. Komunikaty konfiguracyjne (`configure`) wymieniane między urządzeniami powodują zapoczątkowanie przez nie podstawowej procedury konfigurującej, w tym negocjowania opcji połączenia. Komunikaty zakończeniowe (`terminate`) powodują zwolnienie łącza. Komunikaty `echo-request` i `echo-reply` mogą być wymieniane między stacjami w dowolnym czasie — stacja przekonuje się w ten sposób o żywotności partnera. Komunikat `discard request` stanowi informację dla partnera, że wysłany wcześniej pakiet należy odrzucić bez potwierdzenia — postępowanie takie często dyktowane jest względami efektywności. Komunikaty `identification` i `time-remaining` mają znaczenie administracyjne: ich zadaniem jest rozpoznanie typu systemu partnerskiego oraz uzyskanie informacji o wielkości czasu, przez jaki łącze może jeszcze pozostawać aktywne.

Od dawna jednym z podstawowych problemów związanych z łączami punkt-punkt było funkcjonowanie zdalnej stacji w **trybie zapętlenia** (*loopback mode*). Kompanie telefoniczne czasami wykonują doraźne zapętlenie linii w celach testowych; dane wysłane na jednym końcu zapętlonej linii wracają natychmiast na drugim. Mimo iż stanowi to dobry test na drożność linii, dla komunikacji ma skutki fatalne. W związku z tym, protokół LCP posiada funkcję wysyłania komunikatu z „magiczną liczbą" (*magic number*), czyli liczbą całkowitą wybraną arbitralnie przez nadawcę; w przypadku zapętlenia łącza komunikat ten powinien natychmiast powrócić do nadawcy (co jest sygnałem, że linia telefoniczna wymaga interwencji technicznej).

Dla lepszego wyobrażenia, jak przebiega proces nawiązywania połączenia i negocjowania opcji w protokole PPP, przedstawiliśmy na rysunku 3.24 (uproszczony nieco) wykres czasowy wymiany pakietów oraz uproszczoną maszynę stanów (implementowaną w obu stacjach końcowych).

Łącze uważane jest za ustanowione, gdy tylko warstwa łącza danych wykryje aktywne skojarzenie między stacjami (np. sygnał nośny transmisji modemowej). Na tym etapie mogą być też wymieniane komunikaty i potwierdzenia związane z testowaniem jakości łącza (piszemy o nich w następnym podpunkcie). Jeśli wymagane jest uwierzytelnianie (co jest praktyką typową np. przy „dodzwanianiu" do serwera dostawcy), wymieniane są dodatkowe komunikaty mające na celu weryfikację tożsamości jednej lub obu stacji. Łącze zostaje zwolnione bądź to wskutek zewnętrznej przyczyny (np. uszkodzenia nośnika) stwierdzonej przez procedury protokołu lub detektory sprzętowe, bądź na wyraźne żądanie jednej ze stacji (kwitowane potwierdzeniem ze strony partnera).



Rysunek 3.24. *Protokół LCP wykorzystywany jest do ustanawiania łącza PPP i negocjowania opcji między połączonymi stacjami. Typowa wymiana związanych z tym komunikatów obejmuje parę żądań konfiguracyjnych i ich potwierdzeń zawierających listę opcji, wymianę komunikatów i danych uwierzytelniających i wymianę komunikatów wymuszających zakończenie połączenia. Ponieważ PPP jest protokołem ogólnego przeznaczenia, jego funkcjonowanie obejmuje zwykle wiele innych komunikatów nieuwzględnionych na rysunku*

3.6.1.2. Opcje LCP

Podczas ustanawiania połączenia przez protokół LCP możliwe jest negocjowanie kilku *opcji* na użytek wykorzystywanych protokołów sterujących (NCP); w tym podpunkcie ograniczymy się do omówienia najczęściej używanych.

Pierwsza z nich to **mapowanie znaków sterujących dla transmisji asynchronicznej** (*Asynchronous Control Character Map*, w skrócie ACCM lub *asyncmap*), czyli określenie sposobu reprezentowania znaków sterujących (bajtów o kodach 0x00 – 0x1F) w transmitowanym strumieniu. Często znaki takie traktowane są w sposób specjalny przez sprzęt i oprogramowanie: jeśli np. włączone jest programowe sterowanie przepływem w oparciu o znaki XON/XOFF, napotkanie znaku XOFF (kod 0x13) spowoduje zawieszenie wysyłania danych przez stację, aż do odebrania znaku XON (kod 0x11). Jeżeli więc znaki takie są częścią treści komunikatu, czyli nie występują w roli sterującej, konieczne jest ich odpowiednie zakodowanie. Kodowanie znaku o kodzie *c* odbywa się tu jako przekształcenie tegoż znaku do dwubajtowej sekwencji 0x7D<*d*>, gdzie <*d*> jest bajtem stanowiącym wynik operacji c XOR 0x20; przykładowo wspomniany znak o kodzie

0x13 zakodowany zostanie do postaci 0x7D33 (jak wcześniej wyjaśnialiśmy, bajt 0x7D pełni w protokole PPP funkcję „znaku unikowego"). Opisywana opcja polega na selektywnym określeniu, *które* znaki spośród 32 znaków sterujących podlegać mają takiemu kodowaniu; realizuje się to przez odpowiednie ustawienie poszczególnych bitów 32-bitowej maski — tak więc maska 0xFFFFFFFF oznacza kodowanie wszystkich znaków sterujących, natomiast maska 0x00000000 wyraża całkowitą rezygnację z opisanego mapowania. Gdybyśmy chcieli ograniczyć się do kodowania wyłącznie znaków XON (kod dziesiętny 17) i XOFF (kod dziesiętny 19), musielibyśmy użyć maski 0x000A0000. Mimo iż maska 0xFFFFFFFF jest ustawieniem domyślnym, obecnie wiele łączy PPP działa niezawodnie także przy masce 0x00000000.

Ponieważ ramka PPP nie zawiera pola *Długość*, a łącza szeregowe nie realizują zwykle formowania bitów w ramki, teoretycznie nie istnieje górne ograniczenie rozmiaru ramki PPP. W praktyce ograniczenie to ustanawiane jest explicite, w postaci parametru MRU (opcja 0x01); gdy jedna ze stacji specyfikuje taką opcję, druga stacja zobowiązana jest do nieprzekraczania rozmiaru wysyłanych ramek ponad określony limit wynikający ze wspomnianego parametru. Limit ten dotyczy tylko pola danych, bez uwzględniania innych pól (znaczników granicznych oraz pól *Protokół*, *Adres*, *Sterowanie* i *FCS*). Zwykle stosuje się w tej roli wielkości 1500 bajtów lub 1492 bajtów, lecz teoretycznym ograniczeniem jest 65 535 (tyle można maksymalnie zapisać na 16 bitach). Protokół IPv6 wymaga ramek nie mniejszych niż 1280 bajtów, z kolei standard wymaga od implementacji PPP zdolności do akceptowania ramek 1500-bajtowych, tak więc parametr MRU pełni raczej rolę doradczą dla partnera w kwestii doboru rozmiaru ramek, niż faktycznie wyznacza jakieś ograniczenie. Przy dużej wartości tego parametru przeplatanie się dużych i małych pakietów na tym samym łączu skutkować będzie pochłanianiem dużej porcji pasma przez duże pakiety kosztem małych, ze zgubnym skutkiem dla aplikacji multimedialnych, m.in. VoIP i zdalnego logowania (duża wartość parametru *jitter* określającego wariancję opóźnień ramek); użycie mniejszego MRU może zniwelować to zagrożenie, kosztem jednak większego narzutu na ogół przesyłanych ramek.

Protokół PPP obsługuje mechanizm wymiany przez stacje raportów dotyczących jakości łącza (*link quality*). W fazie negocjowania opcji stacja może zażądać od partnera używania tej roli konkretnego protokołu: jego identyfikator umieszcza się w polu *Protokół* — zwykle jednak jest to standardowy dla PPP protokół LQR (*Link Quality Reports* — patrz [RFC1989]) identyfikowany wartością 0xC025. Żądanie obejmuje także minimalną częstotliwość raportowania — komunikaty LQR powinny być wysyłane nie rzadziej niż co ustalony interwał, określony przez 32-bitową liczbę całkowitą wyrażającą czas w jednostkach $\frac{1}{100}$ sekundy. Komunikat LQR obejmuje „magiczną liczbę", liczbę pakietów wysłanych i odebranych, liczbę odebranych błędnych pakietów, liczbę pakietów odrzuconych i ogólną liczbę wszystkich wymienionych dotąd komunikatów LQR. Wspomniany interwał czasowy jest zazwyczaj parametrem konfigurowalnym. Niektóre implementacje PPP oferują też możliwość automatycznego zakończenia połączenia w sytuacji, gdy historia raportów o jakości łącza plasuje tę jakość poniżej pewnego wymaganego progu. Każdy komunikat LQR zawiera numer sekwencyjny, możliwe jest więc obserwowanie pewnych trendów w zachowaniu łącza nawet wówczas, gdy komunikaty te nadchodzą w kolejności innej niż ta, w jakiej są wysyłane.

Wiele implementacji PPP oferuje funkcję **oddzwaniania** (*callback*): klient dodzwania się do serwera dostawcy, uwierzytelnia i rozłącza, a za chwilę do akcji wkracza **protokół sterowania oddzwanianiem** (*Callback Control Protocol* — CBCP): serwer oddzwania na modem klienta i w ten sposób nawiązuje połączenie. Odciąża to klienta od opłaty za połączenie (ponoszący ten koszt właściciel oddzwaniającego serwera korzysta zwykle z pewnych preferencji taryfowych, a często sam jest operatorem telefonii), ale może też być podyktowane względami bezpieczeństwa, ponieważ wyklucza anonimowość klienta. Protokół wykorzystywany do realizacji oddzwaniania zwrotnego jest opcją LCP o numerze 0x0D (patrz [RFC1570]).

Niektóre algorytmy kompresji i szyfrowania używane przez PPP wymagają porcji danych o pewnym ustalonym rozmiarze, stanowiącym wielokrotność **rozmiaru bloku** (*block size*). Zbyt krótkie dane muszą więc zostać sztucznie uzupełnione do żądanej długości; ciąg dopełniających bajtów lokowany jest bezpośrednio po polu danych, przed polem *FCS*. Dopełnienie konstruowane jest w ten sposób, że każdy jego bajt równy jest własnemu indeksowi w ramach tegoż dopełnienia: pierwszy bajt ma wartość 1, drugi wartość 2 itd.; wartość ostatniego bajta dopełnienia — tego bezpośrednio poprzedzającego pole *FCS* — równa jest więc długości dopełnienia (nie większej niż 255, bo tyle można zapisać w bajcie). Dopełnienie ma charakter *samodokumentujący*, dlatego w dokumencie [RFC1570] nazywane jest *self-describing padding*. Ustawienie odnośnej opcji (typ 10) jest dla partnera sygnalizacją, że stacja może interpretować opisane dopełnienia. Maksymalna długość dopełnienia dopuszczalna dla danego łącza jest wartością parametru MPV (*Maximum Pad Value*). Ponieważ ramka PPP nie zawiera jawnego wskazania oryginalnej długości danych, dopełnienie musi być prawidłowo rozpoznane i wyeliminowane przez stację odbierającą.

W celu zmniejszenia narzutu wynikającego z transmitowania nagłówków ramek opracowano mechanizm multipleksowania w jednej ramce PPP wielu odmiennych ładunków użytecznych, pochodzących od potencjalnie różnych protokołów. Mechanizm ten, znany pod nazwą *PPPMux*, opisany jest w dokumencie [RFC3153]. Pole *Protokół* takiej multipleksowanej ramki ma wartość 0x0059, po nim następuje ciąg bloków stanowiących poszczególne ładunki użyteczne. Każdy blok poprzedzony jest subnagłówkiem o rozmiarze od 1 do 4 bajtów, rozmiar ten wynika z dwóch bitów znajdujących się w pierwszym bajcie: pierwszy z tych bitów, zwany PFF, informuje o obecności (1) albo nieobecności (0) pola *Protokół*, drugi natomiast, zwany LXT, ustawiony jest na 1, gdy pole *Długość* w bloku jest dwubajtowe, i wyzerowany, gdy jest ono jednobajtowe. Jeżeli w subnagłówku występuje pole *Protokół*, jego znaczenie jest dokładnie takie samo jak w zwykłej ramce PPP, identyczne są też zasady jego kompresowania. Jeżeli w subnagłówku pole to nie występuje, przyjmuje się dla bloku protokół domyślny, określony na etapie konfigurowania łącza za pomocą protokołu *PPPMux Control Protocol* (*PPPMuxCP*).

Zgodnie z rysunkiem 3.19, pole *FCS* ramki PPP (wzorowane na protokole HDLC) domyślnie jest 16-bitowe, jednak może być rozszerzone do 32 bitów za pomocą odpowiedniej opcji FCS.

Jak już wspominaliśmy, dwie opcje LCP (PFC i ACFC) związane są z kompresją pola *Protokół* oraz eliminacją pól *Adres* i *Sterowanie* ze strumienia transmisyjnego.

Jeszcze inna opcja LCP związana jest z wyborem algorytmu uwierzytelniania.

Mechanizm „umiędzynarodowienia" (*internationalization*[4]) LCP, opisany w dokumencie [RFC2484], umożliwia używanie znaków narodowych i generalnie określonego języka w implementacjach LCP. Zestaw znaków wybierany jest spośród standardowych opcji (patrz [IANA-CHARSET]), zaś sposób specyfikowania języka narodowego opisany jest w dokumentach [RFC5646] i [RFC4647].

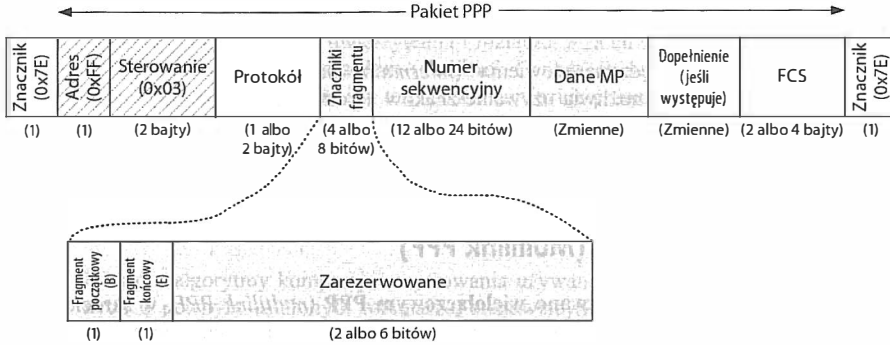
3.6.2. Wielołączowe PPP (Multilink PPP)

Specjalna opcja LCP, **zwana wielołączowym PPP** (*multilink PPP*, w skrócie MP — patrz [RFC1990]), umożliwia agregowanie wielu łączy punkt-punkt do wspólnego funkcjonowania. Jest to idea podobna do opisywanego wcześniej agregowania łączy; wykorzystywana była jako narzędzie agregowania kanałów w sieciach komutowanych (vide kanały ISDN B). Agregowanie takie umożliwia rozdzielenie ruchu PPP na kilka łączy, co np. wykorzystywane było do łączenia domowego komputera z dostawcą Internetu za pośrednictwem dwóch równoległych modemów, na co — oczywiście — mogli sobie pozwolić posiadacze dwóch linii telefonicznych. Dzielenie ramek pomiędzy kilka łączy wiąże się z ich fragmentowaniem i defragmentowaniem, które to funkcje implementowane są przez LCP, łącznie z negocjowaniem ich parametrów. Rezultat zagregowania łączy, zwany **wiązką** (*bundle*), funkcjonuje jako kompletne łącze wirtualne, z własną konfiguracją; łącza składowe wiązki (zwane w oryginale „uczestnikami" — *member links*) mogą niezależnie posiadać swoje własne opcje.

Oczywistą metodą realizacji MP jest naprzemienne kierowanie ramek (a właściwie — ich fragmentów) do poszczególnych łączy składowych; ów „karuzelowy" algorytm, zwany poglądowo **algorytmem kasjera bankowego** (*bank tellers's algorithm*), prowadzi do zjawiska nieznanego na gruncie „zwykłego" łącza PPP — zmiany kolejności pakietów (*packet reordering*), czyli przybywania ich do odbiorcy w kolejności innej niż ta, w której są wysyłane przez nadawcę. Zjawiska tego nie toleruje wiele protokołów warstwy wyższej, a nawet te, które radzą sobie z nim bez problemu (np. TCP czy IP), doświadczają w związku z tym znaczącej utraty wydajności (w porównaniu z osiąganą przy zachowaniu oryginalnej kolejności pakietów). Stąd wniosek, że kwestia zmiany kolejności pakietów musi być „załatwiona" całkowicie w ramach MP — protokoły warstwy wyższej muszą otrzymywać uporządkowany ciąg pakietów. Aby porządkowanie takie mogło być wykonywane przez odbiorcę, poszczególne pakiety zawierają **nagłówek porządkujący** (*sequencing header*) o rozmiarze 2 lub 4 bajtów. Ogólnie struktura pakietu MP prezentuje się tak, jak przedstawiono na rysunku 3.25.

Umiejscowienie danego pakietu w ciągu fragmentów oryginalnej ramki określone jest przez znaczniki *B* i *E* oraz pole *Numer sekwencyjny*. Fragment początkowy wyróżniony jest przez ustawienie znacznika *B* na 1, fragment końcowy przez ustawienia na 1 znacznika *E*. Jeżeli więc ramka nie została pofragmentowana, kombinacja znaczników *BE* ma wartość 11, w przeciwnym razie w początkowym fragmencie są one równe 10, w końcowym — 01, zaś we fragmentach pośrednich — 00. W polu *Numer sekwencyjny* zapisany

[4] Wyraz *internationalization* bywa często skracany do postaci „i18n" — między literami „i" oraz „n" występuje 18 liter — *przyp. tłum.*



Rysunek 3.25. *Fragment MP, zawierający nagłówek sekwencjonowania, umożliwiający przywrócenie oryginalnej kolejności fragmentów. Nagłówek ten występować może w dwóch wersjach: krótkiej (2-bajtowej) i długiej (4-bajtowej)*

jest numer kolejny fragmentu, liczony względem fragmentu początkowego: dla fragmentu początkowego równy jest zeru i inkrementowany w kolejnych fragmentach. Zauważmy, że obszar związany z fragmentacją może występować albo w formacie „długim", 4-bajtowym (8 bitów znaczników + 24-bitowy numer sekwencyjny), albo w formacie „krótkim", 2-bajtowym (4 bity znaczników + 12-bitowy numer sekwencyjny). Wybór między tymi dwoma wariantami jest przedmiotem opcji LCP *short sequence number* (typ 18).

Wykorzystywanie MP związane jest z ustawieniem opcji LCP o nazwie *Multilink Maximum Received Reconstructed Unit* (MRRU, typ 18), stanowiącej odpowiednik parametru MRU odnoszony do całej wiązki. Ramki, których rozmiar przekracza wartość MRU dla poszczególnych łączy składowych, mogą być przesyłane w ramach MP, o ile nie są większe niż wynika to z MRRU.

Przynależność łącza składowego do określonej wiązki wynika z opcji LCP o nazwie *endpoint discriminator* (typ 19) — wartość tej opcji jest jednakowa dla wszystkich łączy składowych danej wiązki i może być np. numerem telefonu, identyfikatorem utworzonym na bazie adresu IP lub MAC bądź też łańcuchem wybranym arbitralnie przez administratora. Szczegółowa struktura tej opcji i związane z nią wymagania opisane są szczegółowo w dokumencie [RFC1990].

W dokumencie tym znajduje się także wymaganie sprawiedliwego obciążania poszczególnych łączy składowych wiązki — na każde z nich trafiać powinna w przybliżeniu ta sama (w danym przedziale czasu) liczba fragmentów. Możliwe są jednak bardziej wyrafinowane sposoby zarządzania dystrybucją fragmentów między łącza wiązki, realizowane przez protokoły o nazwie BAP (*Bandwidth Allocation Protocol* — protokół przydziału pasma) i BACP (*Bandwidth Allocation Control Protocol* — protokół sterowania przydziałem pasma) opisane w dokumencie [RFC2125]. Pierwszy z wymienionych umożliwia dynamiczne dodawanie do i usuwanie z wiązki łączy składowych, zadaniem drugiego jest natomiast wymiana informacji na temat sposobu funkcjonowania pierwszego. Oba ułatwiają realizację mechanizmu *pasma na żądanie* (*bandwidth on demand*, w skrócie BOD): w sieciach, w których zapotrzebowanie aplikacji na określone pasmo wiąże się ze stałym przydziałem zasobów — np. puli połączeń telefonicznych — BOD prowadzi monitorowanie ruchu i w razie dużego obciążenia istniejących połączeń stara się przydzielać

nowe (które zostaną zwolnione, gdy obciążenie się odpowiednio zmniejszy). W ten oto sposób intensywność wykorzystywania puli dostępnych połączeń (i związane z tym opłaty) utrzymywane są na poziomie odpowiednim (nie za niskim, nie za wysokim) do aktualnych wymagań funkcjonujących w sieci aplikacji.

Ponieważ protokół BAP operuje na poszczególnych łączach składowych wiązki, konieczne jest jednoznaczne identyfikowanie tychże. Rolę tę pełni opcja LCP *link discriminator* (typ 23), będąca (unikalną dla łącza w ramach wiązki) 16-bitową liczbą naturalną. Protokół BACP dochodzi do głosu tylko raz dla danej wiązki, w fazie negocjowania, gdy stacja wchodzi w stan *Sieć* (patrz rysunek 3.24), a jego zadaniem jest zidentyfikowanie *uprzywilejowanej stacji* (*favored peer*): jeśli mianowicie w danej chwili konfigurowanych jest kilka wiązek między wieloma stacjami, preferowane są żądania BAP pochodzące od stacji uprzywilejowanej. BAP wykorzystuje trzy typy pakietów: żądania (*request*), odpowiedzi (*response*) i wskazania (*indications*). *Żądanie* może dotyczyć dodania łącza do wiązki lub usunięcia łącza z wiązki przez partnera. Za pomocą *wskazań* rezultat żądania dodania nowego łącza komunikowany jest autorowi tego żądania; wskazania podlegają potwierdzaniu. *Odpowiedź* jest komunikatem `ACK` lub `NACK` wysłanym w reakcji na żądanie lub wskazanie. Więcej szczegółów na ten temat znajduje się w dokumencie [RFC2125].

3.6.3. Protokół sterowania kompresją (CCP)

Protokół PPP ujrzał światło dzienne w czasach, gdy łączność między komputerami odbywała się przeważnie na bazie wolnych łączy modemowych. W konsekwencji, w celu minimalizowania rozmiaru przesyłanych strumieni danych opracowywano różne metody ich kompresji. Mowa tu o kompresji innego typu niż wykonywana sprzętowo przez modemy (V.42bis, V.44 itp.) czy też *kompresji nagłówków*, którą opisywać będziemy w dalszym ciągu rozdziału. Obecnie dostępnych jest kilka opcji kompresji; LCP, w celu wyboru jednej z nich dla każdego kierunku transmisji, wykorzystuje protokół CCP (*Compression Control Protocol* — protokół sterowania kompresją, patrz [RFC1962]). Protokół ten funkcjonuje podobnie do protokołów NCP (piszemy o nich w punkcie 3.6.5), lecz ogranicza swą aktywność do jednorazowej akcji konfigurowania kompresji w czasie, gdy LCP realizuje procedurę negocjowania połączenia.

Dokładniej mówiąc, CCP wykonuje swą akcję po przejściu maszyny stanowej do stanu *Sieć*. Wykorzystuje on taką samą wymianę komunikatów (i w takich samych formatach) jak LCP, jednak z pewnymi różnicami: w polu *Protokół* znajduje się wartość `0x80FD`, dostępne są dodatkowe opcje, a oprócz standardowych wartości pola *Kod* (1 – 7) używane są jeszcze dwie: żądanie resetowania (`reset-request` — `0x0e`) i potwierdzenie resetowania (`reset-ACK` — `0x0f`). Komunikat `reset-request` wysyłany jest do partnera w przypadku stwierdzenia błędu w skompresowanej ramce i oznacza żądanie zresetowania stanu automatu kompresującego (jego słowników, zmiennych stanu, maszyn stanu itp.); po wykonaniu resetowania stacja wysyła potwierdzenie tego faktu w postaci komunikatu `reset-ACK`.

W części informacyjnej ramki PPP (obejmującej dane LCP i ewentualne dopełnienie) może być przesyłanych kilka skompresowanych pakietów. W polu *Protokół* takiej ramki znajduje się wartość `0x00FD`, lecz mechanizm wykrywania wielokrotnej kompresji datagramu zależny jest od tego, jaki konkretnie użyty został algorytm kompresji (patrz punkt 3.6.6). Gdy protokół CCP używany jest w połączeniu z MP, kompresja może być

realizowana na dwa sposoby: dane mogą być kompresowane przed ich rozdziałem na łącza składowe bądź też kompresja może być stosowana niezależnie na poszczególnych łączach składowych już po rozdzieleniu danych. W tym drugim przypadku pole *Protokół* ramki PPP zawiera wartość 0x00FB.

CCP ma do dyspozycji niemal tuzin różnych algorytmów kompresji (patrz [PPPn]). Większość tych algorytmów nie jest usankcjonowana w formie oficjalnych standardów IETF, choć niektóre są treścią informacyjnych dokumentów RFC (tak jak schemat kompresji BSD opisany w [RFC1977] czy też *Microsoft Point-to-Point Compression Protocol* —MPPC — opisany w [RFC2118]).

Warto w tym miejscu nadmienić, że skompresowane ramki PPP rekonstruowane (dekompresowane) są przed wykonaniem dalszego przetwarzania, kompresja jest więc mechanizmem przezroczystym dla operacji PPP w wyższych warstwach — abstrahują one od szczegółów kompresji ramek.

3.6.4. Uwierzytelnianie PPP

Zanim łącze stanie się operatywne po przejściu do stanu *Sieć*, często wymagane jest jeszcze uwiarygodnienie komunikujących się stron, czyli zweryfikowanie ich tożsamości, w procesie zwanym **uwierzytelnianiem** (*authentication*). W podstawowej specyfikacji PPP uwierzytelnianie jest opcjonalne i domyślnie rezygnuje się z niego (o czym należy pamiętać, analizując rysunek 3.24). Najczęściej jednak uwierzytelnianie jest realizowane za pomocą różnych protokołów, jakie w tym celu opracowano na przestrzeni minionych lat. W tym rozdziale zajmiemy się uwierzytelnianiem wyłącznie w ujęciu wysokopoziomowym, odkładając jego szczegółową analizę do rozdziału 18., poświęconego zagadnieniom bezpieczeństwa.

Najprostszą alternatywę dla rezygnacji z uwierzytelniania stanowi schemat zwany **uwierzytelnianiem za pomocą hasła** (*Password Authentication Protocol*, w skrócie PAP): jedna ze stacji żąda przesłania hasła od drugiej, która w formie odpowiedzi żądane hasło przesyła. W polu *Protokół* ramek zaangażowanych w ten proces znajduje się wartość 0xC023. Hasło transmitowane jest przez łącze w postaci jawnej (niezaszyfrowanej), może więc zostać łatwo przechwycone przez intruza, który tym samym zyskuje możliwość podszywania się pod legalnego użytkownika. Schemat ten, jako mało bezpieczny, nie jest więc zalecany do użytku, zwłaszcza w kontekście mechanizmów bardziej bezpiecznych.

Jednym z takich mechanizmów jest **uwierzytelnianie oparte na wyzwaniu** (*Challenge-Handshake Authentication Protocol*, w skrócie CHAP — patrz [RFC1994]). Obie stacje współdzielą klucz szyfrowania, często generowany na podstawie hasła; stacja **A** żądająca od stacji **B** uwierzytelnienia przesyła tejże losowo wybraną liczbę (stanowiącą tytułowe „wyzwanie"), która po zaszyfrowaniu wspomnianym kluczem jest odsyłana jako odpowiedź. Stacja **A** wykonuje identyczne szyfrowanie i porównuje jego wynik z otrzymaną odpowiedzią — identyczność obu wartości jest świadectwem znajomości rzeczonego klucza przez stację **B**. Szyfrowanie odbywa się przy użyciu tzw. **funkcji jednokierunkowej** (*one-way function*), czyli takiej, dla której obliczenie funkcji odwrotnej jest niewykonalne ze względów praktycznych. Przesyłana przez łącze wartość jest za każdym razem inna, jeśli więc nawet zostanie przechwycona przez intruza (w postaci jawnej lub zaszyfrowanej), ten nie będzie mógł jej wykorzystać w niecnych celach (długo musiałby

czekać na to, aż stacja **A** ponownie użyje wartości wyzwania, którą udało mu się przechwycić). Podstawowym mankamentem tego schematu jest jednak podatność na atak z człowiekiem pośrodku (*man-in-the-middle-attack*), którego istotę wyjaśniamy w rozdziale 18.

Opisywany w dokumencie [RFC3748] framework EAP, dostępny dla wielu różnych środowisk sieciowych, oferuje wiele (ponad 40) różnych metod uwierzytelniania, od prostych protokołów w rodzaju PAP i CHAP, do środków bardziej wyrafinowanych (także wykorzystujących karty inteligentne czy cechy biometryczne). Wspomniany dokument definiuje tylko format komunikatu przystosowany do enkapsulowania wielu formatów danych uwierzytelniających, natomiast sposób przenoszenia komunikatów EAP przez różne typy łączy jest z założenia przedmiotem odrębnych specyfikacji.

Gdy EAP używany jest w kontekście PPP, zmienia się nieco opisywany wcześniej podstawowy schemat uwierzytelniania: zamiast negocjowania konkretnej metody uwierzytelniania we wczesnej fazie nawiązywania połączenia (czyli ustanawiania łącza LCP) następuje odłożenie operacji uwierzytelniających do fazy *Uwierzytelnianie* (czyli stanu bezpośrednio poprzedzającego stan *Sieć*). Dzięki temu w procesie uwierzytelniania może zostać użyty bogatszy zestaw informacji, niezbędnych dla podejmowania decyzji przez **serwery zdalnego dostępu** (RAS — *Remote Access Servers*). W sytuacji gdy wiele różnych mechanizmów uwierzytelniania może zostać sprowadzonych do konkretnej, ujednoliconej postaci komunikatów EAP, serwer kontrolujący dostęp do sieci nie musi zajmować się szczegółami tych komunikatów i powierza decyzje w kwestii zezwolenia (lub nie) na dostęp innym wyspecjalizowanym serwerom (np. serwerowi RADIUS opisywanemu w [RFC2865]). Schemat ten jest obecnie powszechnie stosowany przez przedsiębiorstwa i dostawców Internetu.

3.6.5. Protokoły sterowania siecią (NCP)

Chociaż w kontekście łącza PPP można wykorzystywać wiele **protokołów kontroli sieci** (NCP — *Network Control Protocols*), w wielu przypadkach równolegle, skoncentrujemy się w tym miejscu na tych związanych z IPv4 i IPv6, czyli na protokołach (odpowiednio) IPCP (patrz [RFC1332]) i IPV6CP (patrz [RFC5072]). Gdy LCP zakończy ustanawianie połączenia oraz uwierzytelnianie i łącze znajdzie się w stanie *Sieć*, stacje mogą rozpocząć negocjowanie dotyczące warstwy sieciowej, opcjonalnie obejmujące właśnie zestaw protokołów NCP (w którym to zestawie zwykle znajduje się IPCP).

IPCP — standardowy NCP dla IPv4 — organizuje szczegóły transportu datagramów IPv4 przez łącze PPP, m.in. parametry **kompresji nagłówków IPv4 metodą Van Jacobsona** (zwanej popularnie **kompresją VJ** — patrz [RFC1144]). Pakiety IPCP mogą być wymieniane między stacjami od momentu, gdy maszyny stanu tych stacji osiągną stan *Sieć*. IPCP wykorzystuje ten sam format pakietów, co LCP, z tą różnicą, że pole *Protokół* zawiera wartość `0x8021`, a zawartość pola *Kod* ograniczona jest do przedziału wartości 1 – 7 oznaczających typ komunikatu: specyficzny dla dostawcy (patrz [RFC2153]), `configure-request`, `configure-ACK`, `configure-REJECT`, `terminate-request`, `terminate-ACK` lub `code-REJECT`. Protokół IPCP obejmuje negocjowanie kilku opcji, m.in. protokół kompresji (2), adres IPv4 (3) i mobilność IPv4 (4 — patrz [RFC2290]), a także opcji związanych z lokalizacją głównego i alternatywnego serwera DNS (patrz rozdział 11.).

Podobnie IPV6CP wykorzystuje ten sam format pakietów, co LCP, z różnicą dotyczącą dwóch opcji: identyfikatora interfejsu i protokołu kompresji IPv6. Identyfikator interfejsu jest odzwierciedleniem 64-bitowej wartości IID (o której pisaliśmy w rozdziale 2.) stanowiącej podstawę dla tworzenia adresu IPv6 lokalnego dla łącza: adres ten powstaje przez poprzedzenie wspomnianego IID prefiksem wskazującym lokalność dla łącza (tak właśnie realizowany jest proces autokonfiguracji — patrz rozdział 6.). Adres ten, jako lokalny dla łącza, nie musi być globalnie unikatowy.

3.6.6. Kompresja nagłówków

W związku z tym, że tradycyjne „dodzwaniane" łącza PPP charakteryzowały się małą szybkością transmisji (najwyżej 57 600 b/s), poszukiwano różnych metod kompresji zmierzających do minimalizowania rozmiaru przesyłanych danych. U podstawy jednej z takich metod leży fakt, że przez wspomniane łącza transmitowane były przeważnie niewielkie pakiety TCP/IP (np. potwierdzenia TCP — patrz rozdział 15.), rozpoczynające się nagłówkami pozostającymi bez zmian w kolejnych pakietach tego samego połączenia TCP (lub zmieniającymi się w niewielkim stopniu); podobnie miała się rzecz w przypadku innych protokołów warstw wyższych. Uniknięcie powtórnego transmitowania takich samych nagłówków oznaczałoby sporą oszczędność: typowy nagłówek IPv4 ma rozmiar 20 bajtów, a poprzedzający go nagłówek TCP — co najmniej 20 bajtów (bez opcji). Zapamiętanie tychże nagłówków w odpowiedniej tablicy i zastępowanie ich w transmitowanych pakietach jednobajtowym identyfikatorem połączenia pozwalałoby więc na zaoszczędzenie co najmniej 39 bajtów na jednym pakiecie. Mechanizm ten opisywany jest (pod nazwą CSLIP — *Compressed Serial Line IP*) w dokumencie [RFC1144] w odniesieniu do poprzednika PPP — protokołu SLIP (*Serial Line IP* — szeregowe łącze IP). Ponadto, nawet jeżeli pewne pola w nagłówku zmieniają się w kolejnych pakietach, to zmieniają się zwykle w niewielkim stopniu i można by transmitować je w postaci *przyrostów* („delt"), a nie oryginalnych wartości (wraz z indeksami wskazującymi miejsce zmian w nagłówku). Powyższa metoda, stanowiąca istotę wspomnianej wcześniej *kompresji VJ*, oznacza efektywne zastąpienie 40-bajtowych nagłówków 3 – 4-bajtowymi odpowiednikami w każdym pakiecie. Wobec niewielkich (przeważnie) rozmiarów ładunku użytecznego tych pakietów oznaczało to wyraźne poprawienie wydajności protokołu TCP/IP realizowanego na bazie łączy dodzwanianych.

Kolejnym kamieniem milowym na drodze ewolucji kompresji nagłówków jest metoda nazywana (po prostu) „kompresją nagłówków IP" (*IP header compression*), opisywana w dokumentach [RFC2507] i [RFC3544]. Stanowi ona logiczne rozszerzenie i uogólnienie kompresji VJ na wiele pakietów związanych z protokołami TCP i UDP warstwy transportowej, operującymi na bazie zarówno IPv4, jak i IPv6 w warstwie sieciowej, transmitowanymi przy użyciu różnych łączy, nie tylko PPP.

Wspomniane metody kompresji niosą ze sobą pewne niebezpieczeństwo — to mianowicie, że przekłamanie w nagłówku jednego pakietu propaguje się automatycznie na dużą liczbę pakietów następnych. W związku z tym, w dokumencie [RFC2507] wyartykułowane zostało zapotrzebowanie na dodatkowe metody rygorystycznej kontroli poprawności przesyłanych pakietów, niezbędne w przypadku kompresowania ramek na łączach nieposiadających mechanizmu weryfikacji integralności (takiego jak pole FCS w ramkach PPP).

Opisywana w dokumencie [RFC5225] metoda **solidnej kompresji nagłówków** (*Robust Header Compression*, w skrócie ROHC) stanowi kolejne uogólnienie poprzedniczki na jeszcze większą liczbę protokołów transportowych i większy zestaw typów łączy (wśród nich znajduje się, oczywiście, PPP). Jej nowością jest możliwość równoległej realizacji kilku różnych metod kompresji, zależnych od typu konkretnego pakietu.

3.6.7. Przykład

W charakterze praktycznej ilustracji opisywanych mechanizmów wykorzystamy raport debugowania serwera PPP, świadczącego usługi na rzecz klienta połączonego przez modem telefoniczny. Wspomniany serwer jest komputerem zarządzanym przez system Linux, klient jest stacją z systemem MS Windows Vista, implementującą protokół IPv6 i ustawioną (w celach demonstracyjnych) na negocjowanie MP nawet w przypadku pojedynczego łącza (*Właściwości/Opcje/Ustawienia PPP*). Serwer ustawiony jest na negocjowanie protokołu szyfrowania za pomocą CCP (patrz sekcja MPPE poniższego listingu):

```
data dev=ttyS0, pid=28280, caller='none', conn='38400',
     name='',cmd='/usr/sbin/pppd', user='/AutoPPP/'
pppd 2.4.4 started by a_ppp, uid 0
using channel 54
Using interface ppp0
ppp0 <--> /dev/ttyS0
sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth eap>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfNak id=0x1 <auth chap MS-v2>]
sent [LCP ConfReq id=0x2 <asyncmap 0x0> <auth chap MS-v2>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfAck id=0x2 <asyncmap 0x0> <auth chap MS-v2>
     <magic 0xa5ccc449><pcomp> <accomp>]
rcvd [LCP ConfReq id=0x2 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp><callback C8CP> <mrru 1614>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84.f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [LCP ConfRej id=0x2 <callback C8CP> <mrru 1614>]
rcvd [LCP ConfReq id=0x3 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84.f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [LCP ConfAck id=0x3 <asyncmap 0x0> <magic 0xa531e06>
     <pcomp> <accomp>
     <endpoint [local:12.92.67.ef.2f.fe.44.6e.84 f8.
                c9.3f.5f.8c.5c.41.00.00.00.00]>]
sent [CHAP Challenge id=0x1a <4d53c52b8e7dcfe7a9ea438b2b4daf55>,
     name = "dialer"]
rcvd [LCP Ident id=0x4 magic=0xa531e06 "MSRASV5.20"]
rcvd [LCP Ident id=0x5 magic=0xa531e06 "MSRAS-0-VISTA"]
rcvd [CHAP Response id=0x1a
     <4b5dc95ed4e1788b959025de0233d4fc0000000
      00000000033a555d2a77bd1fa692f2a0af707cd 4f0c0072c379c82e0f00>,
     name = "dialer"]
sent [CHAP Success id=0x1a
     "S=7E0868513215C875208EF6725EF8A9945C28E918M=Access granted"]
sent [CCP ConfReq id=0x1 <mppe +H -M +S +L -D -C>]
rcvd [IPV6CP ConfReq id=0x6 <addr fe80::0000:0000:dead:beef>]
sent [IPV6CP TermAck id=0x6]
rcvd [CCP ConfReq id=0x7 <mppe -H -M -S -L -D +C>]
```

```
sent [CCP ConfNak id=0x7 <mppe +H -M +S +L -D -C>]
rcvd [IPCP ConfReq id=0x8 <compress VJ 0f 01> <addr 0.0.0.0>
     <ms-dns1 0.0.0.0> <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0>
     <ms-wins 0.0.0.0>]
sent [IPCP TermAck id=0x8]
rcvd [CCP ConfNak id=0x1 <mppe -H -M +S -L -D -C>]
sent [CCP ConfReq id=0x2 <mppe -H -M +S -L -D -C>]
rcvd [CCP ConfReq id=0x9 <mppe -H -M +S -L -D -C>]
sent [CCP ConfAck id=0x9 <mppe -H -M +S -L -D -C>]
rcvd [CCP ConfAck id=0x2 <mppe -H -M +S -L -D -C>]
MPPE 128-bit stateful compression enabled
sent [IPCP ConfReq id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
sent [IPV6CP ConfReq id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
rcvd [IPCP ConfAck id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
rcvd [IPCP ConfReq id=0xa <compress VJ 0f 01>
     <addr 0.0.0.0> <ms-dns1 0.0.0.0>
     <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
sent [IPCP ConfRej id=0xa <ms-wins 0.0.0.0> <ms-wins 0.0.0.0>]
rcvd [IPV6CP ConfReq id=0xb <addr fe80::0000:0000:dead:beef>]
sent [IPV6CP ConfAck id=0xb <addr fe80::0000:0000:dead:beef>]
rcvd [IPCP ConfAck id=0x1 <compress VJ 0f 01> <addr 192.168.0.1>]
rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
local LL address fe80::0206:5bff:fedd:c5c3
remote LL address fe80::0000:0000:dead:beef
rcvd [IPCP ConfReq id=0xc <compress VJ 0f 01>
     <addr 0.0.0.0> <ms-dns1 0.0.0.0> <ms-dns3 0.0.0.0>]
sent [IPCP ConfNak id=0xc <addr 192.168.0.2> <ms-dns1 192.168.0.1>
     <ms-dns3 192.168.0.1>]
sent [IPCP ConfAck id=0xd <compress VJ 0f 01> <addr 192.168.0.2>
     <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
local IP address 192.168.0.1
remote IP address 192.168.0.2
... dane ...
```

Widzimy tu cokolwiek skomplikowaną wymianę komunikatów PPP (etykieta `rcvd` oznacza komunikaty otrzymywane (*received*) przez serwer, etykieta `sent` — komunikaty wysyłane przez serwer). Proces serwera tworzy (wirtualny) interfejs sieciowy `ppp0` i oczekuje na połączenia przychodzące do modemu podłączonego do portu `ttyS0`. Połączenie nadchodzi, w reakcji na co serwer wysyła żądanie asynchronicznego mapowania znaków 0x00, uwierzytelniania EAP oraz kompresji PFC i ACFC. Klient odrzuca żądanie uwierzytelniania EAP (`ConfNak`), proponując w zamian protokół MS-CHAP-v2 opisany w dokumencie [RFC2759]. Serwer ponawia więc żądanie, zastępując EAP sugerowanym protokołem, i tym razem klient potwierdza zaakceptowanie żądania (`ConfAck`). Kolejne przychodzące żądanie dotyczy protokołu odwołania zwrotnego CBCP; określenia MRRU (1614 bajtów) oraz dyskryminatora punktu końcowego (*endpoint*) związane są z MP. Serwer odrzuca (`ConfRej`) opcje CBCP i MP; klient ponownie przesyła dyskryminator punktu końcowego (tym razem bez MRRU) i jego żądanie zostaje zaakceptowane. Serwer inicjuje następnie proces uwierzytelniania (w oparciu o wynegocjowany protokół), wysyłając „wyzwanie" oraz nazwę `dialer`. Jeszcze przed nadejściem potwierdzenia tego żądania nachodzą do serwera dwa komunikaty identyfikujące klienta za pomocą łańcuchów `MSRASV5.20` i `MSRAS-0-VISTA`. Przychodząca następnie odpowiedź na żądanie CHAP zostaje przez serwer zaakceptowana i uwierzytelnienie uznane zostaje przez serwer jako pomyślnie zakończone. Łącze wchodzi w stan *Sieć*.

Następuje kolejna wymiana komunikatów. Serwer wysyła żądanie CCP ze wskazaniem protokołu MPPE (*Microsoft Point-to-Point Encryption* — patrz [RFC3078]). Na pierwszy rzut oka to nieco dziwne: MPPE to przecież protokół szyfrowania, nie kompresji, wydłużający pakiet o 4 bajty. To prawda, ale z drugiej strony, to prosty sposób wczesnego włączenia szyfrowania w proces negocjacji. Opcje protokołu MPPE określają żądanie operacji bezstanowych (`H`), dostępność poszczególnych klas kluczy szyfrowania (słabe [*low*] — `L`, przeciętne [*medium*] — `M`, bezpieczne [*secure*] — `S`) i wykorzystywanie specyficznego protokołu MPPC opisanego w dokumencie [RFC2118] (`C`); bit parametr `D` jest ignorowany w obecnej wersji MPPE. Ostatecznie strony uzgadniają operacje z informacją o stanie (`H-`) i używanie najsilniejszych kluczy (`S+`). Zauważmy, że w trakcie negocjowania parametrów szyfrowania klient nadsyła żądanie IPCP, które kwitowane jest przez serwer komunikatem `TermAck`, zdefiniowanym przez LCP i przyjętym także przez IPCP, a oznaczającym (w tym kontekście) „potrzebę renegocjacji" (patrz [RFC1661]).

Po zakończeniu negocjacji związanych z MPPE serwer wysyła żądanie dotyczące kompresji VJ, informując klienta o swych adresach: IPv4 (`192.168.0.1`) i IPv6 (`fe80::0206:5bff:fedd:c5c3`) — ten ostatni utworzony jest na podstawie adresu MAC `00:06:5B:DD:C5:C3`. Klient potwierdza tę informację, jednocześnie sugerując swój własny adres `0.0.0.0` i taki sam adres serwera DNS w kontekście protokołu IPCP, sugestia ta zostaje przez serwer odrzucona. Klient ponawia propozycję, tym razem żądając protokołu IPV6CP i podając adres IPv6 `fe80::0000:0000:dead:beef` — propozycja zostaje przyjęta. Klient potwierdza oba adresy serwera i ponownie wysyła żądanie z zerowymi adresami IPv4, na co serwer odpowiada sugestią (`ConfNak`) użycia adresu `192.168.0.1`; klient sugestię przyjmuje i wysyła poprawione żądanie, które zostaje przez serwer zaakceptowane i potwierdzone.

Jak widzimy na przedstawionym przykładzie, negocjacje w ramach PPP są zarówno elastyczne, jak i uciążliwe: mnogość różnych opcji i ich negocjowanie metodą prób i błędów, odrzucanie, sugestie renegocjacji itp. Nie stanowi to dużego problemu na łączach o małym opóźnieniu, lecz nietrudno wyobrazić sobie, jak powolne stawać się może w przypadku łączy, na których dotarcie sygnału do stacji docelowej zajmuje kilka sekund (a nawet więcej), np. łączy satelitarnych: procedura ustanawiania połączenia może być wówczas wyraźnie frustrująca dla użytkownika.

3.7. Pętla zwrotna

Choć w pierwszej chwili może się to wydawać dziwne, w wielu sytuacjach aplikacja kliencka musi komunikować się z serwerami uruchamianymi w tym samym komputerze, wykorzystującymi protokół IP (głównie serwerami TCP/IP). Większość implementacji protokołów TCP/IP oferuje taką możliwość w postaci **pętli zwrotnej** (*loopback*), będącej w istocie programową (wirtualną) analogią fizycznego interfejsu sieciowego. Aplikacja komunikuje się z lokalnym serwerem tak samo jak z serwerami zdalnymi — za pomocą odpowiedniego adresu IP. W protokole IPv4 na potrzeby takiej komunikacji lokalnej zarezerwowano adresy `127/8`; systemy operacyjne grupy Linux wykorzystują tylko jeden adres `127.0.0.1`, a interfejs lokalny opatrzony zostaje nazwą `lo`; w systemach Windows również jest to adres `127.0.0.1`, lecz interfejs lokalny nosi nazwę `localhost`. W protokole IPv6 rolę tę pełni adres `::1` (pisaliśmy o tym w rozdziale 2). Datagram wysłany pod adres lokalny nie ma prawa opuścić komputera; mimo iż łatwo sobie wyobrazić, że

protokoły warstwy transportowej wykrywają lokalny charakter adresu docelowego i organizują transport pakietów „na skróty", z całkowitym pominięciem warstwy sieciowej (i warstw niższych), to jednak większość implementacji TCP/IP wykorzystuje tę możliwość połowicznie: pakiet podlega kompletnemu przetworzeniu w warstwie sieciowej i dopiero po jej opuszczeniu jest do niej natychmiast zawracany. Dzięki temu możliwe jest testowanie protokołów warstwy sieciowej i ich wydajności bez konieczności instalowania rzeczywistej karty sieciowej. W systemie Linux statystykę interfejsu lokalnego można obejrzeć w następujący sposób:

```
Linux% ifconfig lo
lo Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING MTU:16436 Metric:1
          RX packets:458511 errors:0 dropped:0 overruns:0 frame:0
          TX packets:458511 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:266049199 (253.7 MiB)
          TX bytes:266049199 (253.7 MiB)
```

Widzimy tu wykorzystanie adresu `127.0.0.1` na potrzeby interfejsu lokalnego i potwierdzenie znanego faktu, że adresy IPv4 zarezerwowane dla interfejsu lokalnego to `127.0.0.1` przy masce sieciowej `255.0.0.0`, czyli adresy `127/8` (odpowiadające 127. grupie adresów w klasie A), a dla IPv6 jest to prefiks `::1/128`, czyli po prostu jeden adres. Wielkość maksymalnej jednostki transmisji określono na 16436 bajtów (dopuszczalna granica tego parametru to 2 GB). Dość spora porcja ruchu — niemal pół miliona pakietów — została przez interfejs obsłużona bezbłędnie (system zainstalowany został na komputerze dwa miesiące wcześniej); trudno zresztą spodziewać się błędów transmisji odbywającej się faktycznie wewnątrz komputera, bez angażowania jakiejkolwiek fizycznej sieci.

W systemie Windows adapter pętli zwrotnej nie jest domyślnie instalowany, mimo iż wspomniane adresy `127.x.x.x` i `::1` oraz nazwa `localhost` są poprawnie interpretowane. Adapter ten może być przydatny do testowania różnych konfiguracji sieciowych, nawet w sytuacji gdy nie jest dostępny żaden fizyczny interfejs. W systemie Windows XP można wspomniany adapter zainstalować ręcznie, wybierając z *Panelu sterowania* opcję *Dodaj sprzęt*, po czym w kreatorze dodawania nowego sprzętu (po zakończeniu sprawdzania zainstalowanego sprzętu) wybrać w kolejnych oknach dialogowych opcje *Tak, urządzenie zostało już podłączone/Dodaj nowe urządzenie sprzętowe/Zainstaluj sprzęt, który wybiorę ręcznie z listy/Karty sieciowe/Microsoft/Karta Microsoft Loopback*. W Windows Vista i Windows 7 podobny dialog realizowany jest przez program `hdwwiz`, uruchamiany z wiersza poleceń. Po zakończeniu instalacji można zaobserwować jej skutki w sposób następujący (przykład pochodzi z Windows 7):

```
C:\> ipconfig /all
...
Karta Ethernet Połączenie lokalne 2:
   Sufiks DNS konkretnego połączenia :
   Opis . . . . . . . . . . . . . . : Karta Microsoft Loopback
   Adres fizyczny. . . . . . . . . . : 02-00-4C-4F-4F-50
   DHCP włączone . . . . . . . . . . : Tak
   Autokonfiguracja włączona . . . . : Tak
   Adres IPv6 połączenia lokalnego . : fe80::9c0d:77a:52b8:39f0%18(Preferowane)
   Adres IPv4 autokonfiguracji . . . : 169.254.57.240(Preferowane)
```

```
   Maska podsieci. . . . . . . . . . . : 255.255.0.0
   Brama domyślna. . . . . . . . . . . :
   Identyfikator IAID DHCPv6 . . . . . : 302121036
   Serwery DNS . . . . . . . . . . . . : fec0:0:0:ffff::1%1
                                         fec0:0:0:ffff::2%1
                                         fec0:0:0:ffff::3%1
   NetBIOS przez Tcpip . . . . . . . . : Włączony
```

Jak widać, utworzonemu interfejsowi przypisano adresy IPv4 I IPv6, sam zaś interfejs jawi się jako wirtualne urządzenie ethernetowe. Można się teraz przekonać, że komputer honoruje w istocie wiele adresów pętli zwrotnej:

```
C:\> ping 127.1.2.3
Badanie 127.1.2.3 z 32 bajtami danych:
Odpowiedź z 127.1.2.3: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 127.1.2.3: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 127.1.2.3: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 127.1.2.3: bajtów=32 czas<1 ms TTL=128

Statystyka badania ping dla 127.1.2.3:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0
             (0% straty),
Szacunkowy czas błądzenia pakietów w millisekundach:
    Minimum = 0 ms, Maksimum = 0 ms, Czas średni = 0 ms

C:\> ping ::1
Badanie ::1 z 32 bajtami danych:
Odpowiedź z ::1: czas<1 ms
Odpowiedź z ::1: czas<1 ms
Odpowiedź z ::1: czas<1 ms
Odpowiedź z ::1: czas<1 ms

Statystyka badania ping dla ::1:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0
             (0% straty),
Szacunkowy czas błądzenia pakietów w millisekundach:
    Minimum = 0 ms, Maksimum = 0 ms, Czas średni = 0 ms

C:\> ping 169.254.57.240
Badanie 169.254.57.240 z 32 bajtami danych:
Odpowiedź z 169.254.57.240: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 169.254.57.240: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 169.254.57.240: bajtów=32 czas<1 ms TTL=128
Odpowiedź z 169.254.57.240: bajtów=32 czas<1 ms TTL=128

Statystyka badania ping dla 169.254.57.240:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0
             (0% straty),
Szacunkowy czas błądzenia pakietów w millisekundach:
    Minimum = 0 ms, Maksimum = 0 ms, Czas średni = 0 ms
```

Jak widzimy, każdy adres IPv4 o postaci `127.x.x.x` interpretowany jest jako adres pętli zwrotnej; w IPv6 pętli zwrotnej przydzielony jest tylko jeden adres `::1`. Pewna subtelność (wyjaśnimy ją w rozdziale 9.) wiąże się z pytaniem, czy datagramy broadcast i multicast powinny być odsyłane do komputera-nadawcy (poprzez interfejs pętli zwrotnej): decyzja w tej kwestii należy do poszczególnych aplikacji.

3.8. MTU protokołu i MTU ścieżki (PMTU)

Jak wynika z rysunku 3.3, istnieje ograniczenie na rozmiar jednostki protokołu warstwy wyższej (PDU) przenoszonej w ramce ethernetowej — 1500 bajtów — i w przypadku wielu innych ramek warstwy łącza danych obowiązują podobne limity (często o tej samej wartości, z intencją zachowania kompatybilności z Ethernetem). Limit ten nosi nazwę **maksymalnej jednostki transmisji** (MTU — *Maximum Transmission Unit*) i dla wielu sieci pakietowych (m.in. Ethernetu) zdefiniowany jest a priori w specyfikacji. W większości sieci strumieniowych (opartych na łączach szeregowych) jest on natomiast konfigurowalnym parametrem, wykorzystywanym przez protokoły oparte na ramkach, takie jak PPP. Jeśli więc np. datagram IP przekracza pod względem rozmiaru wartość MTU łącza, przez które ma zostać przesłany, protokół IP wykonuje jego **fragmentację**, czyli podział na mniejsze pakiety (**fragmenty**), każdy o wielkości nieprzekraczającej MTU. Szczegółami fragmentacji pakietów zajmować się będziemy w rozdziałach 5. i 10.

W komunikacji między dwoma hostami należącymi do tej samej sieci maksymalny rozmiar wymienianych między nimi pakietów określony jest bezpośrednio przez MTU łącza tej sieci. Gdy ścieżka komunikacji między hostami przebiega przez wiele sieci, każda z nich może charakteryzować się inną wartością MTU; najmniejsza z tych wartości wyznacza wówczas graniczną wielkość pakietu, która umożliwia jego przesłanie przez tę ścieżkę (bez dodatkowego fragmentowania) — wielkość tę nazywamy maksymalną jednostką transmisji dla ścieżki i oznaczamy PMTU, od *Path Maximum Transmission Unit*.

Ponieważ ścieżka komunikacji między dwoma hostami niekoniecznie musi być cały czas jednakowa, wartość PMTU dla tej komunikacji może zmieniać się w czasie. Co więcej, ścieżka tej komunikacji często *nie ma charakteru symetrycznego* — pakiety w kierunku od hosta A do hosta B wędrować mogą ścieżką inną niż pakiety w kierunku przeciwnym.

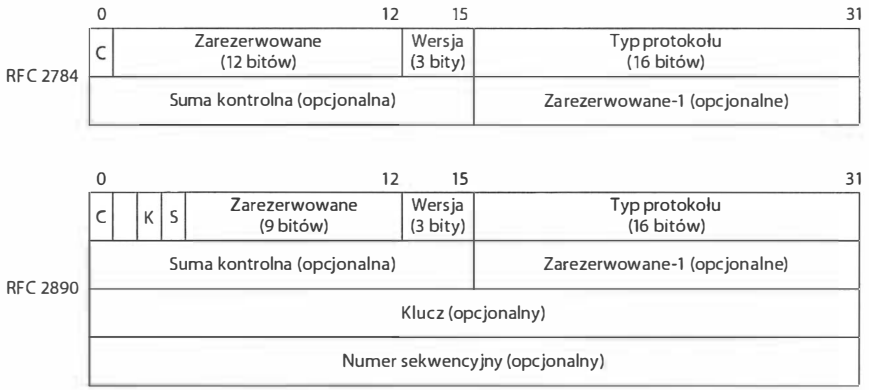
W dokumencie [RFC1191] opisano mechanizm **wykrywania PMTU** (*Path MTU Discovery*, w skrócie PMTUD) dla IPv4, analogiczna specyfikacja dla IPv6 jest przedmiotem dokumentu [RFC1981]. W dokumencie [RFC4821] opisywana jest nowsza metoda, stanowiąca rozszerzenie dwóch wymienionych i wolna od wielu ich mankamentów — **wykrywanie MTU w warstwie pakietującej** (*Packetization Layer Path MTU Discovery* — w skrócie PLPMTUD). Mechanizm PMTUD (lub równoważny) umożliwia dynamiczne określanie wartości MTU w warunkach zmieniających się z upływem czasu; jest on obowiązkową częścią każdej implementacji IPv6. W następnych rozdziałach przedstawimy jego działanie, po uprzednim opisaniu protokołu ICMP oraz fragmentacji datagramów IP. Przy okazji omawiania protokołów TCP i UDP pokażemy także wpływ wartości MTU na wydajność funkcjonowania warstwy transportowej.

3.9. Podstawy tunelowania

Często Internet (lub inna sieć pośrednicząca) wykorzystywany jest wyłącznie jako środek łączności między komputerami: przykładem tak funkcjonującej usługi są prywatne sieci wirtualne (VPN). W implementacji tego typu usług powszechnie wykorzystuje się mechanizm zwany **tunelowaniem** (*tunelling*). Mówiąc ogólnie, polega on na enkapsulowaniu pakietów warstwy niższej w pakietach warstwy wyższej lub tej samej, np. na en-

kapsulowaniu pakietów IPv4 w pakietach IPv4 lub IPv6, czy też enkapsulowaniu ramek ethernetowych w pakietach UDP, IPv4 lub IPv6. Wyraźnie widać, że idea ta stanowi naruszenie (a tak naprawdę — odwrócenie) hierarchii warstw wynikającej z tradycyjnego modelu odniesienia: miejsce rzeczywistych połączeń między warstwami zajmują połączenia wirtualne, implementowane przez specyficzne protokoły, stanowiące podstawę tworzenia tzw. **sieci nakładkowych** (*overlay networks*). Tunelowanie jest techniką niezwykle użyteczną i na swój sposób skomplikowaną; w tym podrozdziale ograniczymy się do omówienia jej wybranych opcji.

Wśród mnogości metod realizujących tunelowanie najczęściej wykorzystywane są trzy protokoły: GRE (*Generic Routing Encapsulation*) opisany w [RFC2784], zdefiniowany przez firmę Microsoft protokół PPTP (*Point-to-Point Tunneling Protocol*) opisany w [RFC2637] oraz L2TP (*Layer 2 Tunneling Protocol*) opisany w [RFC3931]. Istnieje także kilka niestandardowych protokołów enkapsulujących pakiety IP (obu wersji) w innych pakietach IP (obu wersji); właśnie z myślą o ich zastąpieniu rozwiązaniem standardowym zaprojektowano GRE. Podobnie L2TP zaprojektowany został jako ustandaryzowany następca PPTP. Omówimy dwa z wymienionych protokołów — GRE i PPTP, z pewną preferencją drugiego, jako bardziej widocznego dla użytkowników (zauważmy, że nie jest on standardem IETF). L2TP często używany jest w połączeniu z mechanizmem IPsec (patrz rozdział 18.), ponieważ sam z siebie nie posiada żadnych mechanizmów bezpieczeństwa. Protokoły GRE i PPTP są podobne, przeanalizujemy więc nagłówek GRE, który przedstawiono na rysunku 3.26 w postaciach standardowej i rozszerzonej.

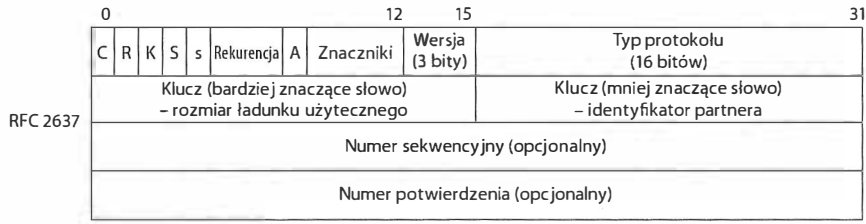


Rysunek 3.26. *Podstawowy nagłówek GRE ma rozmiar tylko 4 bajtów, lecz może także obejmować 2-bajtową sumę kontrolną, stosowaną powszechnie przez wiele protokołów internetowych. Druga wersja nagłówka GRE rozszerzona jest o pola klucza i numeru sekwencyjnego, umożliwiające odtworzenie oryginalnej kolejności fragmentów*

Wersja bazowa GRE (opisana w [RFC2784]) jest, jak widać, dość oszczędna i zapewnia minimalną obsługę enkapsulacji innych pakietów. Bit `C` wskazuje na istotność (1) lub ignorowanie (0) pola *Suma kontrolna*; gdy ustawiony jest bit `C`, w polu tym znajduje się tzw. internetowa suma kontrolna (*Internet checksum*) powszechnie używana do kontroli integralności pakietów związanych z Internetem — algorytm jej obliczania omawiamy w punkcie 5.2.2. Gdy ustawiony jest bit `C`, pole *Zarezerwowane-1* musi być wyzerowane. Dokument [RFC2890] rozszerza ten podstawowy format o pola *Klucz* i *Numer sekwencyjny*;

są to pola opcjonalne, ich istotność uwarunkowana jest ustawieniem na 1 bitów (odpowiednio) K i S. Pole *Klucz* pełni rolę identyfikatora, wspólnego dla wszystkich pakietów należących do tego samego przepływu. Pole *Numer sekwencyjny* umożliwia odtworzenie oryginalnej kolejności odebranych pakietów w przypadku ich „przetasowania" (spowodowanego np. ich przesyłaniem przez wiele różnych łączy).

Mimo iż GRE stanowi podstawę dla PPTP i jest przezeń wykorzystywany, oba protokoły służą cokolwiek innym celom. Tunelowanie GRE wykorzystywane jest zwykle w ramach infrastruktury sieciowej do transmisji między dostawcami Internetu lub do łączności między oddziałami firmy w ramach firmowego intranetu i jako takie niekoniecznie musi wiązać się z szyfrowaniem, choć może być używane w kombinacji z IPsec. Dla odmiany, protokół PPTP wykorzystywany jest najczęściej do transmisji między dostawcą Internetu a jego klientami lub między firmowymi intranetami; szyfrowanie (np. przy użyciu MPPE) jest nieodłącznym elementem tego rodzaju tunelowania. Zasadniczo PPP stanowi kombinację GRE i PPP, tak więc GRE może tworzyć wirtualne łącze punkt-punkt, na bazie którego operować będzie protokół PPP. GRE funkcjonuje na bazie datagramów IPv4 lub IPv6 i jako taki może być uważany za technologię tunelowania na poziomie warstwy 3.; PPTP wykorzystywany jest najczęściej do przenoszenia ramek warstwy 2. (np. ramek ethernetowych) i emulowania w ten sposób bezpośrednich połączeń LAN. Technologia ta może być używana jak ośrodek dostępu zewnętrznych użytkowników do sieci korporacyjnych. W protokole PPTP wykorzystywana jest niestandardowa odmiana nagłówka GRE, widoczna na rysunku 3.27.



Rysunek 3.27. *Nagłówek PPTP wzorowany jest na starszym, niestandardowym nagłówku GRE, zawierającym numer sekwencyjny, narastający licznik potwierdzonych pakietów i pewne informacje identyfikacyjne. Większość pól w pierwszym słowie ma wartość zerową*

Widzimy szereg różnic w stosunku do standardowego nagłówka GRE. Pola *Rekurencja* i *Znaczniki* wywodzą się z wcześniejszych wersji PPTP i obecnie mają wartość 0. W polu *Wersja* znajduje się wartość 1, w polu *Typ protokołu* — wartość 0x880B. Pole *Klucz* podzielone zostało na dwa podpola, zawierające rozmiar ładunku użytecznego (bez uwzględniania nagłówka GRE) oraz identyfikator stacji partnerskiej w sesji. W polu *Numer potwierdzenia* znajduje się największy spośród numerów sekwencyjnych pakietów odebranych dotąd przez partnera. Bity C, R i s (małe) mają wartość 0, bit K ma wartość 1, bity S (duże) i A są wskaźnikami istotności pól (odpowiednio) *Numer Sekwencyjny* i *Numer potwierdzenia* (w pakietach potwierdzających nie występuje ładunek użyteczny ani numer sekwencyjny, więc bit S ma wówczas wartość 0).

Analogicznie do prezentowanego wcześniej raportu z negocjowania połączenia w ramach protokołu PPP, przedstawimy poniżej podobny raport odzwierciedlający funkcjonowanie „surowego" łącza PPP symulowanego przez PPTP. Ponownie serwer jest kom-

puterem linuksowym, a klientem jest komputer z systemem Windows Vista. Raport znajduje się w pliku */var/log/messages* serwera i jest wynikiem włączenia opcji debugowania.

```
pptpd: MGR: Manager process started
pptpd: MGR: Maximum of 100 connections available
pptpd: MGR: Launching /usr/sbin/pptpctrl to handle client
pptpd: CTRL: local address = 192.168.0.1
pptpd: CTRL: remote address = 192.168.1.1
pptpd: CTRL: pppd options file = /etc/ppp/options.pptpd
pptpd: CTRL: Client 71.141.227.30 control connection started
pptpd: CTRL: Received PPTP Control Message (type: 1)
pptpd: CTRL: Made a START CTRL CONN RPLY packet
pptpd: CTRL: I wrote 156 bytes to the client.
pptpd: CTRL: Sent packet to client
pptpd: CTRL: Received PPTP Control Message (type: 7)

pptpd: CTRL: Set parameters to 100000000 maxbps, 64 window size
pptpd: CTRL: Made a OUT CALL RPLY packet
pptpd: CTRL: Starting call (launching pppd, opening GRE)
pptpd: CTRL: pty_fd = 6
pptpd: CTRL: tty_fd = 7
pptpd: CTRL (PPPD Launcher): program binary = /usr/sbin/pppd
pptpd: CTRL (PPPD Launcher): local address = 192.168.0.1
pptpd: CTRL (PPPD Launcher): remote address = 192.168.1.1
pppd: pppd 2.4.4 started by root, uid 0
pppd: using channel 60
pptpd: CTRL: I wrote 32 bytes to the client.
pptpd: CTRL: Sent packet to client
pppd: Using interface ppp0
pppd: Connect: ppp0 <--> /dev/pts/1
pppd: sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth chap MS-v2>
           <magic 0x4e2ca200> <pcomp> <accomp>]
pptpd: CTRL: Received PPTP Control Message (type: 15)
pptpd: CTRL: Got a SET LINK INFO packet with standard ACCMs
pptpd: GRE: accepting packet #0
pppd: rcvd [LCP ConfReq id=0x0 <mru 1400> <magic 0x5e565505>
           <pcomp> <accomp>]
pppd: sent [LCP ConfAck id=0x0 <mru 1400> <magic 0x5e565505>
           <pcomp> <accomp>]
pppd: sent [LCP ConfReq id=0x1 <asyncmap 0x0> <auth chap MS-v2>
           <magic 0x4e2ca200> <pcomp> <accomp>]
pptpd: GRE: accepting packet #1
pppd: rcvd [LCP ConfAck id=0x1 <asyncmap 0x0> <auth chap MS-v2>
           <magic 0x4e2ca200> <pcomp> <accomp>]
pppd: sent [CHAP Challenge id=0x3
           <eb88bfff67d1c239ef73e98ca32646a5>, name = "dialer"]
pptpd: CTRL: Received PPTP Control Message (type: 15)
pptpd: CTRL: Ignored a SET LINK INFO packet with real ACCMs!
pptpd: GRE: accepting packet #2
pppd: rcvd [CHAP Response id=0x3<276f3678f0f03fa57f64b3c367529565000000
           00000000000fa2b2ae0ad8db9d986f8e222a0217a620638a24
           3179160900>, name = "dialer"]
pppd: sent [CHAP Success id=0x3
           "S=C551119E0E1AAB68E86DED09A32D0346D7002E05
           M=Accessgranted"]
pppd: sent [CCP ConfReq id=0x1 <mppe +H -M +S +L -D -C>]
pptpd: GRE: accepting packet #3
pppd: rcvd [IPV6CP ConfReq id=0x1 <addr fe80::1cfc:fddd:8e2c:e118>]
```

```
pppd: sent [IPV6CP TermAck id=0x1]
pptpd: GRE: accepting packet #4
pppd: rcvd [CCP ConfReq id=0x2 <mppe +H -M -S -L -D -C>]
pppd: sent [CCP ConfNak id=0x2 <mppe +H -M +S +L -D -C>]
pptpd: GRE: accepting packet #5
pptpd: GRE: accepting packet #6
pppd: rcvd [IPCP ConfReq id=0x3 <addr 0.0.0.0> <ms-dns1 0.0.0.0>
            <ms-wins 0.0.0.0> <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
pptpd: GRE: accepting packet #7
pppd: sent [IPCP TermAck id=0x3]
pppd: rcvd [CCP ConfNak id=0x1 <mppe +H -M +S -L -D -C>]
pppd: sent [CCP ConfReq id=0x2 <mppe +H -M +S -L -D -C>]
pppd: rcvd [CCP ConfReq id=0x4 <mppe +H -M +S -L -D -C>]
pppd: sent [CCP ConfAck id=0x4 <mppe +H -M +S -L -D -C>]
pptpd: GRE: accepting packet #8
pppd: rcvd [CCP ConfAck id=0x2 <mppe +H -M +S -L -D -C>]
pppd: MPPE 128-bit stateless compression enabled
pppd: sent [IPCP ConfReq id=0x1 <addr 192.168.0.1>]
pppd: sent [IPV6CP ConfReq id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
pptpd: GRE: accepting packet #9
pppd: rcvd [IPCP ConfAck id=0x1 <addr 192.168.0.1>]
pptpd: GRE: accepting packet #10
pppd: rcvd [IPV6CP ConfAck id=0x1 <addr fe80::0206:5bff:fedd:c5c3>]
pptpd: GRE: accepting packet #11
pppd: rcvd [IPCP ConfReq id=0x5 <addr 0.0.0.0>
            <ms-dns1 0.0.0.0> <ms-wins 0.0.0.0>
            <ms-dns3 0.0.0.0> <ms-wins 0.0.0.0>]
pppd: sent [IPCP ConfRej id=0x5 <ms-wins 0.0.0.0> <ms-wins 0.0.0.0>]
pptpd: GRE: accepting packet #12
pppd: rcvd [IPV6CP ConfReq id=0x6 <addr fe80::1cfc:fddd:8e2c:e118>]
pppd: sent [IPV6CP ConfAck id=0x6 <addr fe80::1cfc:fddd:8e2c:e118>]
pppd: local LL address fe80::0206:5bff:fedd:c5c3
pppd: remote LL address fe80::1cfc:fddd:8e2c:e118
pptpd: GRE: accepting packet #13
pppd: rcvd [IPCP ConfReq id=0x7 <addr 0.0.0.0>
            <ms-dns1 0.0.0.0> <ms-dns3 0.0.0.0>]
pppd: sent [IPCP ConfNak id=0x7 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
pptpd: GRE: accepting packet #14
pppd: rcvd [IPCP ConfReq id=0x8 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
pppd: sent [IPCP ConfAck id=0x8 <addr 192.168.1.1>
            <ms-dns1 192.168.0.1> <ms-dns3 192.168.0.1>]
pppd: local IP address 192.168.0.1
pppd: remote IP address 192.168.1.1
pptpd: GRE: accepting packet #15
pptpd: CTRL: Sending ECHO REQ id 1
pptpd: CTRL: Made a ECHO REQ packet
pptpd: CTRL: I wrote 16 bytes to the client.
pptpd: CTRL: Sent packet to client
```

Widoczne jest podobieństwo do poprzedniego raportu, widzimy tu jednak komunikaty wypisywane przez dwa procesy, czyli pppd i pptpd, ustanawiające sesję PPTP na serwerze. Odebranie przez pptpd komunikatu typu 1 oznacza żądanie klienta nawiązania połączenia kontrolnego (PPTP utrzymuje dwa połączenia, przeznaczone do transmisji danych i transmisji poleceń sterujących; drugie z wymienionych nawiązywane jest najpierw). Po odpo-

wiedzi na to żądanie od klienta przychodzi następne (komunikat typ 7) oznaczające nawiązanie zdalnego połączenia. Maksymalna szybkość transmisji w ramach tego połączenia określona zostaje na bardzo wysokim poziomie (100 000 000 b/s), co oznacza praktycznie brak ograniczenia. *Okno*, którego szerokość określa klient na 64, jest koncepcją stosowaną głównie w protokołach transportowych (takich jak TCP — patrz rozdział 15.), stanowiącą element sterowania przepływem. PPTP wykorzystuje w swych ramkach pola *Numer sekwencyjny* i *Numer potwierdzenia* w celu określenia, ile z tych ramek pomyślnie dotarło do miejsca swego przeznaczenia. Jeśli zbyt wiele z nich ulega odrzuceniu, nadawca powinien zwolnić tempo wysyłania. Dla określenia właściwego limitu czasu oczekiwania na potwierdzenie dotarcia wysłanej ramki do celu, PPTP wykorzystuje adaptacyjny mechanizm bazujący na długości czasu potrzebnego do przejścia ramki przez łącze. Przy okazji omawiania protokołu TCP zajmiemy się szczegółami podobnego mechanizmu.

Krótko po zdefiniowaniu okna swą pracę rozpoczyna proces `pppd` — rozpoczyna się scenariusz przetwarzania komunikatów protokołu PPP, scenariusz podobny do prezentowanego poprzednio. Jest między nimi jedna istotna różnica: proces `pptpd` pełni rolę pośrednika w przekazywaniu pakietów do procesu `pppd` i z niego, posiłkując się przy tym własnymi komunikatami (m.in. `set link info` i `echo-request`). Przedstawiony scenariusz jest w istocie ilustracją tego, jak protokół PPTP spełnia rolę agenta tunelowania GRE w odniesieniu do pakietów PPP; tunelowanie to umożliwia używanie bez żadnych zmian istniejących implementacji PPP (tu: procesu `pppd`). Oryginalnie protokół GRE wykorzystuje w charakterze kontenerów pakiety IPv4, podobną funkcjonalność można uzyskać również przy użyciu pakietów IPv6 (co opisane zostało w dokumencie [RFC2473]).

3.9.1. Łącza jednokierunkowe

Interesujący przypadek w kontekście warstwy łącza danych stanowią **łącza jednokierunkowe** (*Unidirectional Links* — UDL), czyli zdolne przenosić informację tylko w jednym kierunku. Większość opisywanych do tej pory protokołów nie będzie na takich łączach funkcjonować z oczywistego powodu — opierają się one na transmisji dwukierunkowej, czego przykładem są chociażby komunikaty konfiguracyjne PPP. Wyjściem z tej sytuacji jest zasymulowanie łącza dwukierunkowego, przy użyciu oryginalnego łącza jednokierunkowego oraz tunelu utworzonego na bazie drugiego, wspomagającego interfejsu sieciowego (koncepcję tę opisano w dokumencie [RFC3077]. Takie właśnie rozwiązanie funkcjonowało we wczesnych stadiach satelitarnego dostępu do Internetu, gdzie łącze satelitarne prowadzące do użytkownika (*downstream*) uzupełniane było połączeniem modemowym prowadzącym w kierunku przeciwnym (*upstream*), którego pakiety enkapsulowane były za pomocą protokołu GRE w pakietach IP. W sytuacji gdy użytkownik preferuje pobieranie plików, rzadziej korzystając z ich wysyłania, rozwiązanie takie spisuje się całkiem nieźle.

W celu automatycznego skonfigurowania tunelu po stronie odbiorcy skonstruowano **protokół dynamicznego konfigurowania tuneli** (*Dynamic Tunnel Configuration Protocol*, w skrócie DTCP). W ramach tego protokołu do potencjalnych odbiorców rozsyłane są komunikaty multicast *Hello*, zawierające informację o istnieniu łącza UDL oraz adresie IP i adresie MAC węzła nadającego na tym łączu. Komunikaty te zawierają także listę

węzłów końcowych tunelu osiągalnych przez wspomniany interfejs pomocniczy. Gdy użytkownik wybierze jeden z tych węzłów, DTCP organizuje tunelowanie ruchu powrotnego przez enkapsulowanie jego pakietów w pakietach IP za pomocą GRE. Zadaniem dostawcy usługi jest wydobywanie z otrzymywanego strumienia tunelowanych ramek warstwy 2. (zwykle są to ramki ethernetowe) i forwardowanie ich zgodnie z przeznaczeniem. Tak więc, podczas gdy po stronie dostawcy wymagane jest ręczne konfigurowanie kanałów przez dostawcę usługi, po stronie użytkowników konfigurowanie to jest zautomatyzowane przy użyciu protokołu DTCP.

Mimo iż opisana technika efektywnie skrywa przed wyższymi warstwami szczegóły emulowania dwukierunkowości łącza (i sam fakt istnienia emulacji), jednak asymetria wynikająca z różnych technologii transmisyjnych (i w konsekwencji różnych szybkości transmisji) w każdym z kierunków jest dla protokołów tych warstw zauważalna z określonymi konsekwencjami (dyskutowanymi szczegółowo w dokumencie [RFC3449]).

Cytowany przykład z łącznością satelitarną ilustruje podstawowy problem tunelowania, jakim jest pracochłonność konfigurowania tuneli. Zabiegi konfiguracyjne wymagają wyboru punktów końcowych tunelu i skonfigurowania urządzeń przyłączonych do tych punktów końcowych, poinformowania tychże urządzeń o adresach IP partnerów, a także dostarczenia niezbędnych informacji uwierzytelniających oraz skoordynowania pracy różnych protokołów. Tradycyjnie wykonywano tę niewdzięczną pracę ręcznie, nic więc dziwnego, że z biegiem czasu sukcesywnie pojawiały się narzędzia umożliwiające jej automatyzację (w różnym stopniu). Tunelowanie znalazło zastosowanie m.in. w skomplikowanym procesie „przestrajania" istniejących sieci IPv4 na nowy protokół IPv6: w ramach mechanizmu o nazwie 6to4 (patrz [RFC3056]) pakiety IPv6 enkapsulowane były w pakietach IPv4 i mogły w tej otoczce z powodzeniem podróżować przez sieć, która nie interpretowała nowego protokołu. Ten wygodny mechanizm sprawiał jednak problemy w sieciach NAT (patrz rozdział 7.), popularnych zwłaszcza w zastosowaniach domowych. Idea przesyłania pakietów IPv6 w automatycznie konfigurowanych tunelach znalazła urzeczywistnienie w mechanizmie o nazwie *Teredo* (opisywanym w [RFC4380]) — pakiety IPv6 podróżują tam ukryte w datagramach UDP, które z kolei transmitowane są za pomocą datagramów IPv4. Zrozumienie szczegółów tej technologii wymaga pewnej wiedzy na temat samych pakietów IPv4, IPv6 i UDP, odkładamy więc szczegółową dyskusję na ten temat do rozdziału 10.

3.10. Ataki na warstwę łącza danych

Ataki na sieci TCP/IP, przypuszczane z poziomu poniżej warstwy transportowej (TCP) i sieciowej (IP), są o tyle podstępne, że wiele informacji przetwarzanej w warstwie łącza danych nie jest współdzielonych ze wspomnianymi warstwami wyższymi, trudno więc wykrywać i neutralizować ingerencję w tę informację. Mimo to, wiele z ataków tej kategorii zostało dobrze zrozumianych i w tym podrozdziale opiszemy kilka z nich, choćby po to, by uzmysłowić czytelnikom problemy, jakie ataki te mogą powodować w funkcjonowaniu warstw wyższych.

Interfejs konwencjonalnego Ethernetu może pracować w **trybie nasłuchiwania** (*promiscuous mode*), który umożliwia temu interfejsowi odbieranie *wszystkich* ramek, również tych do niego nieadresowanych. We wczesnym stadium rozwoju Ethernetu, gdy medium

transmisyjne było po prostu współdzielonym kablem, cecha ta umożliwiała każdemu komputerowi podłączonemu do tegoż kabla „podsłuchiwanie" ruchu i analizowanie zawartości ramek adresowanych do innych komputerów. A zawartość ta była często niezwykle interesująca: poufne informacje w rodzaju loginów czy haseł dostępne były w postaci łatwo zauważalnej, bo formowanej przez ciągi „drukowalnych" znaków ASCII. Na przeszkodzie tej inwigilacji stanęły dwa główne czynniki: wynalezienie przełączników i wykorzystanie kryptografii w warstwach wyższych. Na porcie przełącznika pojawia się informacja przeznaczona wyłącznie dla komputera, do którego port ten jest przyłączony — tak, oczywiście, pojawiają się tam również ramki transmisji broadcast i multicast, lecz jest raczej mało prawdopodobne, by zawierały jakąś „wrażliwą" czy poufną informację. Poważniejszą przeszkodą jest jednak szyfrowanie treści wymienianych przez warstwy wyższe, nawet bowiem sprytne przechwycenie informacji nie na wiele się zdaje: informacja ta jest nieczytelna, jeśli nie zna się odpowiedniego klucza.

Inny typ ataków wymierzony jest bezpośrednio w przełączniki, a dokładniej — w ich tablice trasowania. Jak pamiętamy, w tablicy takiej znajduje się lista hostów powiązanych z portami, za pośrednictwem których hosty te są (z danego przełącznika) osiągalne. Jeśli uda się wymusić szybkie zapełnianie tych tablic (np. za pomocą maskarady udającej mnogość różnych stacji), to — jak łatwo sobie wyobrazić — przełącznik zmuszony będzie do usuwania ich dotychczasowej zawartości (by zrobić miejsce dla nowych pozycji), co skutkować będzie przerywaniem obsługi legalnych użytkowników. Odmiana tego ataku wykorzystuje specyfikę algorytmu STP: atakująca stacja doprowadza do tego, że niektóre przełączniki uznają ją za przełącznik znajdujący się na najkrótszej trasie do korzenia drzewa rozpinającego — i kierują do niej cały ruch.

Wraz z pojawieniem się sieci Wi-Fi techniki podsłuchiwania i maskarady zyskały nowy, groźniejszy wymiar, jako że każda stacja może przełączyć się w tryb monitorowania i wyławiać z wszechotaczającego widma fal radiowych pakiety sieciowe (choć przełączenie interfejsu 802.11 w tryb monitorowania wydaje się nieco trudniejsze niż ustawienie interfejsu ethernetowego w tryb nasłuchiwania, bo wymaga manipulowania raczej sterownikiem urządzenia niż samym fizycznym urządzeniem). Najpopularniejszą techniką wczesnych „ataków" na sieci Wi-Fi (piszemy w cudzysłowie, bo w świetle ówczesnych uregulowań prawnych nie zawsze tak dawały się zakwalifikować) było krążenie po okolicy i skanowanie otoczenia w wyszukiwaniu punktów dostępowych (tzw. „jazda wojenna" — *wardriving*). Mimo iż wiele z tych punktów zabezpieczonych było kryptograficznie przed nieuprawnionym dostępem, zdarzało się mnóstwo niezabezpieczonych bądź ukierunkowanych na tzw. **portale przechwytujące** (*capturing portals*) — kandydat na klienta kierowany jest przez punkt dostępowy do strony rejestracyjnej/logowania, po pomyślnym „przejściu" której transmisja z klientem filtrowana jest na podstawie adresu MAC jego karty sieciowej. Przez „podłożenie" odpowiednio spreparowanej strony logowania w odpowiednim momencie klient-intruz uwierzytelniany był w imieniu legalnego użytkownika.

Ataki bardziej wyrafinowane kierowane były na kryptograficzne zabezpieczenia punktów dostępowych. Stosowany wówczas powszechnie system WEP (patrz [BHL06]) okazał się mało odporny na łamanie — za sprawą zbyt krótkiego klucza, lecz także zbyt słabego algorytmu szyfrowania i innych błędów projektowych — co zmusiło IEEE do opracowania bardziej solidnego następnika. Nowe algorytmy WPA i późniejszy WPA2 okazały się bardziej odporne, a WEP, choć nadal stosowany, uważany jest dziś za (ryzykowny) przeżytek.

Skompromitowanie łącza PPP uwarunkowane jest łatwością dostępu intruza do nośnika oraz wykorzystywanymi protokołami: jeżeli poufne informacje przesyłane są w postaci niezaszyfrowanej (jak np. w protokole PAP), intruz może je łatwo przechwycić i użyć w celu „podszycia się" pod legalnego użytkownika. Zważywszy na potencjalnie wrażliwy charakter informacji, jaką ramki PPP przenosić mogą na rzecz protokołów warstw wyższych (może to być np. informacja związana z trasowaniem), nietrudno przewidzieć fatalne tego konsekwencje dla całej sieci.

Gdy spojrzy się na tunelowanie w kontekście ataków sieciowych, może ono spełniać rolę zarówno celu, jak i narzędzia. Ponieważ tunele prowadzą przez sieć — często jest nią Internet — przesyłane przez nie treści mogą być przechwytywane i analizowane. Ataki mogą być skierowane także na punkty końcowe tunelu bądź to w postaci prób zmiany ich konfiguracji, bądź też przez próby tworzenia nowych kanałów w liczbie przekraczającej zdolności urządzenia, co niechybnie przybiera formę ataku przeciążeniowego (DoS). Jeżeli uda się odpowiednio zmienić konfigurację punktu końcowego, następnym etapem może być udane utworzenie nieautoryzowanego tunelu, co w przypadku np. protokołu L2TP tworzy — niezależne od konkretnego protokołu —„tylne drzwi" dostępu do chronionej sieci. I w tym momencie tunelowanie staje się narzędziem w ręku intruza: znane są przypadki, gdy protokół GRE użyty został do „wstrzykiwania" w pasożytniczy tunel pakietów, które po dotarciu do chronionej sieci traktowane były tak, jakby wysyłane były z jej wnętrza.

3.11. Podsumowanie

Zakończony właśnie rozdział poświęcony był najniższej warstwie modelu odniesienia TCP/IP — warstwie łącza danych. Rozpoczęliśmy od przedstawienia ewolucji Ethernetu zarówno pod względem sukcesywnie wzrastającej prędkości transmisji — od 10 Mb/s do 10 Gb/s i więcej — jak i pod względem coraz większego bogactwa oferowanych możliwości: sieci VLAN, priorytetów transmisji czy agregowania łączy i związanego z tymi możliwościami rozwoju formatu ramek. Omówiliśmy szczegółowo funkcje przełączników i mostków, implementujących elektroniczne ścieżki łączące niezależne zespoły stacji, wyjaśniliśmy także różnice między transmisją półdupleksową a pełnodupleksową. Sporą część rozdziału poświęciliśmy sieciom bezprzewodowym Wi-Fi standardu 802.11, wykorzystującym nielicencjonowane pasma 2,4 GHz i 5 GHz, zwróciliśmy jednocześnie uwagę na istotne ich podobieństwa i różnice w stosunku do „klasycznego" Ethernetu. Zajęliśmy się też pobieżnie problemem zabezpieczenia tychże sieci, szczególnie pierwotnym protokołem WEP i jego silniejszymi następcami WPA i WPA2. Omówiliśmy też koncepcje wspólne dla różnych łączy, m.in. sens maksymalnej jednostki transmisji dla protokołu (MTU) i dla ścieżki między dwoma węzłami (PMTU).

Wychodząc poza standardy IEEE, omówiliśmy także protokół PPP wykorzystywany do przenoszenia różnego rodzaju pakietów — głównie TCP/IP, lecz także innego rodzaju — w ramkach wzorowanych na protokole HDLC, na bazie różnego typu nośników fizycznych, od powolnych połączeń modemowych do superszybkich łączy światłowodowych. Protokół PPP to w istocie cały zestaw protokołów, związanych m.in. z szyfrowaniem, kompresją, uwierzytelnianiem i agregowaniem łączy w wiązki. Ponieważ łącze PPP przebiega między dwoma konkretnymi punktami, wolne jest od problematyki szeregowania dostępu do nośnika, nieodłącznie związanej z protokołami MAC w Ethernecie i Wi-Fi.

Większość implementacji TCP/IP oferuje opcję pętli zwrotnej, identyfikowanej zarówno przez specjalny adres (127.0.0.1 lub ::1, zależnie od wersji protokołu IP), jak i własny adres IP odnośnego hosta. Pakiety pętli zwrotnej, niezależnie od faktu, że nigdy nie opuszczają macierzystego komputera, poddawane są kompletnej obróbce na równi z „zewnętrznymi" pakietami, w warstwie transportowej i warstwie IP.

Później zajęliśmy się użyteczną techniką tunelowania, czyli przenoszenia pakietów danej warstwy w otoczkach utworzonych z pakietów warstwy wyższej lub równorzędnej (a więc niejako na przekór koncepcji multipleksowania pakietów w architekturze warstwowej). Technika ta umożliwia wykorzystywanie Internetu (lub innych sieci połączeniowych) do transmisji pakietów związanych z innym poziomem infrastruktury sieciowej (a być może także zupełnie odmienną infrastrukturą). Technika tunelowania okazała się (i okazuje się nadal) pożyteczna w dostrajaniu sieci opartych na IPv4 do nowych warunków protokołu IPv6: nieinterpretowane jeszcze pakiety IPv6 transmitowane są w otoczkach pakietów IPv4. Tunelowanie wykorzystywane jest także jako podstawa konstrukcji prywatnych sieci wirtualnych (VPN).

Rozdział zakończyliśmy krótkim omówieniem potencjalnych ataków, na jakie narażone są sieci komputerowe na poziomie warstwy łącza danych, której elementy stanowić mogą zarówno cel wspomnianych ataków, jak i narzędzia do ich realizacji. Wiele z tych ataków ogranicza się do biernego podsłuchiwania, zmierzającego do przechwycenia poufnych informacji, wiele ma jednak charakter czynny, polegający na ingerencji w strukturę przesyłanych treści, a w konsekwencji uzyskiwanie nieautoryzowanego dostępu do hostów i przełączników, ich paraliżowanie (za pomocą ataków DoS) bądź fałszowanie przetwarzanej przez nie informacji. Wiele z tych ataków ma bardzo subtelny charakter, związany z detalami algorytmów wykorzystywanych przez przełączniki i routery, m.in. z algorytmem STP. Zagrożenie opisywanymi atakami, i tak już poważne w dobie przewodowego Ethernetu, nabrało nowego wymiaru wraz z pojawieniem się sieci bezprzewodowych.

Ograniczone ramy książki pozwoliły nam na opisanie jedynie wybranych technologii łącza danych, najczęściej stosowanych obecnie w odniesieniu do sieci TCP/IP. Jedną z przyczyn niesamowitej popularności zestawu protokołów TCP/IP jest jego zdolność do funkcjonowania na bazie łączy realizowanych w dowolnej niemal technologii. Protokół IP wymaga jedynie istnienia ścieżki między nadawcą a odbiorcą, nawet jeśli jest to ścieżka w postaci kaskady różnych łączy pośredniczących. I choć jest to wymaganie nader skromne, to prowadzone są badania zmierzające do jego osłabienia — tak by wspomniana ścieżka nie musiała istnieć w całości w danej chwili, lecz mogła być budowana dynamicznie w czasie. W dokumencie [RFC4838] opisano zastosowanie tej koncepcji do komunikacji na dystansach międzyplanetarnych, koniecznej w procesie badania kosmosu.

3.12. Bibliografia

[802.11-2007] *IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, czerwiec 2007.

[802.11n-2009] *IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 5: Enhancements for Higher Throughput*, październik 2009.

[802.11y-2008] *IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 3: 3650-3700 MHz Operation in USA*, listopad 2009.

[802.16-2009] *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, maj 2009.

[802.16h-2010] *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 2: Improved Coexistence Mechanisms for License-Exempt Operation*, lipiec 2010.

[802.16j-2009] *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 1: Multihop Relay Specification*, czerwiec 2009.

[802.16k-2007] *IEEE Standard for Local and Metropolitan Area Networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems Amendment 5: Bridging of IEEE 802.16*, sierpień 2010.

[802.1AK-2007] *IEEE Standard for Local and Metropolitan Area Networks, Virtual Bridged Local Area Networks Amendment 7: Multiple Registration Protocol*, czerwiec 2007.

[802.1AE-2006] *IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Security*, sierpień 2006.

[802.1ak-2007] *IEEE Standard for Local and Metropolitan Area Networks — Virtual Bridged Local Area Networks—Amendment 7: Multiple Registration Protocol*, czerwiec 2007.

[802.1AX-2008] *IEEE Standard for Local and Metropolitan Area Networks — Link Aggregation*, listopad 2008.

[802.1D-2004] *IEEE Standard for Local and Metropolitan Area Networks Media Access Control (MAC) Bridges*, czerwiec 2004.

[802.1Q-2005] *IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks*, maj 2006.

[802.1X-2010] *IEEE Standard for Local and Metropolitan Area Networks Port-Based Network Access Control*, luty 2010.

[802.2-1998] *IEEE Standard for Local and Metropolitan Area Networks Logical Link Control* (także ISO/IEC 8802-2:1998), maj 1998.

[802.21-2008] *IEEE Standard for Local and Metropolitan Area Networks, Part 21: Media Independent Handover Services*, styczeń 2009.

[802.3-2008] *IEEE Standard for Local and Metropolitan Area Networks, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, grudzień 2008.

[802.3at-2009] *IEEE Standard for Local and Metropolitan Area Networks — Specific Requirements, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment 3: Date Terminal Equipment (DTE) Power via the Media Dependent Interface (MDI) Enhancements*, październik 2009.

[802.3ba-2010] *IEEE Standard for Local and Metropolitan Area Networks, Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications, Amendment 4: Media Access Control Parameters, Physical Layers, and Management Parameters for 40Gb/s and 100Gb/s Operation*, czerwiec 2010.

[802.11n-2009] *IEEE Standard for Local and Metropolitan Area Networks, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 5: Enhancements for Higher Throughput*, październik 2009.

[AES01] U.S. National Institute of Standards and Technology, FIPS PUB 197 *Advanced Encryption Standard*, listopad 2001.

[BHL06] A. Bittau, M. Handley, J. Lackey *The Final Nail in WEP's Coffin*, Proc. IEEE Symposium on Security and Privacy, maj 2006.

[BOND] *http://bonding.sourceforge.net*

[ETHERTYPES] *http://www.iana.org/assignments/ieee-802-numbers*

[ETX] D. De Couto, D. Aguayo, J. Bicket, R. Morris, *A High-Throughput Path Metric for Multi-Hop Wireless Routing*, Proc. Mobicom, wrzesień 2003.

[G704] ITU, *General Aspects of Digital Transmission Systems: Synchronous Frame Structures Used at 1544, 6312, 2048k, 8488, and 44736 kbit/s Hierarchical Levels*, ITU-T Recommendation G.704, lipiec 1995.

[IANA-CHARSET] *Character Sets*, *http://www.iana.org/assignments/character-sets*

[ISO3309] International Organization for Standardization, *Information Processing Systems — Data Communication High-Level Data Link Control Procedure — Frame Structure*, IS 3309, 1984.

[ISO4335] International Organization for Standardization, *Information Processing Systems — Data Communication High-Level Data Link Control Procedure — Elements of Procedure*, IS 4335, 1987.

[JF] M. Mathis, *Raising the Internet MTU*, *http://www.psc.edu/~mathis/MTU*

[MWLD] *Long Distance Links with MadWiFi*, *http://madwifi-project.org/wiki/UserDocs/LongDistance*

[PPPn] *http://www.iana.org/assignments/ppp-numbers*

[RFC0894] C. Hornig, *A Standard for the Transmission of IP Datagrams over Ethernet Networks*, Internet RFC 0894/STD 0041, kwiecień 1984.

[RFC1042] J. Postel, J. Reynolds, *Standard for the Transmission of IP Datagrams over IEEE 802 Networks*, Internet RFC 1042/STD 0043, luty 1988.

[RFC1144] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, Internet RFC 1144, luty 1990.

[RFC1191] J. Mogul, S. Deering, *Path MTU Discovery*, Internet RFC 1191, listopad 1990.

[RFC1332] G. McGregor, *The PPP Internet Protocol Control Protocol*, Internet RFC 1332, maj 1992.

[RFC1570] W. Simpson (red.), *PPP LCP Extensions*, Internet RFC 1570, styczeń 1994.

[RFC1661] W. Simpson, *The Point-to-Point Protocol (PPP)*, Internet RFC 1661/STD 0051, lipiec 1994.

[RFC1662] W. Simpson (red.), *PPP in HDLC-like Framing*, Internet RFC 1662/STD 0051, lipiec 1994.

[RFC1663] D. Rand, *PPP Reliable Transmission*, Internet RFC 1663, lipiec 1994.

[RFC1853] W. Simpson, *IP in IP Tunneling*, Internet RFC 1853 (informational), październik 1995.

[RFC1962] D. Rand, *The PPP Compression Protocol (CCP)*, Internet RFC 1962, czerwiec 1996.

[RFC1977] V. Schryver, *PPP BSD Compression Protocol*, Internet RFC 1977 (informational), sierpień 1996.

[RFC1981] J. McCann, S. Deering, *Path MTU Discovery for IP Version 6*, Internet RFC 1981, sierpień 1996.

[RFC1989] W. Simpson, *PPP Link Quality Monitoring*, Internet RFC 1989, sierpień 1996.

[RFC1990] K. Sklower, B. Lloyd, G. McGregor, D. Carr, T. Coradetti, *The PPP Multilink Protocol (MP)*, Internet RFC 1990, sierpień 1996.

[RFC1994] W. Simpson, *PPP Challenge Handshake Authentication Protocol (CHAP)*, Internet RFC 1994, sierpień 1996.

[RFC2118] G. Pall, *Microsoft Point-to-Point (MPPC) Protocol*, Internet RFC 2118 (informational), marzec 1997.

[RFC2125] C. Richards, K. Smith, *The PPP Bandwidth Allocation Protocol (BAP)/The PPP Bandwidth Allocation Control Protocol (BACP)*, Internet RFC 2125, marzec 1997.

[RFC2153] W. Simpson, *PPP Vendor Extensions*, Internet RFC 2153 (informational), maj 1997.

[RFC2290] J. Solomon, S. Glass, *Mobile-IPv4 Configuration Option for PPP IPCP*, Internet RFC 2290, luty 1998.

[RFC2464] M. Crawford, *Transmission of IPv6 Packets over Ethernet Networks*, Internet RFC 2464, grudzień 1988.

[RFC2473] A. Conta, S. Deering, *Generic Packet Tuneling in IPv6 Specification*, Internet RFC 2473, grudzień 1998.

[RFC2484] G. Zorn, *PPP LCP Internationalization Configuration Option*, Internet RFC 2484, styczeń 1999.

[RFC2507] M. Degermark, B. Nordgren, S. Pink, *IP Header Compression*, Internet RFC 2507, luty 1999.

[RFC2615] A. Malis, W. Simpson, *PPP over SONET/SDH*, Internet RFC 2615, czerwiec 1999.

[RFC2637] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn, *Point-to-Point Tunneling Protocol (PPTP)*, Internet RFC 2637 (informational), lipiec 1999.

[RFC2759] G. Zorn, *Microsoft PPP CHAP Extensions, Version 2*, Internet RFC 2759 (informational), styczeń 2000.

[RFC2784] D. Farinacci, T. Li, S. Hanks, D. Meyer, P. Traina, *Generic Routing Encapsulation (GRE)*, Internet RFC 2784, marzec 2000.

[RFC2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, *Remote Authentication Dial In User Service (RADIUS)*, Internet RFC 2865, czerwiec 2000.

[RFC2890] G. Dommety, *Key and Sequence Number Extensions to GRE*, Internet RFC 2890, wrzesień 2000.

[RFC3056] B. Carpenter, K. Moore, *Connection of IPv6 Domains via IPv4 Clouds*, Internet RFC 3056, luty 2001.

[RFC3077] E. Duros, W. Dabbous, H. Izumiyama, N. Fujii, Y. Zhang, *A Link-Layer Tunneling Mechanism for Unidirectional Links*, Internet RFC 3077, marzec 2001.

[RFC3078] G. Pall, G. Zorn, *Microsoft Point-to-Point Encryption (MPPE) Protocol*, Internet RFC 3078 (informational), marzec 2001.

[RFC3153] R. Pazhyannur, I. Ali, C. Fox, *PPP Multiplexing*, Internet RFC 3153, sierpień 2001.

[RFC3366] G. Fairhurst, L. Wood, *Advice to Link Designers on Link Automatic Repeat reQuest (ARQ)*, Internet RFC 3366/BCP 0062, sierpień 2002.

[RFC3449] H. Balakrishnan, V. Padmanabhan, G. Fairhurst, M. Sooriyabandara, *TCP Performance Implications of Network Path Asymmetry*, Internet RFC 3449/BCP 0069, grudzień 2002.

[RFC3544] T. Koren, S. Casner, C. Bormann, *IP Header Compression over PPP*, Internet RFC 3544, lipiec 2003.

[RFC3561] C. Perkins, E. Belding-Royer, S. Das, *Ad Hoc On-Demand Distance Vector (AODV) Routing*, Internet RFC 3561 (experimental), lipiec 2003.

[RFC3610] D. Whiting, R. Housley, N. Ferguson, *Counter with CBC-MAC (CCM)*, Internet RFC 3610 (informational), wrzesień 2003.

[RFC3626] T. Clausen, P. Jacquet (red.), *Optimized Link State Routing Protocol (OLSR)*, Internet RFC 3626 (experimental), październik 2003.

[RFC3748] B. Aboba i in., *Extensible Authentication Protocol (EAP)*, Internet RFC 3748, czerwiec 2004.

[RFC3931] J. Lau, M. Townsley, I. Goyret (red.), *Layer Two Tunneling Protocol — Version 3 (L2TPv3)*, Internet RFC 3931, marzec 2005.

[RFC4017] D. Stanley, J. Walker, B. Aboba, *Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs*, Internet RFC 4017 (informational), marzec 2005.

[RFC4380] C. Huitema, *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)*, Internet RFC 4380, luty 2006.

[RFC4647] A. Phillips, M. Davis, *Matching of Language Tags*, Internet RFC 4647/BCP 0047, wrzesień 2006.

[RFC4821] M. Mathis, J. Heffner, *Packetization Layer Path MTU Discovery*, Internet RFC 4821, marzec 2007.

[RFC4838] V. Cerf I in. *Delay-Tolerant Networking Architecture*, Internet RFC 4838 (informational), kwiecień 2007.

[RFC4840] B. Aboba (red.), E. Davies, D. Thaler, *Multiple Encapsulation Methods Considered Harmful*, Internet RFC 4840 (informational), kwiecień 2007.

[RFC5072] S. Varada (red.), D. Haskins, E. Allen, *IP Version 6 over PPP*, Internet RFC 5072, wrzesień 2007.

[RFC5225] G. Pelletier, K. Sandlund, *RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP, and UDP-Lite*, Internet RFC 5225, kwiecień 2008.

[RFC5646] A. Phillips, M. Davis (red.), *Tags for Identifying Languages*, Internet RFC 5646/BCP 0047, wrzesień 2009.

[S08] D. Skordoulis i in., *IEEE 802.11n MAC Frame Aggregation Mechanisms for Next-Generation High-Throughput WLANs*, „IEEE Wireless Communications", luty 2008.

[S96] B. Schneier, *Applied Cryptography, Second Edition* (John Wiley & Sons, 1996).

[SAE] D. Harkins, *Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks*, Proc. SENSORCOMM, sierpień 2008.

[SC05] S. Shalunov, R. Carlson, *Detecting Duplex Mismatch on Ethernet*, Proc. Passive and Active Measurement Workshop, marzec 2005.

[SHK07] C. Sengul, A. Harris, R. Kravets, *Reconsidering Power Management*, Invited Paper, Proc. IEEE Broadnets, 2007.

[WOL] *http://wake-on-lan.sourceforge.net*

Rozdział 4.
Protokół ARP

4.1. Wprowadzenie

Jak wiadomo, protokół IP zaprojektowany został jako narzędzie komutacji pakietów w sieciach o różnorodnej charakterystyce fizycznej. Wymaga to — oprócz wielu innych rzeczy — mechanizmu odwzorowującego adresy używane przez oprogramowanie warstwy sieciowej na adresy interpretowane przez sprzęt. Ogólnie rzecz biorąc, z każdym interfejsem sieciowym związany jest nierozerwalnie jeden konkretny adres sprzętowy (np. 48-bitowy adres Ethernetu lub standardu 802.11); ramki wymieniane między interfejsami muszą zawierać poprawne adresy sprzętowe, w przeciwnym razie nie będą dostarczane do miejsc swego przeznaczenia. Dla odmiany, węzły sieci IPv4 identyfikowane są za pomocą 32-bitowych adresów, opisywanych w rozdziale 2.; adresy te nie są w żaden sposób „zrozumiałe" dla sprzętu, wyrażane są bowiem w kategoriach oprogramowania protokołu IP. Oprogramowanie systemu operacyjnego (np. sterownik sieci Ethernet) musi ostatecznie posługiwać się wspomnianymi adresami fizycznymi. „Przeliczaniem" adresów IPv4 na adresy sprzętowe zajmuje się protokół o nazwie (po prostu) *protokół odwzorowania adresów* (*Address Resolution Protocol*, w skrócie ARP), opisany w dokumencie [RFC0826]; sieci IPv6 nie korzystają z tego protokołu — przeliczanie adresów IPv6 na adresy sprzętowe jest zadaniem *protokołu wykrywania sąsiadów* (NDP — *Network Discovery Protocol*), stanowiącego element składowy protokołu ICMPv6 (o czym piszemy w rozdziale 8.).

Należy zwrócić uwagę na oczywisty skądinąd fakt, że adresy warstwy sieciowej mają zupełnie inną genezę niż adresy wykorzystywane w wystawie łącza danych. Adres sprzętowy przydzielany jest urządzeniu (np. karcie sieciowej) przez producenta i zostaje trwale związany z tym urządzeniem, np. przez zapisanie w jego pamięci stałej. Ramka kierowana do takiego urządzenia musi zawierać jego adres sprzętowy w polu adresu docelowego. Dla odmiany, adres IP przydzielany jest węzłowi sieci przez jej oprogramowanie, z inicjatywy użytkownika lub administratora, na podstawie dostępnej puli takich adresów (patrz rozdział 6.). Adres IP może zmieniać się w czasie, co jest zjawiskiem powszechnym w urządzeniach mobilnych.

Wspomniany protokół ARP jest protokołem generycznym w tym sensie, że realizowane przez niego odwzorowanie może być wykonywane między różnymi przestrzeniami adresowymi; prawie zawsze jednak jest to przeliczanie adresów IPv4 na 48-bitowe adresy ethernetowe (jemu właśnie poświęcony jest wspomniany dokument [RFC0826]) i do tego

przypadku ograniczymy się w tym rozdziale. Będziemy przy tym zamiennie używać określeń „adres ethernetowy" i „adres MAC".

Protokół ARP funkcjonuje w sposób *dynamiczny* — odwzorowanie adresów dokonywane jest automatycznie, w miarę potrzeb, stosownie do zmieniającej się sytuacji, bez interwencji administratora. Jeżeli więc np. w danym hoście wymieniona zostanie karta sieciowa, zmieni się adres sprzętowy skojarzony z adresem tego hosta; protokół ARP zareaguje na tę zmianę prawidłowo, choć z pewnym opóźnieniem. Szczegóły tej operacji nie są istotne ani dla programisty aplikacyjnego, ani nawet dla administratora systemu.

Odwzorowanie adresów w drugą stronę — z ethernetowych na IPv4 — jest zadaniem protokołu RARP (*Reverse Address Resolution Protocol*). Wykorzystywane było głównie w systemach pozbawionych dysków, czyli m.in. terminalach X i bezdyskowych stacjach roboczych. Choć jest wygodnym sposobem statycznego przydzielania adresów IP urządzeniom (całkowicie pod kontrolą administratora), stosowane jest dziś raczej rzadko. Czytelników zainteresowanych tym tematem odsyłamy do dokumentu [RFC0903].

4.2. Przykład

Każdorazowo, gdy zamierzamy skorzystać z jakiejś usługi internetowej, np. wpisując URL w pole adresowe przeglądarki WWW, nasz komputer musi znaleźć sposób skontaktowania się z serwerem świadczącym żądaną usługę, w tym przypadku z serwerem WWW. Poszukiwanie tego sposobu rozpoczyna się od określenia, czy serwer ten funkcjonuje na odległym komputerze w innej sieci, czy też na komputerze zlokalizowanym w sieci (podsieci) tej samej, co nasz komputer. Przypadek pierwszy stanowi zadanie dla routera, w drugim wkracza do akcji protokół ARP.

Załóżmy, że jako docelowy adres WWW wpisaliśmy w przeglądarce

```
http://10.0.0.1
```

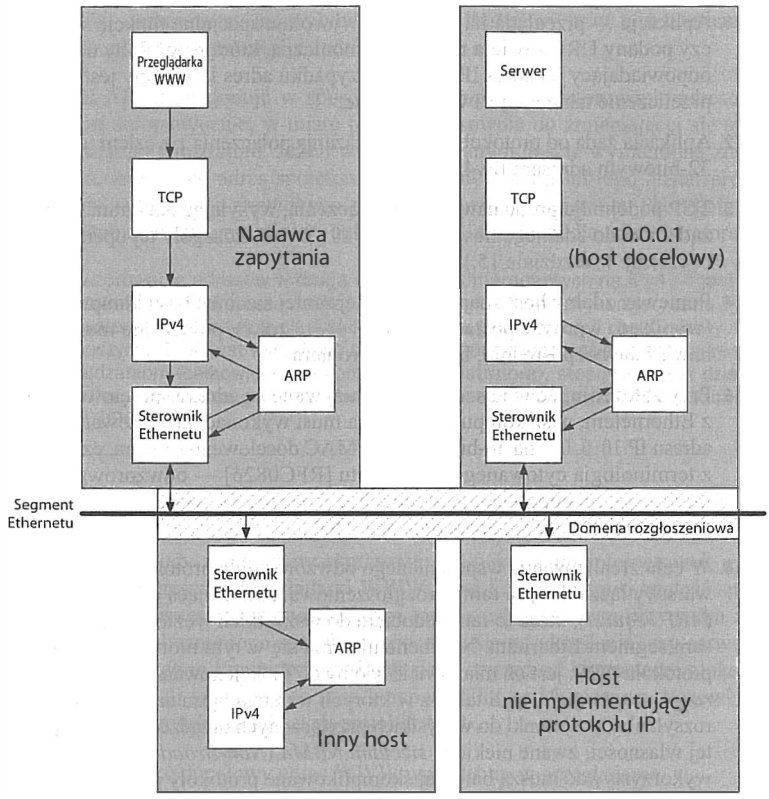
Trochę to nietypowe, użytkownicy częściej posługują się nazwami mnemonicznymi niż adresami IP. Chcieliśmy jednak uwydatnić fakt lokalności odwołania — zgodnie z tabelą 2.7 użyty przez nas adres identyfikuje lokalny host, dzięki czemu zaobserwować będziemy mogli operację **bezpośredniego dostarczania** (*direct delivery*). Tego rodzaju lokalne serwery WWW są coraz bardziej popularne jako urządzenia wbudowane w drukarki czy adaptery VoIP.

4.2.1. Dostarczanie bezpośrednie i ARP

Przeanalizujmy więc kolejne kroki scenariusza bezpośredniego dostarczania, koncentrując się na operacjach ARP. Scenariusz ten realizowany jest w sytuacji, gdy datagram IP wysyłany jest do adresu docelowego o tym samym prefiksie, co adres nadawcy — jak zobaczymy w rozdziale 5., scenariusz ten odgrywa istotną rolę w ogólnym procesie forwardowania datagramów IP.

Tak więc wpisaliśmy do przeglądarki URL zawierający adres docelowy `10.0.0.1`; zobaczmy, co kolejno wydarzy się w konsekwencji tej czynności.

1. Aplikacja — przeglądarka WWW — wywołuje specjalną funkcję sprawdzającą, czy podany URL zawiera nazwę mnemoniczną, którą należałoby odwzorować na odpowiadający jej adres IP. W tym przypadku adres IP podany jest jawnie, więc przeliczenie takie nie jest wykonywane.
2. Aplikacja żąda od protokołu TCP nawiązania połączenia z węzłem identyfikowanym 32-bitowym adresem IPv4 `10.0.0.1`.
3. TCP podejmuje próbę nawiązania połączenia, wysyłając datagram z odpowiednim żądaniem do zdalnego hosta o adresie `10.0.0.1` (szczegóły tej operacji omawiać będziemy w rozdziale 15.)
4. Ponieważ zdalny host znajduje się w tej samej sieci, co nasz komputer-nadawca (wynika to wprost z postaci adresu IP — `10.x.x.x`), połączenie może zostać nawiązane bezpośrednio, bez udziału routera.
5. Przy założeniu, że w naszej podsieci używane są adresy sprzętowe kompatybilne z Ethernetem, nasz komputer-nadawca musi wykonać odwzorowanie docelowe adresu IP `10.0.0.1` na 48-bitowy adres MAC docelowego serwera, czyli — zgodnie z terminologią cytowanego dokumentu [RFC0826] — odwzorowanie adresu *logicznego* na odpowiadający mu adres *fizyczny*. To właśnie odwzorowanie zrealizowane zostanie przez protokół ARP.
6. W celu zrealizowania wspomnianego odwzorowania protokół ARP wysyła do warstwy łącza danych ramkę rozgłoszeniową, zawierającą odpowiednie zapytanie (*ARP request*), która to ramka dociera do wszystkich węzłów współdzielących ten sam segment Ethernetu. Notabene ujawnia się w tym momencie istotna cecha protokołu ARP: jest on mianowicie zdolny do funkcjonowania wyłącznie w *sieciach rozgłoszeniowych*, czyli takich, w których warstwa łącza danych realizuje funkcję rozsyłania danej ramki do wszystkich przyłączonych urządzeń. Sieci nieposiadające tej własności, zwane niekiedy *sieciami NBMA* (*Non-Broadcast Multiple Access*), wykorzystywać muszą bardziej skomplikowane protokoły przeliczania adresów (patrz [RFC2332]). Na rysunku 4.1 widzimy prostą *domenę rozgłoszeniową* (*broadcast domain*), reprezentowaną przez zakreskowane pole. W procesie rozgłaszania poszukiwany jest host o adresie IP `10.0.0.1` w celu odczytania jego adresu MAC.
7. Wszystkie hosty w domenie rozgłoszeniowej odbierają zapytanie ARP, dotyczy to także systemów nieimplementujących IPv4 ani IPv6 (lecz już nie systemów w innej sieci VLAN, jeśli implementowany jest mechanizm sieci wirtualnych — patrz rozdział 3.). Jeżeli w domenie rozgłoszeniowej znajduje się host z adresem IP zawartym w zapytaniu ARP, wysyła on ramkę *odpowiedzi ARP* (*ARP Reply*) zawierającą adres IP przesłany w zapytaniu (dla identyfikacji) i właściwą odpowiedź na pytanie — żądany adres MAC. W przeciwieństwie do ramki zapytania, ramka odpowiedzi nie jest ramką rozgłoszeniową, lecz przesyłana jest bezpośrednio do nadawcy zapytania. Jednocześnie host odpowiadający na zapytanie odnotowuje w swych tablicach ARP adres IP nadawcy i jego adres MAC (przesłane w ramce zapytania) do późniejszego użytku (do tej kwestii powrócimy za chwilę).
8. Ramka odpowiedzi ARP zostaje odebrana przez nadawcę zapytania; może on już wysłać przedmiotowy datagram IP, bo zna docelowy adres MAC. Datagram ten dociera wyłącznie do hosta docelowego, bez angażowania innych hostów czy routerów — co stanowi istotę dostarczenia bezpośredniego.



Rysunek 4.1. *Hosty ethernetowe w tej samej domenie rozgłoszeniowej. Zapytanie ARP rozsyłane jest do wszystkich hostów tej domeny za pośrednictwem ramki rozgłoszeniowej; host posiadający adres IP zawarty w zapytaniu odpowiada bezpośrednio na to zapytanie. Hosty nieimplementujące protokołu IP muszą jawnie odrzucać zapytania ARP*

ARP wykorzystywany jest tylko w sieciach IPv4 z wielodostępną warstwą łącza danych, w której każdy host zawiera unikatowy adres sprzętowy. Nie jest więc używany w sieciach PPP (patrz rozdział 3.) — przy zestawianiu łącza PPP (z inicjatywy użytkownika bądź przy bootowaniu systemu) obaj uczestnicy komunikacji wymieniają informację o swych adresach IP, bez wykorzystywania adresów sprzętowych, czyli bez potrzeby odwoływania się do ARP.

4.3. Tablice ARP cache

Podstawowym element warunkującym efektywne działanie implementacji protokołu ARP jest utrzymywana przez tę implementację *pamięć cache*, stanowiąca tablicę skojarzeniową złożoną z par „adres IP-adres MAC". Każda pozycja tej tabeli (z pewnymi wyjątkami) ma ograniczony okres ważności, w przypadku adresów IPv4 równy 20 minut (patrz [RFC1122]).

Szczegóły funkcjonowania implementacji protokołu ARP — a szczególnie zawartość wspomnianej pamięci cache — możemy w systemach Linux i Windows podejrzeć za pomocą polecenia `arp`; parametr wywołania `-a` powoduje wyświetlenie wszystkich pozycji tej pamięci. Raport wyświetlany w systemie Linux prezentuje się podobnie do poniższego:

```
Linux% arp
Address              HWtype  HWaddress           Flags Mask  Iface
gw.home              ether   00:0D:66:4F:60:00   C           eth1
printer.home         ether   00:0A:95:87:38:6A   C           eth1

Linux% arp -a
printer.home (10.0.0.4) at     00:0A:95:87:38:6A [ether] on eth1
gw.home (10.0.0.1) at 00:0D:66:4F:60:00 [ether] on eth1
```

W systemie Windows raport ma nieco inną formę:

```
c:\> arp -a

Interfejs: 10.0.0.56 --- 0x2
  Adres internetowy      Adres fizyczny      Typ
  10.0.0.1               00-0d-66-4f-60-00   dynamiczne
  10.0.0.4               00-0a-95-87-38-6a   dynamiczne
```

W pierwszym raporcie linuksowym każda pozycja zawiera pięć elementów: nazwę hosta (odpowiadającą adresowi IP), typ adresu sprzętowego, adres sprzętowy, znaczniki oraz lokalny interfejs, którego dotyczy mapowanie. Symbole w kolumnie znaczników oznaczają typ pozycji: `C` — dynamiczna, `M` — definiowana statycznie, za pomocą polecenia `arp -s` (patrz podrozdział 4.9) i `P` — publikowana, wykorzystywana do konfigurowania proxy ARP (patrz podrozdział 4.7).

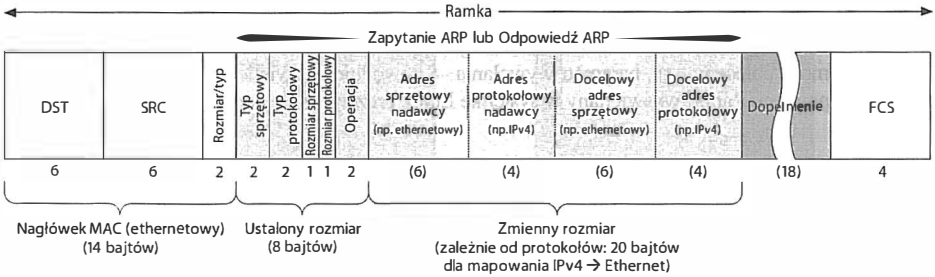
Drugi raport linuksowy zawiera te same informacje, w innej aranżacji (w stylu systemu BSD).

W raporcie windowsowym widzimy natomiast adres IPv4 interfejsu i jego numer (szesnastkowo); dla każdej pozycji w tablicy ARP widzimy ponadto informację, czy pozycja ta została utworzona samodzielnie (dynamicznie) przez protokół ARP, czy też wprowadzona została ręcznie (statycznie) przez użytkownika. Zauważmy, że adres MAC wyświetlany jest w nieco innej postaci niż w Linuksie — dwucyfrowe grupy rozdzielone są myślnikami, nie dwukropkami; dwukropki w roli separatorów są typowe dla systemów uniksowych, podczas gdy inne systemy (i standardy IEEE) przejawiają tendencję do preferowania myślników.

Szerszym omówieniem opcji programu ARP zajmiemy się w podrozdziale 4.9.

4.4. Format ramki ARP

Na rysunku 4.2 przedstawiony jest format ramki ARP, wspólny dla zapytania (*ARP request*) i odpowiedzi (*ARP reply*), dla przeliczania adresów IPv4 na adresy ethernetowe (jak już wspominaliśmy, ARP jest protokołem generycznym, przydatnym dla adresów innych niż IPv4, choć w takiej roli używany bywa bardzo rzadko). Początkowe 14 bajtów to znajomy nagłówek ramki ethernetowej bez znaczników 802.1 p/q (patrz punkt 3.2.2), pozostałe pola są charakterystyczne dla protokołu ARP. Pierwsze 8 bajtów ma ustaloną postać, ciąg dalszy zależny jest od typu mapowanych adresów — na rysunku widoczny jest wariant dla mapowania adresów IPv4 na 48-bitowe adresy ethernetowe.



Rysunek 4.2. *Format ramki ARP dla mapowania adresów IPv4 na 48-bitowe adresy ethernetowe*

W nagłówku ARP ramki z rysunku 4.2 dwa pierwsze pola zawierają adresy ethernetowe docelowy i źródłowy. W ramce żądania ARP adres docelowy jest adresem rozgłoszeniowym, czyli adresem o postaci `ff:ff:ff:ff:ff:ff` (same bity jedynkowe), wskutek czego ramka dostarczona zostanie do wszystkich interfejsów w domenie rozgłoszeniowej. W 2-bajtowym polu *Rozmiar/Typ* znajduje się wartość `0x0806`, identyfikująca ramkę zapytania lub odpowiedzi ARP.

Dwa kolejne pola — *Typ sprzętowy* i *Typ protokołu* — określają typy i rozmiary kolejnych czterech pól, zgodnie z ustaleniami IANA (patrz [RFC5494]). Przymiotniki „sprzętowy" i „protokołowy" używane są do rozróżniania pól związanych z adresami sprzętowymi (tu: ethernetowymi) od pól związanych z adresami protokołu warstwy wyższej (tu: IPv4). Poza kontekstem protokołu ARP przymiotniki te są jednak rzadko używane, częściej adres „sprzętowy" jest *adresem MAC*, *adresem fizycznym* lub *adresem warstwy łącza danych* (w szczególnym przypadku — *adresem ethernetowym*). Typ adresów sprzętowych używanych w ramce określony jest w polu *Typ sprzętowy* — wartość 1 oznacza adres ethernetowy; analogicznie pole *Typ protokołowy* identyfikuje typ adresów podlegających odwzorowaniu na adresy sprzętowe — adresy IPv4 reprezentowane są przez wartość `0x0800` (tę samą, która identyfikuje ramkę ethernetową zawierającą datagram IPv4). Dwa kolejne, 1-bajtowe pola — *Rozmiar sprzętowy* i *Rozmiar protokołowy* — zawierają rozmiary (w bajtach) adresów (odpowiednio) sprzętowych (6 dla Ethernetu) i protokołowych (4 dla IPv4). Pole *Operacja* rozróżnia typ operacji związanej z ramką: zapytanie ARP (wartość 1), odpowiedź ARP (wartość 2), zapytanie RARP (wartość 3), odpowiedź RARP (wartość 4). Rozróżnienie to jest konieczne, bo dla wszystkich wymienionych operacji w polu *Rozmiar/Typ* znajduje się ta sama wartość `0x0806`.

W czterech następnych polach znajdują się kolejno adresy: sprzętowy i protokołowy nadawcy oraz sprzętowy i protokołowy odbiorcy. Zwróćmy uwagę na pewną redundancję — adres sprzętowy nadawcy znajduje się również w polu *SRC* nagłówka ramki. W ramce zapytania pole adresu sprzętowego odbiorcy jest wyzerowane; odbiorca, konstruując ramkę odpowiedzi, wpisuje w to pole swój adres sprzętowy, zamienia rolami adresy nadawcy i odbiorcy, wpisuje wartość 2 w pole *Operacja* i odsyła ramkę.

4.5. Przykłady użycia ARP

W tym podrozdziale zajmiemy się szczegółami tego, co dzieje się w ramach protokołu ARP, gdy użytkownik uruchamia aplikację TCP/IP, np. `telnet` — prostą aplikację ustanawiającą połączenie TCP/IP między dwoma systemami.

4.5.1. Typowy przypadek

Zainicjujmy więc połączenie z lokalnym serwerem WWW o adresie `10.0.0.3`, na porcie 80 (synonim `www`):

```
C:\> arp -a                    Upewniamy się, że tablica ARP jest pusta
No ARP Entries Found
C:\> telnet 10.0.0.3 www       Łączymy się z serwerem WWW na porcie 80
Connecting to 10.0.0.3...
Escape character is '^]'.
```

Naciskając (zgodnie z sugestią) kombinację klawiszy *Ctrl+]*, dostajemy możliwość wprowadzania poleceń; gdy wydamy polecenie `quit`, kończymy pracę programu.

```
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+]'
Microsoft Telnet> quit
```

W czasie naszego krótkiego eksperymentu z programem `telnet` na odnośnym serwerze WWW (tym o adresie `10.0.0.3`) uruchomiony był inny pożyteczny program — `tcpdump` — rejestrujący szczegóły ruchu TCP odbywającego się w kontekście tegoż serwera. Uruchomienie programu z opcją `-e` powoduje rejestrowanie również adresów MAC (w naszym przykładzie są to 48-bitowe adresy ethernetowe). Raport sporządzony przez program widoczny jest poniżej — usunęliśmy jedynie cztery ostatnie jego wiersze związane z kończeniem połączenia (do tej operacji powrócimy rozdziale 13.) jako nieistotne z punktu widzenia operacji ARP (w konkretnym systemie forma raportu może się nieco różnić od poniższej).

```
Linux# tcpdump -e
1     0.0 0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff arp 60:
      arp who-has 10.0.0.3 tell 10.0.0.56
2     0.002174 (0.0022)0:0:c0:c2:9b:26 0:0:c0:6f:2d:40 arp 60:
      arp reply 10.0.0.3 is-at 0:0:c0:c2:9b:26

3     0.002831 (0.0007)0:0:c0:6f:2d:40 0:0:c0:c2:9b:26 ip 60:
      10.0.0.56.1030 > 10.0.0.3.www: S 596459521:596459521(0)
      win 4096 <mss 1024> [tos 0x10]
4     0.007834 (0.0050)0:0:c0:c2:9b:26 0:0:c0:6f:2d:40 ip 60:
      10.0.0.3.www > 10.0.0.56.1030: S 3562228225:3562228225(0)
      ack 596459522 win 4096 <mss 1024>
5     0.009615 (0.0018)0:0:c0:6f:2d:40 0:0:c0:c2:9b:26 ip 60:
      10.0.0.56.1030 > 10.0.0.3.discard: . ack 1 win 4096 [tos 0x10]
```

I tak w rekordzie 1 widzimy adres sprzętowy nadawcy `0:0:c0:6f:2d:40` i adres docelowy `ff:ff:ff:ff:ff:ff`, będący w Ethernecie adresem rozgłoszeniowym. Ramki z takim adresem docelowym docierają do wszystkich węzłów sieci LAN lub VLAN, nawet tych nieimplementujących protokołu IPv4, zgodnie z rysunkiem 4.1. Na podstawie pola

Rozmiar/Typ (o zawartości 0x0806) ramka zostaje rozpoznana jako arp, czyli niosąca zapytanie lub odpowiedź ARP. Wartość 60, następująca po słowach arp oraz ip w każdym z pięciu pakietów, oznacza długość ramki ethernetowej; rzeczywista długość ramki zapytania lub odpowiedzi ARP wynosi 42 bajty (14 bajtów nagłówka plus 28 bajtów danych ARP), konieczne jest więc jej dopełnienie do (minimalnej dla Ethernetu) długości 60 bajtów. Ostatnim polem ramki jest suma kontrolna FCS (sposób jej obliczania wyjaśnialiśmy w rozdziale 3.).

Kolejny wiersz raportu dotyczącego rekordu 1 (who-has ...) identyfikuje ramkę jako zapytanie ARP dotyczące docelowego adresu IP 10.0.0.3, wysłane przez nadawcę o adresie 10.0.0.56. Program tcpdump wyświetla także nazwy mnemoniczne odpowiadające adresom IP, o ile nazwy te są dostępne; w naszym przykładzie są one niedostępne, ponieważ nie zdefiniowano odwrotnej translacji DNS — szczegóły usługi DNS wyjaśniamy w rozdziale 11. (Wywołując program tcpdump z opcją -n, wyłączamy wyświetlanie nazw mnemonicznych niezależnie od ich dostępności).

Ramka (raportowana jako rekord 1) wysłana przez nadawcę 10.0.0.56 dociera ostatecznie do hosta docelowego 10.0.0.3, który w odpowiedzi konstruuje ramkę zawierającą jego adres MAC 0:0:c0:6f:2d:40. Ramka odpowiedzi *nie jest zasadniczo ramką rozgłoszeniową*, lecz skierowana jest bezpośrednio do nadawcy zapytania (który teraz staje się adresatem odpowiedzi) — w podrozdziale 4.8 opiszemy jednak wyjątki od tej reguły. Program tcpdump raportuje adres IP hosta udzielającego odpowiedzi oraz jego adres MAC będący sednem tej odpowiedzi.

Rekord 3 raportu odzwierciedla żądanie nawiązania połączenia, pochodzące od segmentu TCP (szczegółami segmentów TCP zajmiemy się w rozdziale 13.). Żądanie skierowane jest do adresu MAC otrzymanego jako odpowiedź w ramach rekordu 2.

Dla każdego raportowanego rekordu program tcpdump wyświetla czas jego wysłania (w sekundach), liczony jako przesunięcie względem czasu wysłania rekordu 1. Dla każdego rekordu (z wyjątkiem rekordu 1) wyświetlany jest także (w nawiasach) czas, jaki upłynął od wysłania rekordu *poprzedniego*; widzimy więc, że czas między wysłaniem zapytania ARP a otrzymaniem odpowiedzi równy jest ok. 2,2 ms, a po upływie dalszych 0,7 ms wysyłany jest pierwszy segment TCP; narzut związany z przeliczaniem adresu IP na odpowiadający mu adres MAC wynosi więc niespełna 3 ms.

Przy okazji zwróćmy uwagę na pewną subtelność związaną z udzielaniem odpowiedzi na zapytanie ARP. Generalnie jeśli host **B** (ten o adresie 10.0.0.3) otrzymuje od hosta **A** (tego o adresie 10.0.0.56) żądanie udostępnienia swego adresu MAC, oprócz sformułowania odpowiedzi na to żądanie, korzystając niejako z okazji, odnotowuje w swych tablicach ARP adres MAC hosta **A**. Owo przysłowiowe upieczenie dwóch pieczeniu na jednym ogniu jest rezultatem dalekowzrocznej polityki: skoro host **A** żąda od hosta **B** adresu sprzętowego, to niechybnie za chwilę host **B** otrzyma od hosta **A** datagram IP, *na który najprawdopodobniej trzeba będzie odpowiedzieć*; host **B** będzie wówczas potrzebował adresu sprzętowego hosta **A** — zapamiętując ów adres zawczasu, uwalnia się od konieczności wywoływania w tym celu protokołu ARP. Tłumaczy to nieobecność zapytania ARP, jakie spodziewałby się czytelnik ujrzeć przed rekordem 4, raportującym wysyłanie segmentu TCP z hosta 10.0.0.3 do hosta 10.0.0.56.

4.5.2. Zapytanie ARP o nieistniejący host

Oczywiście, nie sposób nie zapytać, jaki będzie rezultat zapytania ARP skierowanego do *nieistniejącego adresu IP*? Najprościej sprawdzić to eksperymentalnie, wybierając adres nieistniejący w lokalnej podsieci (identyfikowanej prefiksem 10/8):

```
Linux% date ; telnet 10.0.0.99 ; date
Fri Jan 29 14:46:33 CET 2010
Trying 10.0.0.99...
telnet: connect to address 10.0.0.99: No route to host

Fri Jan 29 14:46:36 CET 2010          3 sekundy później
Linux% arp -a
? (10.0.0.99) at <incomplete> on eth0
```

A co działo się w tym czasie na serwerze?

```
Linux# tcpdump -n arp
1 21:12:07.440845 arp who-has 10.0.0.99 tell 10.0.0.56
2 21:12:08.436842 arp who-has 10.0.0.99 tell 10.0.0.56
3 21:12:09.436836 arp who-has 10.0.0.99 tell 10.0.0.56
```

Tym razem nie używamy parametru -e, bo wiemy już, że w ramach zapytania ARP adres docelowy (*DST*) jest adresem rozgłoszeniowym. Z raportu sporządzonego przez tcpdump wynika, że protokół ARP ponawia nieudaną próbę przeliczenia adresu IP co ok. sekundę, co jest wartością maksymalną w świetle sugestii dokumentu [RFC1122]. Testowanie ARP w systemie Windows (niepokazane tutaj) ujawnia jednak inne zachowanie: zamiast trzech prób w sztywnym odstępnie jednej sekundy, liczba prób i odstępy czasowe między nimi uzależnione są od konkretnej aplikacji i od innych wykorzystywanych protokołów. Gdy przykładowo protokołami tymi są ICMP (patrz rozdział 8.) lub UDP (rozdział 10.), odstęp czasowy między kolejnymi próbami wynosi ok. 5 sekund, podczas gdy w przypadku protokołu TCP jest on równy ok. 10 sekund. Takie postawienie sprawy pozwala na dwa zapytania ARP pozostawione bez odpowiedzi, zanim protokół TCP da za wygraną i zrezygnuje z nieudanej próby nawiązania połączenia.

4.6. Przeterminowanie danych ARP

Z każdą pozycją tablicy ARP związany jest czas ważności (*timeout*), po upływie którego pozycja ta jest z tablicy usuwana (jak jednak później zobaczymy, polecenie arp pozwala administratorowi na definiowanie w tej tablicy pozycji niepodlegających usuwaniu na tej zasadzie). W większości implementacji protokołu ARP czas ten wynosi 20 minut w przypadku pozycji kompletnej i 3 minuty dla pozycji niekompletnej (czyli takiej, dla której proces przeliczania nie został zakończony — co widzieliśmy choćby w przykładzie z punktu 4.5.2). Czas ważności pozycji liczony jest najpierw od jej pojawienia się w tablicy, a następnie od momentu *odwołania się do niej*; przeterminowaniu podlegają więc te opcje, do których nie nastąpiło odwołanie w ciągu interwału czasowego równego okresowi ich ważności. Co prawda, zgodnie z zaleceniem dokumentu [RFC1122] formułującego wymagania stawiane hostom internetowym, fakt odwoływania się do pozycji nie powinien mieć wpływu na jej przeterminowanie — decydujący powinien być jedynie czas przebywania pozycji w tablicy — lecz wiele implementacji ARP zalecenie to zwyczajnie ignoruje.

Notabene pozycja tablicy ARP, dla której okres ważności zaczyna biec od nowa po każdym odwołaniu, jest przykładem tzw. **miękkiego stanu** (*soft state*) — czyli informacji wymagającej okresowego odświeżania, pod rygorem usunięcia, gdy odświeżenie to nie nastąpi w założonym interwale czasowym. Miękki stan wykorzystywany jest przez wiele protokołów internetowych jako ułatwiający automatyczną rekonfigurację w obliczu zmieniających się warunków sieciowych. Jego automatyczne odświeżanie — jako ochrona przed nieuzasadnionym usunięciem — jest jedną z głównych cech projektowych wielu protokołów.

4.7. Proxy ARP

Proxy ARP (opisywany w [RFC1027]) to system (najczęściej odpowiednio skonfigurowany router) odpowiadający na zapytania ARP w imieniu innego hosta. Z punktu widzenia nadawcy zapytania wszystko wygląda tak, jakby na zapytanie to odpowiedział host o podanym adresie; naprawdę jednak taki host może nawet nie istnieć, za wszystkim stoi host proxy. Mechanizm ten nie jest powszechnie wykorzystywany, zaleca się jego unikanie.

Proxy ARP określany bywa także swojskim mianem *promiscuous ARP*[1] lub mianem „hackingu ARP" (*ARP Hack*), a to z powodów niegdysiejszego zastosowania, jakim było wzajemne ukrywanie dwóch (fizycznych) sieci nawzajem przed sobą. Obie sieci współdzielą ten sam prefiks IP, a dzielący je router skonfigurowany jest jako agent proxy ARP; zapytania ARP przychodzące z jednej sieci kwitowane są adresami sprzętowymi hostów z drugiej. W ten sposób wybrana grupa hostów jest efektywnie ukrywana przed inną grupą. W przeszłości mechanizm ten był substytutem podziału sieci na podsieci (w sytuacji gdy podział taki był niemożliwy do przeprowadzenia), był ponadto niezbędny w obliczu używania starszych adresów rozgłoszeniowych, złożonych z samych zer (a nie, jak obecnie, z samych bitów jedynkowych).

Mechanizm proxy ARP wspierany jest przez system Linux pod postacią zwaną **auto-proxy ARP**. Jego włączenie następuje w wyniku wpisania znaku 1 do pliku */proc/sys/net/ipv4/conf/*/proxy_arp* (gwiazdka symbolizuje tu konkretne urządzenie) lub przy użyciu polecenia `sysctl`. Polecenie to umożliwia podanie zakresu adresów IP, które będą objęte wspomnianym mechanizmem, co jest wygodną alternatywą dla ręcznego wpisywania każdego z tych adresów.

4.8. Gratuitous ARP i wykrywanie konfliktu adresów IP

Kolejna użyteczna opcja protokołu ARP nosi nazwę *Gratuitous ARP* [2](w skrócie GARP) i polega na tym, że host wysyła zapytanie ARP dotyczące *swego własnego* adresu IP.

[1] Pomysłodawca nazwy przyrównał po prostu host proxy ARP do stręczyciela kojarzącego pary hostów-partnerów wywodzących się z dwóch różnych sieci. I choć może metafora to w nie najlepszym guście, trafności spostrzeżeniu odmówić nie sposób — *przyp. tłum.*

[2] Nie znalazłem wiarygodnego polskiego odpowiednika; ze względu na naturę tej opcji można by nazwać ją „retorycznym ARP" (co pozostawiam jako propozycję do oceny przez czytelników) — *przyp. tłum.*

Tak dzieje się m.in. w przypadku doraźnego konfigurowania interfejsu w czasie bootowania systemu. Powróćmy do poprzednich przykładów — gdy uruchamialiśmy naszą stację windowsową (tę o adresie 10.0.0.56), w dzienniku serwera linuksowego pojawił się taki oto rekord:

```
Linux# tcpdump -e -n arp
1       0.0 0:0:c0:6f:2d:40 ff:ff:ff:ff:ff:ff arp 60:
            arp who-has 10.0.0.56 tell 10.0.0.56
```

(użyliśmy opcji -n, ponieważ interesują nas jedynie adresy IP hostów, nie ich nazwy mnemoniczne). W kategoriach formatu ramki zapytania ARP pola *Adres protokołowy nadawcy* i *Docelowy adres protokołowy* mają tę samą wartość 10.0.0.56, ponadto adres w polu *SRC* nagłówka ethernetowego jest taki sam jak *Adres sprzętowy nadawcy* (0:0:c0:6f:2d:40). Opisywany mechanizm ma do spełnienia dwojaki cel:

1. Pozwala na wykrywanie istnienia dwóch (lub więcej) hostów o tym samym adresie IP. Host wysyłający zapytanie *Gratuitous ARP* nie powinien otrzymać na nie odpowiedzi; jeśli odpowiedź taka jednak nadejdzie, pochodzić będzie od „bliźniaczego" hosta — w większości implementacji sytuacja taka kwitowana jest wyświetleniem komunikatu w rodzaju „Zdublowany adres IP sygnalizowany przez host o adresie sprzętowym ...", co stanowi poważne ostrzeżenie dla użytkownika (lub administratora) sieci LAN lub VLAN o błędnym skonfigurowaniu domeny rozgłoszeniowej.
2. Gdy host zmieni swój adres sprzętowy (np. w wyniku wymiany karty sieciowej), fakt ten powinien zostać odzwierciedlony we wszystkich tablicach ARP zawierających adnotację o „starym" adresie sprzętowym tegoż hosta. Jak wcześniej pisaliśmy, host odbierający zapytanie ARP przed odesłaniem odpowiedzi zapisuje (lub uaktualnia) w swych tablicach ARP adres IP i adres MAC nadawcy. Gratuitous ARP skwapliwie wykorzystuje ten mechanizm.

Mimo iż mechanizm gratuitous ARP umożliwia wykrycie zdublowania adresów IP, nie dostarcza żadnych środków do radzenia sobie z tą nieprzyjemną sytuacją (poza wspomnianym wyświetleniem komunikatu, zauważonym lub nie przez administratora). W celu załatania tej luki opracowano mechanizm **wykrywania konfliktu adresów** IPv4 (ACD — *Address Conflict Detection*) opisany w dokumencie [RFC5227]. Mechanizm ten definiuje dwa pakiety: **próbkowania ARP** (*ARP probe*) i **ogłaszania ARP** (*ARP announcement*). Pakiet próbkowania jest pakietem zapytania ARP, w którym *Adres protokołowy nadawcy* ma wartość zerową; celem próbkowania jest wykrycie ewentualnego istnienia *Docelowego adresu protokołowego*, a zerowa wartość *Adresu protokołowego nadawcy* stanowi zabezpieczenie przed zaśmieceniem tablicy ARP (ewentualnie) wykrytego hosta — nie ma tej ochrony w przypadku gratuitous ARP. W pakiecie ogłaszania oba pola adresu logicznego — *Adres protokołowy nadawcy i Docelowy adres protokołowy* — mają wartość równą „kandydackiemu" adresowi IPv4; w ten sposób host-nadawca oznajmia pozostałym hostom domeny rozgłoszeniowej, że zamierza używać konkretnego adresu IPv4 jako własnego.

Host rozpoczyna wysyłanie ramek próbkowania po włączeniu interfejsu, wybudzeniu interfejsu ze stanu oszczędzania energii, ustanowieniu nowego łącza bezprzewodowego itp. Przed wysłaniem pierwszej ramki odczekuje jednak przez losowy okres nie dłuższy niż 1 sekunda, co stanowić ma zabezpieczenie przed zmasowanym atakiem na ramkę próbkowania ze strony wielu np. równocześnie wybudzonych interfejsów; w przypadku ne-

gatywnego wyniku próbkowania (czyli niewykrycia adresu kandydackiego) jest ono jeszcze powtarzane dwukrotnie, za każdym razem z losowo wybranym opóźnieniem nie krótszym niż sekunda i nie dłuższym niż dwie sekundy. Negatywny wynik wszystkich trzech próbkowań oznacza nieistnienie w domenie adresu kandydackiego IPv4, czyli brak potencjalnego konfliktu adresów.

W czasie próbkowania stacja otrzymywać może zarówno odpowiedzi ARP (świadczące o istnieniu adresu kandydackiego), jak i pakiety próbkowania pochodzące od innych stacji (a to oznacza, że dwie stacje wypróbowują właśnie ten sam kandydacki adres IPv4). System powinien potraktować oba te przypadki jako konflikt adresów i zaproponować adres alternatywny. Scenariusz taki zalecany jest w przypadku automatycznego przydzielania adresów IP przez protokół DHCP (piszemy o tym w rozdziale 6.). W dokumencie [RFC5227] zdefiniowano górną granicę *dziesięciu* prób uzyskania adresu w opisany sposób; po wyczerpaniu tego limitu host kontynuować ma do skutku próbę uzyskania adresu, jednak ze znacznie większym odstępem między kolejnymi próbkowaniami — 60 sekund.

Jeśli opisana procedura nie doprowadzi do wykrycia konfliktu adresów, host przyjmuje testowany adres kandydacki jako swój i oznajmia ten fakt innym hostom w domenie rozgłoszeniowej, wysyłając dwa komunikaty *Ogłoszenie ARP* w odstępie 2 sekund; jak wcześniej pisaliśmy, w komunikatach tych oznajmiany adres znajduje się w polach *Adres protokołowy nadawcy i Docelowy adres protokołowy*, a celem ich wysłania jest zaktualizowanie ewentualnych adnotacji dotyczących tego adresu w tablicach ARP innych hostów.

W przeciwieństwie do gratuitous ARP stanowiącego pojedynczy akt wysłania komunikatu, ACD traktowane jest jak ciągły proces, niekończący się bynajmniej wysłaniem wspomnianego oznajmienia. Host, który przyjął określony adres IP, kontynuuje inspekcję nadchodzących komunikatów ARP (zapytań i odpowiedzi) w celu sprawdzania, czy jego adres nie pojawia się w polu *Adres protokołowy nadawcy* — taka sytuacja świadczyłaby o tym, że inny host używa, w dobrej wierze, tego samego adresu IP. W dokumencie [RFC5227] opisane są trzy możliwe sposoby rozwiązania wykrytego w ten sposób konfliktu:

- porzucenie nabytego adresu,
- zatrzymanie nabytego adresu z jednoczesnym wysłaniem kilku „defensywnych" ogłoszeń ARP i porzucenie adresu, jeśli konflikt będzie się utrzymywał,
- zatrzymanie nabytego adresu mimo istniejącego konfliktu.

Ostatnia z opcji zalecana jest wyłącznie w odniesieniu do systemów rzeczywiście wymagających ustalonego adresu, przede wszystkim wbudowanych urządzeń w rodzaju drukarek lub routerów.

Autorzy dokumentu [RFC5227] zwracają także uwagę na potencjalne korzyści wynikające z rozsyłania niektórych odpowiedzi ARP za pośrednictwem mechanizmu rozgłaszania warstwy łącza danych, w wyniku czego każda stacja należąca do danego segmentu przetwarzałaby cały ruch ARP generowany w tym segmencie. Mimo iż nie jest to standardowy tryb funkcjonowania ARP, jest korzystny z perspektywy ACD, ponieważ poszczególne stacje mogą szybciej unieważniać wpisy w swych tablicach ARP w przypadku stwierdzenia konfliktu adresów IP.

4.9. Polecenie arp

W prezentowanych przykładach pokazaliśmy już zastosowanie opcji `-a` w wywołaniu programu `arp` — jej włączenie powoduje wyświetlenie wszystkich pozycji tablicy ARP. W Linuksie opcja ta jest domyślnie włączona, więc wyświetlenie wszystkich pozycji uzyskamy przez wywołanie `arp` bez parametrów.

Administrator i superużytkownik mogą selektywnie *usuwać* pozycje tablicy ARP, używają parametru `-d` (wykorzystywaliśmy tę możliwość w prezentowanych wcześniej przykładach) w celu wymuszenia wymiany komunikatów ARP.

Dodawanie pozycji do tablic ARP jest możliwe przy użyciu parametru `-s`. Należy jawnie określić oba adresy; IPv4 i MAC (zamiast adresu IPv4 można podać odpowiadającą mu nazwę mnemoniczną DNS). Dodana w ten sposób opcja ma charakter półtrwały: nie podlega przeterminowaniu (ma nieograniczony termin ważności), ale — uwaga! — znika przy ponownym bootowaniu systemu.

Linuksowa wersja programu `arp` oferuje dodatkowe opcje (w porównaniu z windowsowym odpowiednikiem). Użycie słowa kluczowego `temp` w połączeniu z opcją `-s` powoduje, że dodawana pozycja nie uzyskuje charakteru półtrwałego, lecz podlega przeterminowaniu na równi z pozycjami dynamicznymi. Z kolei użycie słowa kluczowego `pub` w połączeniu z opcją `-s` oznacza dodanie pozycji, która służyć będzie do *odpowiadania* na zapytania ARP dotyczące danego adresu IP, czyli raportowania podanego adresu MAC jako związanego z owym adresem IP. W taki właśnie sposób realizować można funkcję agenta proxy ARP, o której pisaliśmy w podrozdziale 4.7 — funkcję działającą nawet wtedy, gdy w pliku */proc/sys/net/ipv4/conf/*/proxy_arp* znajduje się znak `0`.

4.10. Przypisywanie adresów IPv4 za pomocą ARP

W miarę coraz większej popularności wbudowanych (*embedded*) urządzeń zgodnych z Ethernetem i TCP/IP istotniejsza staje się kwestia odnajdywania tych urządzeń w sieciach — zwłaszcza tych urządzeń, których nie da się konfigurować bezpośrednio, np. z powodu braku klawiatury, przy użyciu której można by wprowadzić adres IP do pamięci urządzenia.

W takiej sytuacji konfigurowanie wspomnianych urządzeń przeprowadzać można na dwa sposoby. Pierwszym z nich jest użycie protokołu DHCP do automatycznego przyporządkowania urządzeniu adresu IP i innych pokrewnych informacji — szczegółami tej operacji zajmiemy się w rozdziale 6. Drugi, rzadziej wykorzystywany sposób, wiąże się z użyciem protokołu ARP, za pomocą którego (wydając polecenie `arp -s`) wiąże się adres MAC urządzenia z wybranym adresem IP, po czym wysyła się datagram IP pod ten adres. Ponieważ przyporządkowanie „IP → MAC" figuruje już w tablicy ARP, nie jest generowane zapytanie ARP. Oczywiście, wykonanie tej operacji wymaga znajomości adresu MAC urządzenia — ten zwykle uwidaczniany jest na jego obudowie i często powtarzany w numerze seryjnym nadawanym urządzeniom przez producenta. Gdy urządzenie otrzyma przeznaczony dla niego pakiet na podstawie adresu MAC umieszczonego w ramce ethernetowej, zawarty w pakiecie IP adres IPv4 ma drugorzędne znaczenie.

Wiele urządzeń posiada także inne możliwości konfiguracyjne, np. przy użyciu wbudowanego serwera WWW.

4.11. Ataki sieciowe z użyciem ARP

Najprostszy bodaj z możliwych ataków angażujących protokół ARP polega na użyciu mechanizmu proxy ARP w charakterze maskarady. Na zapytanie ARP dotyczące określonego adresu IP można wówczas udzielić dowolnej odpowiedzi, nawet jeśli nie istnieje host o tym adresie — gdyby jednak istniał, to również odpowiedziałby na zapytanie i podstęp zostałby łatwo zdemaskowany.

Bardziej subtelny atak możliwy jest w odniesieniu do stacji przyłączonych do dwóch (lub więcej) sieci jednocześnie, za pośrednictwem oddzielnych interfejsów. Błędy w implementacji ARP mogą powodować „przecieki" między tablicami ARP różnych interfejsów. Błędne pozycje w tych tablicach mogą powodować dostarczanie pakietów do sieci innej niż ta, do której zostały oryginalnie skierowane. W Linuksie można się zabezpieczać przed atakami tego typu, wpisując znak `1` do pliku */proc/sys/net/ipv4/conf/*/arp_filter*. *Adres protokołowy nadawcy* zawarty w ramce nadchodzącego zapytania ARP jest wówczas weryfikowany na okoliczność tego, który interfejs służyć ma do odsyłania datagramów na ów adres — jeśli jest to interfejs różny od tego, na który przyszło zapytanie ARP, zapytanie to zostaje zignorowane (nie jest udzielana odpowiedź ARP).

Znacznie bardziej dotkliwe w skutkach mogą być ataki ukierunkowane na statyczne pozycje w tablicach skojarzeniowych. „Statyczność" tych pozycji oznacza (w założeniu), z jednej strony, konfigurowanie ich explicite, nie na zasadzie dialogu „żądanie-odpowiedź", z drugiej natomiast, czyni je nienaruszalnymi, czyli niewrażliwymi na próby aktualizacji. W ten sposób szczególnie ważne hosty chronić można przed nieuprawnionym dostępem. Niestety, teoria teorią, a w praktyce większość implementacji ARP nie zapewnia wspomnianej odporności, co czyni pozycje statyczne podatne na aktualizację w wyniku odpowiedzi ARP — i to nawet w sytuacji, gdy odpowiedź ta nie jest reakcją na wcześniej wysłane żądanie. Szczególnie ważne pozycje w tablicy skojarzeniowej mogą być więc podmienione przez hakera równie łatwo, jak pozostałe pozycje dynamiczne.

4.12. Podsumowanie

ARP jest jednym z podstawowych protokołów w każdej niemal implementacji zestawu protokołów TCP/IP, choć jego funkcjonowanie jest raczej mało spektakularne z perspektywy aplikacji lub jej użytkownika. Jego zadaniem jest zidentyfikowanie adresu sprzętowego urządzenia opatrzonego określonym adresem IPv4, znajdującego się w tej samej podsieci; jest również wykorzystywany do zidentyfikowania odpowiedniego routera w sytuacji, gdy wspomniane urządzenie znajduje się w innej (pod)sieci — szczegóły tego mechanizmu omawiamy w rozdziale 5. Podstawą działania protokołu ARP są jego tablice, w których cache zapamiętywane są wyniki zapytań. Zarówno kompletne, jak i niekompletne pozycje tych tablic mają określony czas ważności, po upływie którego są ze swych tablic usuwane. Do przeglądania zawartości tych tablic i manipulowania ich zawartością służy polecenie `arp`.

W rozdziale omówiliśmy zarówno standardowe funkcjonowanie protokołu ARP, jak i jego przypadki szczególne: proxy ARP (zwracający adres sprzętowy hosta należącego do podsieci innej niż ta, z której nadchodzi zapytanie) oraz Gratuitous ARP (czyli zapytanie hosta o skojarzony z jego własnym adresem IP adres sprzętowy, zwykle podczas bootowania systemu). Omówiliśmy także podstawy mechanizmu ACD, wykrywającego konflikty adresów IP (czyli przypadki opatrzenia tym samym adresem IP dwóch lub więcej urządzeń w tej samej domenie rozgłoszeniowej) i usiłującego konfliktom tym przeciwdziałać. Na zakończenie rozważyliśmy kilka możliwości przypuszczania ataków na sieci z wykorzystaniem protokołu ARP — większość z nich sprowadza się do podszywania się hostów pod „cudze" adresy IP, co prowadzić może do poważnych problemów w sytuacji, gdy protokoły warstw wyższych nie implementują silnych mechanizmów zabezpieczeń (którymi zajmiemy się w rozdziale 18.).

4.13. Bibliografia

[RFC0826] D. Plummer, *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*, Internet RFC 0826/STD 0037, listopad 1982.

[RFC0903] R. Finlayson, T. Mann, J.C. Mogul, M. Theimer, *A Reverse Address Resolution Protocol*, Internet RFC 0903/STD 0038, czerwiec 1984.

[RFC1027] S. Carl-Mitchell, J.S. Quarterman, *Using ARP to Implement Transparent Subnet Gateways*, Internet RFC 1027, październik 1987.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts*, Internet RFC 1122/STD 0003, październik 1989.

[RFC2332] J. Luciani, D. Katz, D. Piscitello, B. Cole, N. Doraswamy, *NBMA Next Hop Resolution Protocol (NHRP)*, Internet RFC 2332, kwiecień 1998.

[RFC5227] S. Cheshire, *IPv4 Address Conflict Detection*, Internet RFC 5227, lipiec 2008.

[RFC5494] J. Arkko, C. Pignataro, *IANA Allocation Guidelines for the Address Resolution Protocol (ARP)*, Internet RFC 5494, kwiecień 2009.

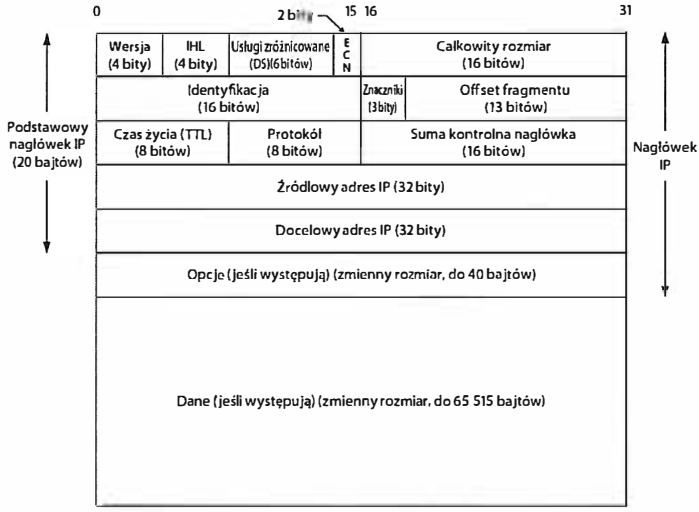
Rozdział 5.
Protokół internetowy (IP)

5.1. Wprowadzenie

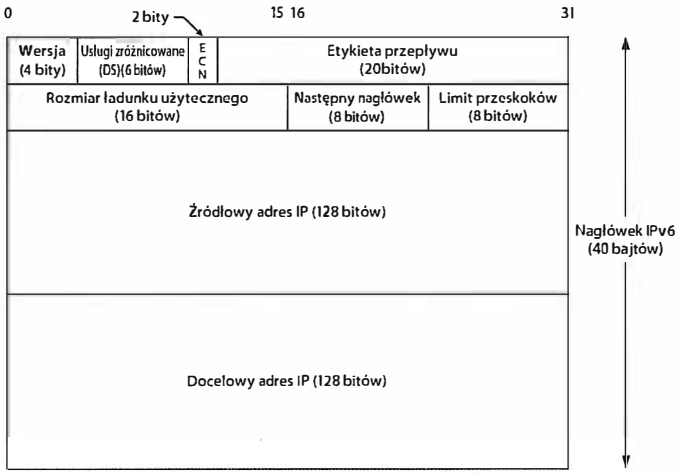
Protokół internetowy (*Internet Protocol*), znany powszechnie pod akronimem IP, jest motorem napędowym całego zestawu protokołów TCP/IP. Dane protokołów TCP, UDP, ICMP i IGMP transmitowane są właśnie w postaci datagramów IP. Protokół IP zapewnia usługę niegwarantowanego (*best-effort*), bezpołączeniowego (*connectionless*) transferu datagramów — przymiotnik „niegwarantowany" oznacza tu, że mimo „najlepszych starań" (bo tak tłumaczy się *best-effort*) wysłanie pakietu nie daje gwarancji, że pakiet ten dotrze do zamierzonego celu. Nie oznacza to, rzecz jasna, że gubienie pakietów jest naturalną funkcją protokołu IP: po prostu, po wysłaniu pakietu protokół (a właściwie jego implementacja w routerze) nie troszczy się już o jego los, a niemożność przetworzenia odebranego pakietu (np. z powodu chwilowego przepełnienia buforów routera) albo też stwierdzenie jego przekłamania „po drodze" kwitowane są banalną reakcją routera — odrzuceniem pakietu. Zasada ta wspólna jest dla obu wersji protokołu — IPv4 i IPv6. Wynika z niej natychmiast zasada pokrewna: to na (wymienionych wcześniej) protokołach warstwy wyższej spoczywa obowiązek zweryfikowania, czy zadanie zlecone protokołowi IP zostało przezeń poprawnie wykonane.

Przymiotnik „bezpołączeniowy" oznacza, że przesyłanie datagramów między dwoma węzłami sieci (routerami) nie wiąże się z uprzednim negocjowaniem jakiegoś porozumienia między tymi węzłami. Każdy docierający do routera datagram traktowany jest niezależnie od innych datagramów, a jego przetwarzanie opiera się wyłącznie na informacji zawartej wewnątrz niego, bez kontekstu jakiejkolwiek innej informacji o stanie transferu. W szczególności oznacza to, że wysyłane datagramy docierać mogą do miejsca przeznaczenia w kolejności innej niż kolejność ich wysyłania, bo podążać mogą do tegoż miejsca różnymi drogami. Możliwe jest także *powielanie* (duplikowanie) datagramów oraz *zniekształcanie* ich treści („przekłamywanie") na skutek różnych czynników fizycznych oddziałujących na nośniki transmisyjne. Z tym wszystkim liczyć się muszą protokoły zlecające protokołowi IP transfer swoich danych.

Szczegóły transferu danych enkapsulowanych w pakietach IP określone są przez *nagłówki* (*headers*) tych pakietów, różniące się istotnie w obu wersjach. Na rysunku 5.1 przedstawiono strukturę pakietu w wersji IPv4, natomiast nagłówek pakietu IPv6 widoczny jest na rysunku 5.2; oficjalna specyfikacja protokołu IPv4 zawarta jest w dokumencie [RFC0791], protokołowi IPv6 poświęcona jest natomiast cała seria dokumentów, zapoczątkowana przez [RFC2460].



Rysunek 5.1. *Datagram IPv4. Nagłówek ma zmienny rozmiar, do 60 bajtów (15 słów 32-bitowych, na tyle pozwala 4-bitowe pole IHL); w (typowym) przypadku braku opcji rozmiar ten wynosi 20 bajtów. Adresy IP źródłowy i docelowy zapisane są w standardowej postaci w słowach 32-bitowych. Pola Identyfikacja i Offset fragmentu związane są z funkcją fragmentowania pakietów IPv4. Suma kontrolna, chroniąca integralność pól nagłówka, obliczana jest wyłącznie dla obszaru nagłówka, bez związku z polem danych (których integralność nie jest w związku z tym chroniona)*



Rysunek 5.2. *Nagłówek pakietu IPv6 ma ustalony rozmiar 40 bajtów. Pole Następny nagłówek służy do wiązania w ciąg ewentualnych nagłówków rozszerzeń, które występować mogą w pakiecie po nagłówku podstawowym, zwykle bezpośrednio przed danymi protokołu warstwy wyższej. Oba adresy IP — źródłowy i docelowy — zapisane są w binarnej postaci 128-bitowej*

5.2. Nagłówki IPv4 i IPv6

Widoczny na rysunku 5.1 nagłówek datagramu IPv4 ma zasadniczo rozmiar 20 bajtów, lecz rozmiar ten może być większy, jeśli wspomniany nagłówek zawiera dodatkowe *opcje* (co zdarza się raczej rzadko). Przedstawiony na rysunku 5.2 nagłówek datagramu IPv6 ma *zawsze* rozmiar 40 bajtów; opcjonalne funkcje związane z datagramem reprezentowane są w postaci łańcucha nagłówków rozszerzeń, którymi zajmiemy się szczegółowo w dalszym ciągu tego rozdziału.

We wszystkich prezentowanych diagramach bity numerowane są w kolejności *malejącego* znaczenia: najbardziej znaczący bit ma numer 0. W 32-bitowym słowie najmniej znaczący bit ma numer 31, zaś poszczególne bajty słowa uszeregowane są w ten sposób, że najbardziej znaczący bajt znajduje się pod najmniejszym adresem, tak więc np. ciąg bajtów o wartości 0xDE, 0xAF, 0xC0, 0xDA (w kolejności wzrastających adresów) tworzy słowo o wartości 0xDEAFC0DA. Taka konwencja traktowania słów wielobajtowych nosi nazwę *big-endian*[1] i wymagana jest dla wszystkich binarnych liczb całkowitych występujących w transmitowanych przez sieć nagłówkach protokołów TCP/IP (z tego względu konwencję tę nazywa się *sieciową kolejnością bajtów — network byte order*). Uporządkowanie odwrotne, w którym najbardziej znaczący bajt znajduje się pod najwyższym adresem, nosi nazwę *little-endian* i notabene stosowane jest przez znacznie więcej procesorów (m.in. przez procesory serii x86 Intela); cytowany ciąg bajtów 0xDE, 0xAF, 0xC0, 0xDA tworzy więc w konwencji *little-endian* słowo 0xDAC0AFDE. W komputerach wykorzystujących procesory z uporządkowaniem *little-endian* konieczna jest więc konwersja uporządkowania bajtów nagłówka pakietu przed jego wysłaniem w sieć oraz po jego odebraniu z sieci.

5.2.1. Pola nagłówków IP

Pierwszym polem nagłówka datagramu obu wersji jest czterobitowe pole *Wersja*, zawierające wartość 4 (binarnie 0100) dla datagramu IPv4 i wartość 6 (binarnie 0110) dla datagramu IPv6. Pierwsze cztery bity datagramu umożliwiają więc rozpoznanie jego wersji — rozpoznanie bardzo ważne, bowiem na polu *Wersja* kończy się zgodność obu formatów i każdy z nich wymaga innego przetwarzania przez host lub router. Mimo iż opracowano i zaproponowano (w charakterze kandydatury do standardu) kilka jeszcze innych wersji protokołu IP (ich wykaz, wraz z odsyłaczami, dostępny jest na stronie [IV]), w użyciu są niemal wyłącznie wersje o numerach 4 i 6.

Pole *IHL* (od *Internet Header Length* — rozmiar nagłówka internetowego) określa rozmiar nagłówka IPv4 liczony w 32-bitowych słowach. Ponieważ na 4 bitach pola można zapisać maksymalnie liczbę 15, rozmiar nagłówka IPv4 ograniczony jest do 60 bajtów; w dalszym ciągu rozdziału pokażemy, jak ograniczenie to niweluje w praktyce użyteczność wielu opcji, m.in. opcji *Rejestracja trasy* (*Record Route*). Dla nagłówka pozbawionego opcji pole to ma wartość 5. Ponieważ nagłówek IPv6 ma ustalony rozmiar (40 bajtów), nie występuje w nim analogiczne pole.

[1] Czytelnikom zainteresowanym genezą tej nazwy oraz innymi szczegółami uporządkowania bajtów w słowach wielobajtowych polecam lekturę strony *http://pl.wikipedia.org/wiki/Kolejność_bajtów — przyp. tłum.*

Zgodnie z oryginalną specyfikacją [RFC0791] definiującą protokół IPv4, kolejnym polem w nagłówku IPv4 jest pole definiujące *Typ usługi* (ToS — *Type of Service*); w nagłówku IPv6 analogiczne pole nosi nazwę *Klasa ruchu* (*Trafic Class*), zgodnie ze specyfikacją [RFC2460]. Ponieważ żadne z tych pól nie okazało się szczególnie użyteczne, każde z nich zostało podzielone (w jednakowy sposób w obu wersjach) na mniejsze podpola (który to podział jest przedmiotem rozważań wielu dokumentów RFC, m.in. [RFC3260], [RFC3168] i [RFC2474]). Pierwsze z podpól stanowiących wynik tego podziału zajmuje 6 bitów i nosi nazwę *Usługi Zróżnicowane* (*Differentiated Services*, w skrócie DS), pozostałe dwa bity tworzą podpole o nazwie ECN (od *Explicit Congestion Notification* — jawne powiadomienie o przeciążeniu). Rolę obu podpól w forwardowaniu datagramów omówimy szczegółowo w punkcie 5.2.3.

Pole *Całkowity rozmiar* w nagłówku IPv4 specyfikuje całkowity rozmiar datagramu w bajtach — maksymalnie 65 535, bo tyle można zapisać na 16 bitach. W połączeniu z polem *IHL* umożliwia ono określenie rozmiaru danych stanowiących ładunek użyteczny (miejsce, w którym kończy się nagłówek, a zaczynają wspomniane dane, wyznaczone jest wprost przez zawartość pola *IHL*). Konieczność jawnego wskazania rozmiaru datagramu IPv4 (w polu *Całkowity rozmiar*) wynika z faktu, że niektóre z protokołów warstw niższych stosują zabiegi prowadzące do utraty informacji o tym rozmiarze; koronnym tego przykładem jest Ethernet, wydłużający zbyt małe ramki do minimalnej wymaganej długości 64 bajtów (o czym pisaliśmy w rozdziale 3.) — tak więc ładunek użyteczny ramki w postaci 20-bajtowego datagramu IPv4 zostanie sztucznie uzupełniony do długości 64 bajtów bez wskazania miejsca, w którym rozpoczyna się owo dopełnienie, a kończy się rzeczywisty datagram IP.

Mimo iż wielkość pola *Rozmiar całkowity* (16 bitów) wyznacza 65 535 bajtów jako maksymalny rozmiar datagramu IPv4, większość technologii warstwy łącza danych (w tym Ethernet) nie radzi sobie z tak dużymi datagramami w jednym kawałku, w związku z czym datagramy te podlegają dzieleniu na mniejsze porcje, zwane **fragmentami** (proces tego podziału nosi nazwę **fragmentacji**); ponadto, zgodnie ze specyfikacją IPv4, nie można wymagać od hostów zdolności przetwarzania otrzymywanych datagramów o rozmiarze przekraczającym 576 bajtów (implementacja IPv6 musi natomiast radzić sobie z przetwarzaniem datagramów o wielkości równej MTU łącza transferującego te datagramy IP, przy czym minimalny rozmiar ramki wynosi 1280 bajtów). Wiele aplikacji wykorzystujących protokół UDP (patrz rozdział 10.) do transferu danych (m.in. DNS i DHCP) we własnym zakresie limituje datagramy transportowe do rozmiaru 512 bajtów, automatycznie czyniąc tym samym zadość wspomnianemu ograniczeniu do 576 bajtów. Protokół TCP stosuje własną politykę doboru rozmiaru datagramów IP (patrz rozdział 15.).

Gdy datagram IPv4 podzielony zostaje na kilka fragmentów, każdy z tych fragmentów jest niezależnym datagramem, wartość w polu *Rozmiar całkowity* jest więc rozmiarem fragmentu, a nie oryginalnego datagramu przed podziałem. Szczegółami fragmentacji zajmiemy się w rozdziale 10., przy okazji omawiania protokołu UDP. W datagramie IPv6 fragmentacja nie znajduje odzwierciedlenia w nagłówku; ponadto zamiast rozmiaru całego pakietu w polu *Rozmiar ładunku użytecznego* znajduje się liczba bajtów zajętych przez transmitowane dane oraz nagłówki rozszerzeń, bez uwzględniania jednakże 40-bajtowego nagłówka podstawowego. Tak jak w datagramie IPv4, wielkość pola (16 bitów) ogranicza wspomniany rozmiar do 65 535 bajtów, lecz protokół IPv6 zapewnia także opcję obsługi większych ładunków użytecznych (do 4 GB) w ramach datagramów zwanych jumbogramami (patrz podpunkt 5.3.1.2).

Pole *Identyfikacja* ułatwia protokołowi IPv4 identyfikowanie poszczególnych datagramów wysyłanych przez host. Pole to zwiększane jest systematycznie o 1 w datagramach kolejno wysyłanych przez host. Pole to ma kluczowe znaczenie w sytuacji fragmentowania datagramów, w połączeniu z polami *Znaczniki* i *Offset Fragmentu*; szczegółowo zajmiemy się tym tematem w rozdziale 10. W protokole IPv6 fragmentacja datagramu odzwierciedlana jest w jednym z nagłówków rozszerzenia — powrócimy do tej kwestii w punkcie 5.3.3.

Pole *Czas życia* w IPv4 (i jego odpowiednik *Limit przeskoków* w IPv6) zawiera liczbę przeskoków, czyli routerów, jakie datagram ma prawo jeszcze przemierzyć na swej trasie do miejsca przeznaczenia. Jest ono inicjowane przez nadawcę pewną wartością początkową — dokument [RFC1122] zaleca w tej roli wartość 64, choć często spotyka się także 128 i 255. Pole to jest dekrementowane (zmniejszane) o 1 przy przetwarzaniu w kolejnym routerze; datagram z zerową wartością tego pola jest odrzucany przez router w momencie odebrania, a nadawca powiadamiany jest o tym fakcie za pomocą odpowiedniego komunikatu ICMP (o czym piszemy w rozdziale 8.). Mechanizm ten zapobiega niepożądanym sytuacjom krążenia w nieskończoność po sieci „zapętlonych" datagramów.

W oryginalnej specyfikacji pole *Czas życia* określać miało maksymalny czas życia datagramu w sekundach (stąd nazwa); router po przetworzeniu datagramu zmniejszał to pole o wartość równą liczbie sekund, jakie poświęcił na przetworzenie pakietu; specyfikacja stawiała ponadto oczywiste wymaganie, by wspomniane pole zawsze dekrementowane było co najmniej o 1, nawet jeśli czas przetworzenia pakietu plasował się poniżej sekundy. Ponieważ we współczesnych routerach mało prawdopodobne jest przetrzymywanie datagramu dłużej niż przez sekundę, pole, które pierwotnie pełnić miało rolę czasomierza dekrementowanego zgodnie z upływem czasu, stało się tak naprawdę licznikiem, dekrementowanym przy każdym przeskoku. Ta nowa sytuacja odzwierciedlona została w nazewnictwie protokołu IPv6, zgodnie z którym wspomniane pole otrzymało nową adekwatną nazwę *Limit przeskoków*, zachowując dotychczasową funkcję.

Pole *Protokół datagramu* IPv4 zawiera liczbę identyfikującą protokół, którego dane przenoszone są jako ładunek użyteczny datagramu; najczęściej spotykanymi w tym polu wartościami są 17 (identyfikująca UDP) i 6 (identyfikująca TCP). Pole to zapewnia datagramom IPv4 dużą uniwersalność w postaci zdolności do przenoszenia danych pochodzących z różnych protokołów. Mimo iż w oryginalnej koncepcji protokołu IP ładunek użyteczny zawierał dane pochodzące z protokołu warstwy transportowej, nie jest to warunek konieczny i nic nie stoi na przeszkodzie enkapsulowaniu w datagramach IPv4 jednostek innych protokołów, np. innych datagramów IPv4 (w polu *Protokół* znajduje się wówczas wartość 4) czy IPv6 (wartość 41); oficjalna lista zdefiniowanych identyfikatorów protokołów dostępna jest na stronie [AN].

W datagramie IPv6 pole *Następny nagłówek* jest analogią (i jednocześnie uogólnieniem) pola *Protokół* z IPv4. Zapewnia ono wiązanie w łańcuch ewentualnych nagłówków rozszerzeń oraz pola ładunku użytecznego — niebawem (w podrozdziale 5.3) opiszemy szczegółowo ten mechanizm.

Zadaniem pola *Suma kontrolna nagłówka* jest (zgodnie z nazwą) kontrola integralności samego nagłówka, bez jakiegokolwiek związku z ładunkiem użytecznym. Oznacza to, że protokół IPv4 nie zapewnia sam z siebie ochrony integralności przenoszonego w datagramach ładunku użytecznego, zatem protokoły warstw wyższych, zarządzające tym

ładunkiem, muszą zapewnić własny mechanizm ochrony i weryfikacji jego integralności — i rzeczywiście, większość z tych protokołów (ICMP, IGMP, UDP i TCP) czyni to za pomocą odpowiednich sum kontrolnych, obejmujących ich własne nagłówki i dane, a być może również — uwaga! — niektóre fragmenty zewnętrznego nagłówka IP, te które z perspektywy danego protokołu wydają się szczególnie istotne (co notabene stanowi wyraźne odstępstwo od ścisłej hierarchii warstw).

Być może zaskakujący jest fakt, że nagłówek datagramu IPv6 nie posiada żadnego pola sumy kontrolnej.

Rezygnacja z sumy kontrolnej weryfikującej integralność nagłówka IPv6 była decyzją cokolwiek kontrowersyjną. Podstawową przesłanką tej rezygnacji był fakt, że te protokoły warstwy wyższej, dla których istotna jest integralność nagłówka datagramu IP, same przeprowadzają weryfikację integralności tych obszarów, które wydają się dla nich istotne. Konsekwencją przekłamania w obszarze nagłówka IP może być błędne trasowanie pakietu, błędne wskazanie jego źródła pochodzenia itp., lecz, po pierwsze, przekłamania takie zdarzają się dziś raczej rzadko (za sprawą nowoczesnych łączy światłowodowych przenoszących ruch Internetu), po drugie, jeśli już przekłamanie się zdarzy, to do jego wykrycia i zneutralizowania wymagane są mechanizmy daleko bardziej zaawansowane niż prosta suma kontrolna. Ostatecznie więc, ograniczona potencjalnie przydatność sumy kontrolnej weryfikującej integralność nagłówka IPv6 jest powodem jej nieobecności w tym nagłówku.

Algorytm obliczania wspomnianej sumy kontrolnej wykorzystywany jest w większości innych protokołów związanych z Internetem, dlatego też suma ta nazywana jest popularnie **internetową sumą kontrolną** (*Internet checksum*). Zwróćmy uwagę, że skoro po przejściu datagramu IPv4 przez router zmienia się wartość w jego polu *Czas życia*, to zmieniać się musi również suma kontrolna nagłówka. Wspomniany algorytm omówimy dokładnie w punkcie 5.2.2.

Każdy datagram IP zawiera w nagłówku dwa adresy: źródłowy, odzwierciedlający pochodzenie datagramu, i docelowy, identyfikujący miejsce przeznaczenia tegoż datagramu. Adresy te są 32-bitowe w wersji IPv4 i 128-bitowe w wersji IPv6. Zwykle każdy z nich identyfikuje konkretny interfejs, lecz adresy multicast i broadcast (patrz rozdział 2.) stanowią odstępstwo od tej zasady. Mimo iż liczba możliwych adresów IPv4 — $2^{32} \approx 4{,}5$ miliarda — wydawała się swego czasu ogromna, to ich pula jest obecnie na wyczerpaniu (pisaliśmy o tym w rozdziale 2.) i to właśnie stanowi główną motywację „przesiadki" Internetu na wersję IPv6. Zgodnie z wyliczeniami prezentowanymi w [H05], gdyby równomiernie obdzielić dostępnymi adresami IPv6 cały obszar kuli ziemskiej (z oceanami włącznie), na każdy metr kwadratowy przypadałoby tych adresów 3 911 873 538 269 506 102, co raczej powinno wystarczyć na dość długo.

5.2.2. Internetowa suma kontrolna

Internetowa suma kontrolna jest 16-bitową liczba całkowitą stanowiącą funkcję obszaru danych i wykorzystywaną do zweryfikowania — z dużym prawdopodobieństwem — autentyczności tych danych. Spełnia więc ona rolę taką samą jak nadmiarowa kontrola cykliczna (CRC — patrz rozdział 3. i [PB61]), lecz algorytm jej obliczania jest zupełnie inny — znacznie prostszy, więc zapewniający mniejszy stopień ochrony.

Zakładamy, że chroniony obszar jest ciągiem parzystej liczby bajtów, ciąg o długości nieparzystej dopełniamy bajtem zerowym. Na ciągu tym wykonujemy następnie kolejno następujące operacje.

1. Pary sąsiadujących bajtów łączymy w 16-bitowe słowa według konwencji *big-endian* — bajt o niższym adresie staje się bardziej znaczącym bajtem słowa. W przykładzie przedstawionym na rysunku 5.3 ciąg bajtów 0xE3, 0x4F, 0x23, 0x96, 0x44, 0x27, 0x99, 0xF3 przekłada się w opisanej konwencji na cztery słowa 0xE34F, 0x2396, 0x4427 i 0x99F3, odpowiadające liczbom dziesiętnym 58191, 9110, 17447 i 39411.
2. Kolejne słowa 16-bitowe dodajemy do siebie zgodnie z regułami arytmetyki uzupełnienia do jedności (*one-complement*)[2], w skrócie U1. W przełożeniu na arytmetykę procesorów operujących w arytmetyce „uzupełnień do dwóch" U2 (czyli wszystkich używanych obecnie procesorów) polega to na zwykłym binarnym dodaniu do siebie wspomnianych słów, potraktowanie wyniku jako 32-bitowej liczby binarnej, „złamanie" jej na pół, dodanie do siebie obu połówek i zachowanie 16 najmniej znaczących bitów tak otrzymanego wyniku (to ostatnie dodawanie jest więc dodawaniem modulo 0x1000). Zsumowanie słów, o których mowa w punkcie 1., daje w wyniku 124159 (dziesiętnie), czyli 0x0001E4FF (w przełożeniu na liczbę 32-bitową). Dodając do siebie 0x0001 i 0xE4FF, dostajemy 0xE500.
3. Tworzymy negację bitową wyniku otrzymanego w punkcie 2., dostając w ten sposób wartość żądanej sumy kontrolnej. Dla przykładu z rysunku 5.3:

```
~(0xE500) = ~(1110 0101 0000 0000) = 0001 1010 1111 1111 = 0x1AFF
```

Przedstawiony algorytm wykorzystywany jest do weryfikowania nagłówka w następujący sposób: zerujemy pole *Suma kontrolna nagłówka* i obliczamy dla tego nagłówka sumę kontrolną w sposób powyżej opisany. Otrzymany wynik wpisujemy w pole *Suma kontrolna nagłówka*; gdybyśmy teraz ponownie obliczyli sumę kontrolną nagłówka, okazałoby się, że ma ona wartość 0. Gdy wysłany datagram zostanie odebrany przez router (lub host docelowy), oblicza się sumę kontrolną jego nagłówka i sprawdza, czy faktycznie jest ona zerowa. Otrzymanie innego wyniku świadczy o zniekształceniu nagłówka w czasie transmisji datagramu — datagram jest wówczas odrzucany bez żadnego ostrzeżenia, wykrycie tego faktu i zainicjowanie retransmisji datagramu jest zadaniem protokołów warstw wyższych.

[2] W reprezentacji „uzupełnienia do jedności" liczba ujemna zapisywana jest jako bitowa negacja jej dodatniego odpowiednika. Dodawanie liczb binarnych w tej reprezentacji wykonuje się jak dodawanie zwykłych liczb binarnych (czyli traktując bit znaku na równi z pozostałymi bitami), lecz ewentualne przeniesienie z bitu znaku należy dodać na najmniej znaczącej pozycji. Przykładowo „zwykłe" dodawanie słów 0xE34F i 0x2396 daje w wyniku 0x106E5, czyli w arytmetyce 16-bitowej otrzymamy wynik 0x06E5 oraz przeniesienie 1, które dodane na najmłodszej pozycji daje ostatecznie 0x06E6. Ponieważ arytmetyka uzupełnienia do jedności jest już w świecie komputerów historią — wykorzystywały ją m.in. jednostki centralne komputerów CDC i Univac — rodzi się pytanie o sens jej używania na gruncie nowoczesnych technologii. Otóż podstawową zaletą przedstawionego algorytmu jest jego *symetria względem uporządkowania bajtów*: gdybyśmy mianowicie wykonali grupowanie opisane w punkcie 1. według konwencji *little-endian*, dostalibyśmy identyczną wartość sumy kontrolnej, z zamienioną kolejnością bajtów. Oznacza to, że w implementacjach wykonujących konwersję między uporządkowaniami *big-endian* i *little-endian* obliczanie sumy kontrolnej może następować przed konwersją wysyłanego datagramu i po konwersji odebranego, albo odwrotnie, byle symetrycznie po obu stronach — *przyp. tłum.*

Przed wysłaniem

Komunikat:	E3 4F 23 96 44 27 99 F3 [00 00] ← Pole sumy kontrolnej = 0000
Suma w uzupełnieniu do dwóch:	1E4FF
Suma w uzupełnieniu do jedności:	1E4FF -> E34F + 2396 + 4427 + 99F3 = 1E4FF
Negacja bitowa :	~(E500) = ~(1110 0101 0000 0000) = 0001 1010 1111 1111 = 1AFF (suma kontrolna)

Po odebraniu

Komunikat + suma kontrolna:	E34F + 2396 + 4427 + 99F3 + 1AFF = E500 + 1AFF = FFFF
	~(Komunikat + suma kontrolna) = 0000

Rysunek 5.3. *Internetowa suma kontrolna jako negacja bitowa wyniku sumowania 16-bitowych słów kontrolowanego obszaru w arytmetyce U1 (w przypadku obszaru o nieparzystej liczbie bitów mniej znaczący bajt ostatniego słowa ma wartość 0). Jeżeli pole sumy kontrolnej jest częścią kontrolowanego obszaru (jak w nagłówku IPv4), do pola tego wpisuje się wartość 0, oblicza sumę kontrolną i zapisuje we wspomnianym polu. Ponowne obliczenie sumy kontrolnej dla tak zmodyfikowanego obszaru daje wartość 0, ponieważ wartość znajdująca się w polu sumy kontrolnej jest negacją sumy kontrolnej obliczanej dla pozostałej części obszaru*

5.2.2.1. Własności matematyczne internetowej sumy kontrolnej

Niech $\mathbf{V}$ będzie zbiorem nieujemnych liczb 16-bitowych, $\mathbf{V} = \{0x0001, 0x0002, \ldots, 0xFFFF\}$, niech $\otimes$ symbolizuje operację dodawania dwóch liczb w uzupełnieniu do jedności, opisaną w punkcie 2. scenariusza z punktu 5.2.2. Zbiór $\mathbf{V}$ w połączeniu z operacją $\otimes$ tworzy grupę abelową (przemienną), ponieważ:

1. Operacja $\otimes$ jest *wewnętrzna* w zbiorze $\mathbf{V}$: dla dowolnych $x, y \in \mathbf{V}$, $x \otimes y \in \mathbf{V}$. Jest to oczywiste w przypadku, gdy $0x0001 \le x + y \le 0xFFFF$; w pozostałych przypadkach $0x10000 \le x + y \le 0x1FFFE$, a więc $0x0001 \le x \otimes y \le 0xFFFF$.
2. Operacja $\otimes$ jest *wewnętrzna* w zbiorze $\mathbf{V}$: dla dowolnych $x, y, z \in \mathbf{V}$, $(x \otimes y) \otimes z = x \otimes (y \otimes z)$. Wynika to wprost z łączności dodawania modulo $0x10000$.
3. Element $e = 0xFFFF$ jest w zbiorze $\mathbf{V}$ elementem jednostkowym: dla każdego $x \in \mathbf{V}$, $x \otimes e = e \otimes x = x$. Istotnie: dodawanie $x \otimes 0xFFFF$ zawsze powoduje powstanie przeniesienia równego 1, a więc $x \otimes 0xFFFF = x + 0xFFFF + 1 = (x + 0xFFFF) \bmod 0x10000 + 1$. Ponieważ $(x + 0xFFFF) \bmod 0x10000 < 0xFFFF$, więc $(x + 0xFFFF) \bmod 0x10000 + 1 = (x + 0xFFFF + 1) \bmod 0x10000 = x \bmod 0x10000 + (0xFFFF + 1) \bmod 0x10000 = x + (0x10000 \bmod 0x10000) = x + 0 = x$.
4. Dla każdego $x \in \mathbf{V}$ istnieje *dokładnie jeden* element odwrotny $x^{-1} \in \mathbf{V}$ taki, że $x + x^{-1} = e = 0xFFFF$. Po chwili zastanowienia staje się oczywiste, że dla $x < 0xFFFF$ element x^{-1} jest bitową negacją x. Element $x = 0xFFFF = e$ jest swoją własną odwrotnością (musi nią być jako element jednostkowy).
5. Działanie $\otimes$ jest przemienne: dla dowolnych $x, y \in \mathbf{V}$, $x \otimes y = y \otimes x$ — co wynika z przemienności zwykłego dodawania.

Zwróćmy uwagę, że zbiór $\mathbf{V}$ przestałby być grupą ze względu na działanie $\otimes$, gdybyśmy włączyli do niego element $0x0000$. Dla elementu $x = 0xFFFF$ istniałyby wówczas *dwa* elementy y spełniające równanie $x \otimes y = e = 0xFFFF$: $y = 0x0000$ oraz $y = 0xFFFF$. Nie byłby więc spełniony warunek jednoznaczności elementu odwrotnego, wyartykułowany w punkcie 4.

Czytelnikom pragnącym zapoznać się z podstawami algebry abstrakcyjnej polecamy książkę [P90].

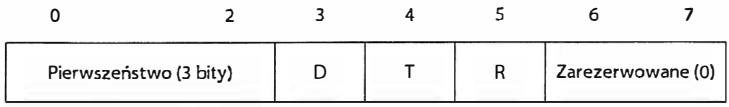
5.2.3. Pola DS i ECN (dawniej ToS i Klasa ruchu)

Pola trzecie i czwarte nagłówka IPv4 (drugie i trzecie w nagłówku IPv6), czyli *Usługi zróżnicowane* i *ECN*, związane są (ogólnie rzecz biorąc) z mechanizmami modyfikującymi standardowe forwardowanie datagramów. Pod pojęciem różnicowania usług rozumiemy świadczenie ich w standardach odbiegających poza rutynowy ruch niegwarantowany (*best-effort delivery*), głównie w oparciu o priorytety transmisji (patrz dokumenty [RFC2474], [RFC2475] i [RFC3260]). Wyższy priorytet oznacza większe uprzywilejowanie datagramu i krótsze (w stosunku do datagramów o niższym priorytecie) oczekiwanie w kolejce do przetworzenia przez router. Liczba znajdująca się w polu *Usługi zróżnicowane* nazywana bywa **punktem kodowym** (*Differentiated Services Code Point*, w skrócie DSCP). Każdy z „punktów kodowych" jest kombinacją bitów o uzgodnionym a priori znaczeniu. Zazwyczaj datagram opatrywany jest odpowiednią wartością DSCP przy wejściu do infrastruktury sieciowej, która to wartość pozostaje niezmieniona przez cały czas wędrówki datagramu przez sieć; często jednak polityka administracyjna (np. w kwestii limitowania liczby uprzywilejowanych pakietów transmitowanych w jednostce czasu) prowadzić może do modyfikowania DSCP pakietu (w kierunku zmniejszenia jego uprzywilejowania).

Para bitów tworzących pole ECN wykorzystywana jest do opatrywania datagramu tzw. **indykatorem przeciążenia** w sytuacji, gdy datagram ten przechodzi przez router, w którego kolejce czeka na przetworzenie duża liczba pakietów; routery implementujące funkcję powiadamiania o przeciążeniu ustawiają wówczas na 1 oba bity. W zamierzeniu projektantów tego mechanizmu, gdy oznakowany w ten sposób pakiet dotrze do miejsca przeznaczenia, odpowiednio inteligentny protokół (np. TCP) zostanie w ten sposób powiadomiony o trudnej sytuacji w warstwie sieciowej i zasygnalizuje nadawcy konieczność spowolnienia tempa wysyłania nowych pakietów. Jest to jeden z mechanizmów przeznaczonych do unikania przeciążeń i radzenia sobie z nimi; dokładniejszemu omówieniu tych mechanizmów poświęcamy rozdział 16.

Chociaż pola *Usługi zróżnicowane* i *ECN* nie wydają się mieć ze sobą ścisłego związku, to jednak mają wspólną genealogię, jak wcześniej wspomnieliśmy, są wynikiem podziału pola *Typ usługi* (w IPv6 pola *Klasa ruchu*). Często więc opisywane są łącznie, a wymienione oryginalne nazwy w dalszym ciągu są sankcjonowane w literaturze, także w różnych odmianach („bajt ToS", „bajt klasy ruchu" itp.). I mimo że oryginalna koncepcja pojedynczego pola *Typ usługi/Klasa ruchu* nigdy nie doczekała się pełnego wsparcia, to jednak we współczesnych implementacjach IP obsługa pola *Usługi zróżnicowane* zrealizowana została z myślą o kompatybilności wstecz (w pewnym stopniu) z pierwowzorem. Aby wyjaśnić szczegóły tej koncepcji, przyjrzyjmy się wpierw oryginalnej strukturze pola *Typ usługi*, przedstawionej na rysunku 5.4 (zaczerpniętym ze specyfikacji [RFC0791]).

Bity *D*, *T* i *R* wyrażają potrzebę szczególnego traktowania pakietu, pod względem małego opóźnienia, dużej przepustowości i wysokiej niezawodności (zgodnie z podpisem pod rysunkiem 5.4). Wartości priorytetu (pierwszeństwa) zmieniają się od 0 (rutynowy) do 7 (krytyczny z punktu widzenia zarządzania siecią). Szczegółowe znaczenie każdej



Rysunek 5.4. *Oryginalna struktura pola Typ usługi (w IPv6 Klasa ruchu). Wartość w polu Pierwszeństwo określa stopień uprzywilejowania (priorytetu) pakietu — większa wartość oznacza wyższy priorytet. Ustawienie bitów D, T lub R jest sygnałem, że dla danego pakietu szczególnie pożądane jest (odpowiednio) małe opóźnienie (delay), duża przepustowość (throughput) i niezawodność dostarczenia (reliability)*

wartości priorytetu przedstawiono w tabeli 5.1; bazuje ono na schemacie wywłaszczania zwanym *MultiLevel Precedence and Preemption* (MLPP), datującym się jeszcze z czasów systemu telefonicznego AUTOVON departamentu obrony USA (patrz [AUTOVON]), w którym rozmowa o niższym priorytecie mogła zostać przerywana („wywłaszczona" z przydziału linii telefonicznej) na rzecz połączenia o wyższym priorytecie. Widoczne w tabeli nazwy są w dalszym ciągu obowiązujące i zostały zaadaptowane na gruncie VoIP.

Tabela 5.1. *Oryginalne znaczenie podpola Pierwszeństwo w polu Typ usługi/Klasa ruchu*

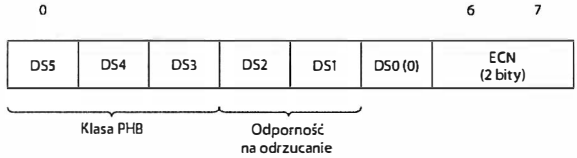
Wartość	Nazwa priorytetu
000	Zwyczajny (*routine*)
001	Nadrzędny (*priority*)
010	Natychmiastowy (*immediate*)
011	Błyskawiczny (*flash*)
100	Ponadbłyskawiczny (*flash override*)
101	Krytyczny (*critical*)
110	Sterowanie międzysiecią (*internetwork control*)
111	Sterowanie siecią (*network control*)

Z każdą wartością DSCP związana jest określona strategia routera w zakresie forwardowania pakietów, zwana *zachowaniem na przeskoku* (*Per-Hop Behavior*, w skrócie PHB). Definiując PHB dla poszczególnych DSCP, projektanci starali się zachować kompatybilność z pierwowzorem, czyli zachowaniem, jakie wynikałoby z mapowania bitowego wzorca DSCP na dawną strukturę podpól *Pierwszeństwo*, *D*, *T* i *R*. Przykładowo DSCP równe 22 — binarnie 010110 — odwzorowuje się na *Pierwszeństwo* równe 010 i ustawione bity *D* i *T*, tak więc PHB wynikające z owego DSCP powinno charakteryzować się priorytetem na poziomie „Natychmiastowy" i zapewnieniem pakietowi jak najmniejszego opóźnienia i jak największej przepustowości. Zamiar ten został jednak zrealizowany w ograniczonym zakresie, bowiem zbiór 64 możliwych wartości DSCP podzielony został na trzy grupy, identyfikowane najmniej znaczącymi bitami: numery DSCP kończące się bitem 0 przeznaczone są do standardowego użytku, te kończące się kombinacją 11 mają przeznaczenie wyłącznie eksperymentalne, pozostałe — czyli te z końcówką 01 — mają obecnie charakter eksperymentalny, lecz w przyszłości mogą stać się elementami standardu. Klasyfikację tę ilustruje tabela 5.2, zaczerpnięta z [DSCPREG].

Tabela 5.2. *Podział możliwych wartości na trzy grupy: standardowe, wyłącznie eksperymentalne i eksperymentalne z możliwością standaryzacji*

Grupa	Szablon DSCP	Przeznaczenie
1	*xxxxx*0	Użytek standardowy.
2	*xxxx*11	Wykorzystywanie eksperymentalne i w zastosowaniach lokalnych.
3	*xxxx*01	Wykorzystywanie eksperymentalne i w zastosowaniach lokalnych, z możliwością ewentualnej standaryzacji w przyszłości.

Wspomniane odwzorowanie DSCP na strukturę *Pierwszeństwo-D-T-R* zostało więc zrealizowane połowicznie, bowiem (zgodnie z tabelą 5.2) bit *R* ma w tym odwzorowaniu zawsze wartość 0. Ostatecznie trzy najbardziej znaczące bity DSCP identyfikują tzw. *klasę* PHB, pozostałe dwa natomiast odzwierciedlają *odporność pakietu na odrzucanie*: im większa wartość tej odporności, tym mniejsze prawdopodobieństwo, że w sytuacji przeciążenia router odrzuci ten właśnie pakiet[3]. Mapowanie to przedstawiono schematycznie na rysunku 5.5.



Rysunek 5.5. *Znaczenie poszczególnych grup bitów DSCP, wynikające z (częściowej) kompatybilności ze znaczeniem pola Typ usługi/Klasa ruchu. W grupie standardowych DSCP bit DS0 ma zawsze wartość 0. 2-bitowe pole ECN zawiera powiadomienie dla odbiorcy, że przetworzenie datagramu odbyło się w warunkach permanentnego przeciążenia routera, co dla tegoż odbiorcy stanowić może sygnał dla podjęcia stosownych działań, np. zażądania od nadawcy spowolnienia tempa transmisji*

Zgodnie z rysunkiem 5.5, podpole *Klasa PHB* jest odpowiednikiem podpola *Pierwszeństwo* w dawnym polu *Typ usługi/Klasa ruchu*. „Klasa PHB" oznacza tu odpowiedni sposób traktowania datagramów — datagramy zawierające tę samą wartość na bitach *DS5*, *DS4* i *DS3* (wartość ta nazywana jest *selektorem klasy*) traktowane są przez router według tej samej strategii. W ramach danej klasy PHB pakiety mogą być zróżnicowane pod względem preferowania przez router w sytuacji wymagającej odrzucania pakietów; jedna z czterech możliwych wartości kombinacji bitów *DS2* i *DS1* jest wskaźnikiem odporności pakietu na odrzucenie: 01 oznacza odrzucanie w pierwszej kolejności (w ramach danej klasy PHB), wartość 11 — odrzucanie tylko w ostateczności (ponownie — w ramach tej samej klasy PHB). Zerowa wartość obu bitów *DS2* i *DS1* oznacza brak jawnej preferencji dla pakietu pod względem odporności na odrzucanie — taka (domyślna) strategia nazywana jest często *zgodnym z klasami selektorów zachowaniem na przeskoku* (*class*

[3] W literaturze polskiej nazwa *Drop probability* tłumaczona jest powszechnie jako „Prawdopodobieństwo odrzucenia pakietu" i choć lingwistycznie wszystko jest w porządku, należy zdawać sobie sprawę z faktu, że związek wartości pola *Drop probability* ze wspomnianym prawdopodobieństwem jest dokładnie odwrotny — *największa* możliwa wartość tego pola (11) oznacza *najmniejsze* prawdopodobieństwo odrzucenia pakietu. Wartość ta jest zatem nie tyle *prawdopodobieństwem* odrzucenia, co raczej umownym *znacznikiem* tego prawdopodobieństwa — *przyp. tłum.*

selektor compliant PHBs); w zamierzeniu projektantów ma ona realizować (częściową) kompatybilność wstecz z oryginalną funkcją pola *Pierwszeństwo* sformułowaną w [RFC0791].

Spośród możliwych 32 wartości DSCP odpowiadających szablonowi *xxxxx*0 do standardowego użytku wybrano tylko niektóre — ich zestawienie znajduje się w tabeli 5.3. Dla każdej kombinacji podano jej oficjalne oznaczenie, definiujący ją dokument oraz wyjaśnienie znaczenia, także z nawiązaniem do dawnej struktury podpola *Pierwszeństwo*. Na początku widzimy więc selektory klas i ich odpowiedniki z tabeli 5.1, kolejne pozycje reprezentują grupy tzw. **gwarantowanego forwardowania** (*assured forwarding*): 12 grup oznaczonych AF<*x*><*y*> to rezultat możliwych kombinacji wartości selektorów klas <*x*> od 1 do 4 i wskaźników podwyższonej odporności na odrzucanie <*y*> od 1 do 3. W ramach selektora klasy 5 wyróżniono dwa poziomy preferencyjnego ruchu: **przyspieszone forwardowanie** (*expedited forwarding*) oraz forwardowanie ograniczone jedynie dostępnym pasmem (*capacity-admitted*), mające absolutne pierwszeństwo przed innymi DSCP.

Tabela 5.3. *Wykorzystywane standardowo wartości DSCP, częściowo kompatybilne z pierwowzorem w postaci pola Typ usługi/Klasa ruchu. Grupy AF i EF oznaczają strategię preferencyjną w stosunku do ruchu niegwarantowanego (best-effort delivery)*

Oznaczenie	Wartość (binarnie)	Dokument	Znaczenie
CS0	000000	[RFC2474]	Selektor klasy (ruch niegwarantowany)
CS1	001000	[RFC2474]	Selektor klasy (ruch nadrzędny)
CS2	010000	[RFC2474]	Selektor klasy (ruch natychmiastowy)
CS3	011000	[RFC2474]	Selektor klasy (ruch błyskawiczny)
CS4	100000	[RFC2474]	Selektor klasy (ruch ponadblyskawiczny)
CS5	101000	[RFC2474]	Selektor klasy (ruch o znaczeniu krytycznym)
CS6	110000	[RFC2474]	Selektor klasy (sterowanie międzysiecią)
CS7	111000	[RFC2474]	Selektor klasy (sterowanie siecią)
AF11	001010	[RFC2597]	Gwarantowane forwardowanie (assured forwarding) (klasa 1, odporność 1)
AF12	001100	[RFC2597]	Gwarantowane forwardowanie (1,2)
AF13	001110	[RFC2597]	Gwarantowane forwardowanie (1,3)
AF21	010010	[RFC2597]	Gwarantowane forwardowanie (2,1)
AF22	010100	[RFC2597]	Gwarantowane forwardowanie (2,2)
AF23	010110	[RFC2597]	Gwarantowane forwardowanie (2,3)
AF31	011010	[RFC2597]	Gwarantowane forwardowanie (3,1)
AF32	011100	[RFC2597]	Gwarantowane forwardowanie (3,2)
AF33	011110	[RFC2597]	Gwarantowane forwardowanie (3,3)
AF41	100010	[RFC2597]	Gwarantowane forwardowanie (4,1)
AF42	100100	[RFC2597]	Gwarantowane forwardowanie (4,2)
AF43	100110	[RFC2597]	Gwarantowane forwardowanie (4,3)
EFPHB	101110	[RFC3246]	Forwardowanie przyspieszone (*expedited forwarding*)
VOICE-ADMIT	101100	[RFC5865]	Ruch uwarunkowany pasmem (*capacity-admitted traffic*)

Usługa przyspieszonego forwardowania (*expedited forwarding*) jest de facto uwolnieniem forwardowanego pakietu od kontekstu ewentualnego przeciążenia routera: jeśli pakiet taki w ogóle musi oczekiwać w kolejce, to tylko z powodu przetwarzania innego pakietu z DSCP równym EFPHB. W efekcie pakiety tej grupy rzadko są gubione, a dane przekazywane za ich pośrednictwem doświadczają małych opóźnień (i w konsekwencji małych błędów *jitter*).

Zagadnienie świadczenia usług zróżnicowanych było przedmiotem znaczącego wysiłku projektantów i inżynierów u schyłku XX wieku. Mimo iż zabiegi standaryzacyjne uwieńczone zostały powodzeniem jeszcze w latach 90., to dopiero po roku 2000 niektóre z mechanizmów tej kategorii doczekały się praktycznych zastosowań. Stało się to z przyczyn natury nie tyle technicznej, co raczej ekonomicznej: usługi zróżnicowane dają klientowi różnego rodzaju dodatkowe korzyści i przywileje, za które ten powinien odpowiednio płacić i właśnie kwestia sprawiedliwości systemu opłat jest tu problemem skomplikowanym, wymykającym się możliwościom dyskusji w ograniczonych ramach książki. Zainteresowani czytelnicy znajdą wiele szczegółów tego tematu w książce [MB97] i publikacji [W03].

5.2.4. Opcje IP

Proces przetwarzania datagramu IP może być modyfikowany za pomocą rozmaitych opcji o charakterze lokalnym, czyli dotyczących wyłącznie tego datagramu. Wiele tych opcji zostało zdefiniowanych wraz z oryginalną definicją protokołu IPv4 w dokumencie [RFC0791]; Internet miał wówczas rozmiary znacznie mniejsze od obecnych, mniejsze też było zagrożenie czyhające ze strony niesubordynowanych internautów. Gdy poczęły się kształtować zręby protokołu IPv6, rzeczywistość była już wyraźnie inna, wskutek czego niektóre z opcji straciły rację bytu, zaś użyteczność innych okazała się ograniczona z powodu limitowanych rozmiarów samego nagłówka IPv4, jeszcze inne okazały się wątpliwe z perspektywy bezpieczeństwa sieci. Projektanci IPv6 postanowili więc o generalnym usunięciu opcji IP z nagłówka datagramu i umieszczeniu ich w nowym elemencie — **nagłówku rozszerzeń**, umiejscowionym w pakiecie między nagłówkiem podstawowym a obszarem ładunku użytecznego. Nagłówki rozszerzeń przetwarzane są generalnie dopiero przez host końcowy, wyjątkowo jednak niektóre z nich przetwarzane są także przez routery pośredniczące. W niektórych routerach obecność nagłówków rozszerzeń powoduje spowolnienie przetwarzania datagramu, nawet jeśli nagłówki te nie są przez router przetwarzane.

Rozpoczniemy od omówienia opcji IPv4, po czym zajmiemy się opcjami i nagłówkami rozszerzeń IPv6. W tabeli 5.4 widoczny jest podsumowujący wykaz opcji IPv4, jakie z biegiem lat zyskały sobie rangę standardów. Lista opcji IP nie jest kompletna — jest okresowo uaktualniana i dostępna pod adresem [IPPARAM].

Tabela 5.4. *Opcje datagramu IPv4, jeśli występują, umiejscowione są bezpośrednio po obowiązkowych, podstawowych polach nagłówka. Każda opcja identyfikowana jest na podstawie 8-bitowego pola Typ opcji, podzielonego na trzy podpola, reprezentujące kopiowanie (1 bit), klasę (2 bity) i numer (5 bitów). Dla niektórych opcji pole to jest jedynym polem, większość opcji ma jednak strukturę bardziej rozbudowaną: po 8-bitowym polu Typ opcji występuje 8-bitowe pole wskazujące Rozmiar opcji, potem następują Dane opcji (w polu Rozmiar opcji uwzględnione są wszystkie trzy pola)*

Nazwa	Numer opcji	Typ opcji (bajt identyfikacyjny)	Rozmiar	Opis	Dokument	Komentarz
Koniec listy	0	0	1	Wskazuje koniec obszaru opcji.	[RFC0791]	Nie występuje w przypadku braku opcji.
Opcja pusta	1	1	1	Opcja niereprezentująca żadnego znaczenia.	[RFC0791]	Występuje między opcjami, np. w celu wyrównania początku następnej opcji do granicy 32-bitowego słowa.
Trasowanie źródłowe (*Source Routing*)	3 9	131 137	Zmienny	Zawiera (stworzoną przez nadawcę) listę routerów pośredniczących w forwardowaniu pakietu. Występuje w dwóch odmianach: typ 3 oznacza zezwolenie na uzupełnienie wspomnianej listy o dodatkowe pozycje, typ 9 oznacza listę ściśle ustaloną.	[RFC0791]	Rzadko używana, zwykle filtrowana.
Etykiety bezpieczeństwa i zarządzania (*Security and Handling Labels*)	2 5	130 133	11	Określa sposób dołączania etykiet bezpieczeństwa i ograniczeń w przetwarzaniu datagramów IP w zastosowaniach militarnych USA.	[RFC1108]	Obecnie niewykorzystywana.
Rejestracja trasy (*Record Route*)	7	7	Zmienny	Zawiera listę routerów, jakie dotychczas przebył pakiet na swej trasie.	[RFC0791]	Rzadko używana.
Znakowanie czasowe (*Timestamp*)	4	68	Zmienny	Zawiera listę znaczników czasowych, odzwierciedlających momenty przetwarzania przez poszczególne węzły.	[RFC0791]	Rzadko używana.

Tabela 5.4. — ciąg dalszy

Nazwa	Numer opcji	Typ opcji (bajt identyfikacyjny)	Rozmiar	Opis	Dokument	Komentarz
Identyfikator strumienia (*Stream ID*)	8	136	4	Zawiera 16-bitowy identyfikator strumienia SATNET.	[RFC0791]	Obecnie niewykorzystywana.
EIP	17	145	Zmienny	Opcja *Extended Internet Protocol* — eksperymentalnego protokołu z lat 90. ubiegłego wieku.	[RFC1385]	Obecnie niewykorzystywana.
Śledzenie trasy (*Traceroute*)	18	82	Zmienny	Powoduje śledzenie trasy i wysyłanie komunikatów ICMP — w zastosowaniach eksperymentalnych z lat 90. ubiegłego wieku.	[RFC1393]	Obecnie niewykorzystywana.
Alarm dla routera (*Router Alert*)	20	148	4	Wskazuje konieczność interpretowania przez router treści datagramu.	[RFC2113] [RFC5350]	Używana okazjonalnie.
Szybki start (*Quick-Start*)	25	25	8	Wskazuje szybki start protokołu transportowego.	[RFC4782]	Opcja eksperymentalna, rzadko używana.

Obszar opcji w nagłówku IPv4 kończy się na granicy słowa 32-bitowego (ze względu na charakter pola *IHL* długość nagłówka musi być wielokrotnością 4 bajtów). Pole *Typ opcji* jest zawsze pierwszym bajtem opcji, identyfikującym ją. Najbardziej znaczący bit tego bajta określa, czy w razie fragmentacji datagramu opcja ma być kopiowana do nagłówka każdego fragmentu (1), czy też ma pojawić się wyłącznie w nagłówku pierwszego fragmentu (0). Dwa kolejne bity określają klasę opcji; spośród opcji wymienionych w tabeli 5.4 opcje *Znakowanie czasowe* i *Śledzenie trasy* są opcjami klasy 2 (*debugging and measurement* — debugowanie i pomiary), pozostałe opcje są opcjami klasy 0. Klasy 1 i 3 są zarezerwowane do przyszłego użytku. Pięć najmniej znaczących bitów bajta identyfikacyjnego składa się na *Numer opcji* (wyjaśnia to różnicę między drugą i trzecią kolumną tabeli). Niektóre opcje reprezentowane są tylko przez swój bajt identyfikacyjny, niektóre natomiast posiadają bardziej rozbudowaną strukturę; w tym drugim przypadku bezpośrednio po bajcie identyfikacyjnym następuje bajt jawnie wskazujący *Rozmiar*, czyli liczbę bajtów owej struktury (począwszy od bajta identyfikacyjnego do ostatniego bajta danych). Przynależność opcji do jednej z tych dwóch kategorii wynika z jej numeru — spośród opcji wymienionych w tabeli 5.4 tylko opcje *Koniec listy* i *Opcja pusta* należą do kategorii pierwszej.

Większość z wymienionych opcji standardowych używana jest dziś rzadko lub wcale, także z powodu przyrodzonych ograniczeń protokołu IPv4, a konkretnie — ograniczenia rozmiaru nagłówka do 60 bajtów. Przykładowo do dyspozycji opcji *Trasowanie źródłowe* lub *Rejestracja trasy* stoi w najlepszym przypadku 40 bajtów (20 ze wspomnianego limitu zajęte jest przez obowiązkowe pola nagłówka), co jest niewystarczające w sytuacji, gdy

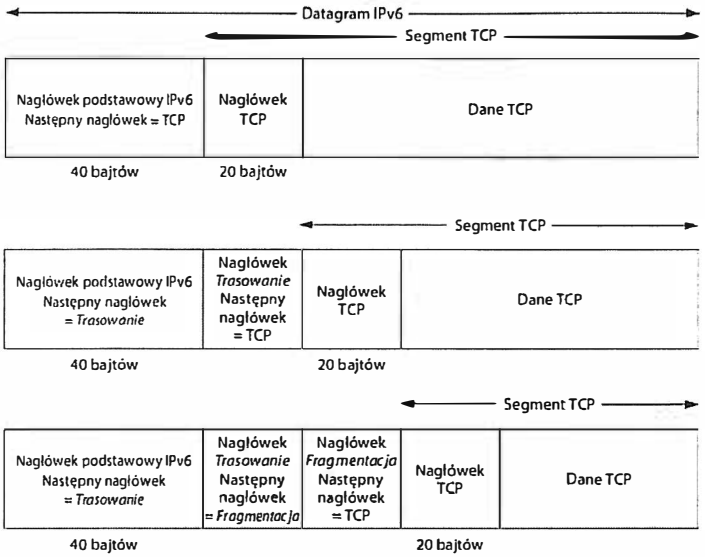
(jeśli wierzyć artykułowi [LFS07]) typowy datagram IP przebywa na swej trasie średnio 15 przeskoków. Ponadto większość z opcji ma charakter diagnostyczny i jedną z konsekwencji ich istnienia jest większa komplikacja konstrukcji firewalli i większe ryzyko związane z definiowaniem wyjątków w ich konfiguracji. Niejako wyjątkiem jest w tym względzie opcja *Alarm dla routera*, wskazująca na konieczność przetworzenia pakietu przez router w sposób wykraczający poza konwencjonalne forwardowanie. Eksperymentalną opcję *Szybki start*, wykorzystywaną zarówno w IPv4, jak i IPv6, omówimy w następnym podrozdziale, w związku z nagłówkami rozszerzeń IPv6.

5.3. Nagłówki rozszerzeń IPv6

W protokole IPv6 opcje i inne funkcje dodatkowe potraktowane zostały w sposób całkowicie odmienny w stosunku do IPv4. Ponieważ z założenia używane są opcjonalnie i wybiórczo, uznano za niecelowe lokowanie ich w obrębie nagłówka podstawowego, obarczonego dotkliwym ograniczeniem rozmiaru; znalazły swe miejsce w obszarze nagłówków rozszerzeń (*extension headers*) ułożonych w łańcuch, którego ostatnim ogniwem jest obszar danych protokołu warstwy wyższej, co (w kilku wariantach) przedstawiono na rysunku 5.6. Nagłówek podstawowy ma dzięki temu ustaloną strukturę i ustalony rozmiar 40 bajtów (patrz rysunek 5.2). Ponieważ nagłówek podstawowy jest jedynym elementem datagramu przetwarzanym przez routery pośredniczące (istnieje jeden ważny wyjątek od tej reguły), upraszcza się algorytm obsługi datagramu przez router, co w zamierzeniu powinno prowadzić do skrócenia czasu tej obsługi (choć czas ten zależny jest także od wielu innych czynników, takich jak złożoność protokołu, możliwości sprzętu, wydajność oprogramowania czy bieżące obciążenie routera).

Jak pokazano na rysunku 5.6, ciąg nagłówków rozszerzeń (o ile występują) umiejscowiony jest między podstawowym nagłówkiem IPv6 a obszarem protokołu warstwy wyższej (na rysunku jest nim protokół TCP). Elementem wiążącym nagłówki rozszerzeń w łańcuch jest pole *Następny nagłówek*, wskazujące typ następnego nagłówka, zgodnie z tabelą 5.5. Z definicji ostatnim elementem wspomnianego łańcucha jest obszar protokołu warstwy wyższej; specyfikacja IPv6 dopuszcza jednak brak tego obszaru, wówczas w polu *Następny nagłówek* ostatniego ogniwa łańcucha znajduje się wartość 59, jawnie wskazująca koniec łańcucha (patrz sekcja 4.7 dokumentu [RFC2460]). Kompletna lista wartości dopuszczalnych w polu *Następny nagłówek* dostępna jest na stronie [IP6PARAM], w tabeli 5.5 przedstawiono tylko wybrane.

Wskazywana w drugiej kolumnie kolejność występowania nagłówków rozszerzeń jest jedynie zaleceniem, a nie bezwzględnym wymogiem; wyjątkiem są opcje „Skok po skoku", które — o ile występują — muszą znaleźć się bezpośrednio za nagłówkiem podstawowym. Z perspektywy implementatora oznacza to, z jednej strony, dość dużą dowolność w zakresie wspomnianej kolejności, z drugiej natomiast, wymaga od implementacji przetworzenia dowolnego nagłówka niezależnie od pozycji, na której występuje. Każdy z typów nagłówka rozszerzenia może wystąpić co najwyżej jeden raz, wyjątkiem jest nagłówek *Opcje docelowe*, który może wystąpić dwukrotnie: pierwszy raz w związku z docelowym adresem IP zawartym w nagłówku podstawowym IPv6, drugi raz w związku z adresem IP określającym ostateczne przeznaczenie datagramu — w niektórych sytuacjach adresy te mogą się różnić, np. w przypadku użycia nagłówka *Trasowanie* docelowy adres IP w nagłówku podstawowym może zmieniać się przy przechodzeniu przez kolejne routery.



Rysunek 5.6. *Trzy przykłady umiejscowienia łańcucha nagłówków rozszerzeń w datagramie IPv6. Elementem wiążącym nagłówki w łańcuch jest pole Następny nagłówek, ostatnim elementem tego łańcucha jest obszar protokołu warstwy wyższej (tu TCP) albo nagłówek jawnie oznaczony jako ostatni, czyli zawierający wartość 59 we wspomnianym polu*

Tabela 5.5. *Wartość w polu Następny nagłówek może wskazywać na nagłówek rozszerzenia bądź obszar protokołu; większość z tych wartości rozpoznawalna jest również w polu Protokół nagłówka IPv4*

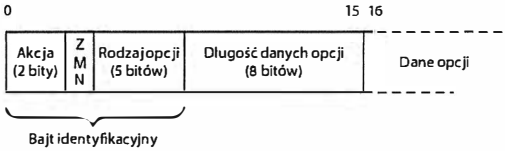
Typ nagłówka	Pozycja w łańcuchu	Wartość	Uwagi
Nagłówek podstawowy IPv6	1	41	[RFC2460] [RFC2473]
Opcje „skok po skoku" (HOPOPT)	2	0	[RFC2460] Muszą wystąpić bezpośrednio po nagłówku podstawowym.
Opcje docelowe	3,8	60	[RFC2460]
Trasowanie	4	43	[RFC2460] [RFC5095]
Fragmentacja	5	44	[RFC2460]
Nagłówek ESP (*Encapsulating Security Payload*)	7	50	Patrz rozdział 18.
Nagłówek uwierzytelniający (AH —*Authentication Header*)	6	51	Patrz rozdział 18.
Mobilne IP (MIPv6)	9	135	[RFC6275]

Tabela 5.5. *Wartość w polu Następny nagłówek może wskazywać na nagłówek rozszerzenia bądź obszar protokołu; większość z tych wartości rozpoznawalna jest również w polu Protokół nagłówka IPv4 — ciąg dalszy*

Typ nagłówka	Pozycja w łańcuchu	Wartość	Uwagi
Koniec łańcucha	Ostatnia	59	[RFC2460]
ICMPv6	Ostatnia	58	Patrz rozdział 8.
UDP	Ostatnia	17	Patrz rozdział 10.
TCP	Ostatnia	6	Patrz rozdziały 13. – 17.
Protokoły warstw wyższych	Ostatnia	—	Kompletna lista znajduje się na stronie [AN].

5.3.1. Opcje IPv6

Nowa koncepcja realizacji opcjonalnych elementów przetwarzania datagramu IP, czyli mechanizm nagłówków rozszerzeń IPv6, stwarza dla tych elementów bardziej komfortowe środowisko, wolne od skrępowania wynikającego chociażby z ograniczonego rozmiaru nagłówka IPv4. Oznacza to m.in. przywrócenie do życia niektórych opcji IPv4, które wskutek owego skrępowania stały się praktycznie bezużyteczne w obliczu nowej rzeczywistości internetowej — mowa tu m.in. o opcjach *Trasowanie źródłowe* oraz *Rejestracja trasy*, o czym wcześniej wspominaliśmy. Protokół IPv6 dzieli swe opcje na dwie podstawowe grupy: te uwzględniane jedynie przez docelowy host (*Opcje docelowe*) oraz uwzględniane przez każdy router na trasie datagramu (zwane opcjami „Skok po skoku"). Sposób kodowania opcji jest jednakowy dla obu grup, przedstawiamy go na rysunku 5.7.



Rysunek 5.7. *Schemat TLV kodowania opcji IPv6. Pierwszy bajt, identyfikujący typ opcji, dzieli się na trzy podpola: podpole Akcja określa sposób postępowania w przypadku nierozpoznania opcji przez router lub host, podpole ZMN oznacza zezwolenie (1) albo brak zezwolenia (0) na modyfikowanie opcji przez routery pośredniczące, zaś w polu Rodzaj opcji znajduje się identyfikator określający funkcję spełnianą przez opcję. W drugim bajcie wskazany jest jawnie rozmiar danych związanych z opcją*

Trzy elementy opcji określają kolejno jej identyfikację, długość i dane, co znajduje odzwierciedlenie w akronimie TLV określającym ten schemat (to skrót od *Type-Length-Value*). Pierwszy bajt — identyfikacyjny — podzielony jest na trzy podpola, określające: sposób postępowania w przypadku, gdy opcja nie zostanie rozpoznana lub nie jest implementowana w routerze (podpole *Akcja*), zezwolenie (albo jego brak) na modyfikowanie opcji przez routery pośredniczące (*ZMN*) oraz *Rodzaj opcji* (stanowiący odpowiednik numeru opcji z IPv4). Dwa bity w podpolu *Akcja* pozwalają na zakodowanie czterech różnych wariantów postępowania, zgodnie z tabelą 5.6.

Tabela 5.6. *Dwa najbardziej znaczące bity bajta Typ opcji określają sposób reakcji routera na napotkanie nierozpoznanej lub niezaimplementowanej opcji*

Wartość	Podejmowana akcja
00	Zignorowanie opcji, kontynuowanie przetwarzania datagramu.
01	Odrzucenie datagramu bez ostrzeżenia.
10	Odrzucenie datagramu i wysłanie do jego nadawcy komunikatu ICMPv6 Parameter Problem.
11	Odrzucenie datagramu i wysłanie do jego nadawcy komunikatu ICMPv6 Parameter Problem, o ile adres docelowy datagramu IP nie jest adresem multicast.

Komunikat ostrzegający o odrzuceniu datagramu stwarza sytuację bardziej klarowną niż odrzucenie bez ostrzeżenia, jednakże w sytuacji, gdy adres docelowy datagramu jest adresem multicast, jego odrzucenie mogłoby wywołać lawinę takich komunikatów. Fakt ten jest powodem rozróżnienia akcji identyfikowanych kodami 10 i 11. Tolerowanie przez router niezrozumiałych dla niego opcji jest cechą niezmiernie pożyteczną z punktu widzenia projektowania i testowania nowych opcji: routery nieimplementujące obsługi nowej opcji po prostu ją ignorują, dzięki czemu może być ona wdrażana w sposób przyrostowy, czyli implementowana selektywnie na poszczególnych routerach.

Gdy bit *ZMN* ma wartość 1, dopuszczalne jest modyfikowanie opcji przez router (w szczególności —wyzerowanie tego bitu), w przeciwnym razie opcja musi pozostać w dotychczasowej postaci.

Zestawienie zdefiniowanych obecnie opcji IPv6 znajduje się w tabeli 5.7. W pierwszej kolumnie figuruje oficjalna nazwa opcji, w drugiej wskazany jest typ nagłówka, w ramach którego opcja może wystąpić: *Skok po skoku* (H) lub *Opcje docelowe* (D). Trzy kolejne kolumny odzwierciedlają wartości podpól *Akcja*, *ZMN* i *Rodzaj* bajta identyfikacyjnego, kolejna kolumna odzwierciedla wartość znajdującą się w drugim bajcie, wskazującym długość danych opcji. Opcja Pad1 jest wyjątkowa w tym sensie, że w jej przypadku nie występuje ów drugi bajt — opcja nie zawiera żadnych danych, jedyną przekazywaną treścią jest sam fakt jej wystąpienia, sygnalizowany przez bajt identyfikacyjny o wartości 0.

Tabela 5.7. *Opcje IPv6 w podziale na dwie grupy Skok po skoku (H) oraz Opcje docelowe (D). Kolumny Akcja, ZMN i Rodzaj ukazują wartość poszczególnych podpól bajta identyfikacyjnego, w kolumnie Długość znajduje się wartość drugiego bajta, określającego długość danych (w opcji Pad1 bajt ten nie występuje)*

Nazwa opcji	Grupa	Akcja	ZMN	Rodzaj	Długość	Dokument
Pad1	HD	00	0	0	Nie dotyczy	[RFC2460]
PadN	HD	00	0	1	Zmienna	[RFC2460]
Jumbo Payload	H	11	0	194	4	[RFC2675]
Tunnel Encapsulation Limit	D	00	0	4	4	[RFC2473]
Router Alert	H	00	0	5	4	[RFC2711]
Quick-Start	H	00	1	6	8	[RFC4782]
CALIPSO	H	00	0	7	8 lub więcej	[RFC5570]
Home Address	D	11	0	201	16	[RFC6275]

5.3.1.1. Opcje Pad1 i PadN

Wymaga się, aby każda z opcji IPv6 rozpoczynała się na offsecie stanowiącym wielokrotność 8 bajtów (w stosunku do początku datagramu[4]), więc w przypadku opcji o długości niestanowiącej tej wielokrotności konieczne jest wypełnianie przestrzeni między nimi w sposób zrozumiały dla hosta przetwarzającego nagłówek rozszerzenia. Gdy przestrzeń ta ma rozmiar jednego bajta, wypełniana jest pojedynczym bajtem zerowym, co w kategoriach tabeli 5.7 oznacza wystąpienie opcji `Pad1`. Gdy przestrzeń ta jest większa, do wypełniania używana jest opcja `PadN`, rozpoczynająca się bajtem identyfikacyjnym o wartości `1`, po którym następuje bajt długości, zliczający liczbę zer wypełniających pozostałe bajty. Tak więc np. pięciobajtowy ciąg wypełniający składa się w bajtów o wartościach `1, 3, 0, 0, 0`, a wypełnienie przestrzeni dwubajtowej może mieć postać `0, 0` (dwie opcje `Pad1`) lub `1, 0` (opcja `PadN`).

5.3.1.2. Opcja Jumbo Payload

W niektórych sieciach TCP/IP, m.in. w sieciach łączących superkomputery, używanie datagramów o rozmiarze 64 kB oznaczałoby zbyt duży narzut sieciowy w przypadku przesyłania masywnych porcji danych. Rozwiązaniem tego problemu jest użycie datagramów o większych rozmiarach (do 4 GB), zwanych jumbogramami. Gdy obecna jest opcja *Jumbo Payload*, rozmiar ładunku użytecznego datagramu odczytywany jest z 32-bitowego pola danych tej opcji, a nie jak zwykle z pola *Rozmiar ładunku użytecznego* w nagłówku podstawowym IPv6 — dla jumbogramu pole to powinno mieć wartość 0.

Opcja *Jumbo Payload* może nie być implementowana przez węzły operujące na bazie łączy, których MTU nie przekracza wartości 64 kB; napotkanie jej w takiej sytuacji powoduje odrzucenie datagramu z ostrzeżeniem (vide wartość 11 w podpolu *Akcja*). Jej wystąpienie jest także nieobojętne dla protokołu TCP. Jak pokażemy w dalszym ciągu książki, integralność jego pakietów kontrolowana jest za pomocą sumy kontrolnej, uwzględniającej również rozmiar ładunku użytecznego w datagramie enkapsulującym segment TCP, więc w tym celu konieczne jest odwołanie się do właściwego pola (tego w opcji, nie tego w nagłówku podstawowym). Należy ponadto pamiętać, że większy rozmiar pakietu oznacza większe prawdopodobieństwo wystąpienia *niewykrytego* błędu (patrz [RFC2675]).

5.3.1.3. Opcja Tunnel Encapsulation Limit

Jak wyjaśnialiśmy w rozdziale 3., **tunelowanie** to enkapsulowanie pakietów jednego protokołu w pakietach innego, z naruszeniem hierarchicznej zależności warstw wynikającej z modelu odniesienia TCP/IP; przykładowo datagramy IP mogą być enkapsulowane w innych pakietach IP. Tunelowanie wykorzystywane jest do konstrukcji wirtualnych sieci nakładkowanych, w ramach których jedna z sieci (np. Internet) pełni rolę niezawodnej warstwy łącza danych dla innej warstwy IP (patrz [TWEF03]). Nic nie stoi na przeszkodzie tunelowaniu *wielopoziomowemu* — czyli sytuacji, w której pakiet enkapsulujący staje się również pakietem enkapsulowanym.

[4] Dokładniej mówiąc: w stosunku do początku nagłówka rozszerzenia. Ponieważ jednak nagłówek podstawowy ma rozmiar 40 bajtów, czyli wielokrotność 8 bajtów, oba stwierdzenia są równoważne — *przyp. tłum.*

Generalnie host lub router wysyłające pakiet IP nie mają możliwości kontroli nad liczbą poziomów tunelowania zagnieżdżonych w tym pakiecie; opisywana opcja daje im możliwość określenia limitu tych poziomów. Router zamierzający wykonać enkapsulację pakietu IPv6 w tunelu sprawdza najpierw, czy limit ów został zdefiniowany: jeśli wynosi zero, pakiet jest odrzucany, a do nadawcy (czyli punktu wejścia do aktualnego tunelu) wysyłane jest ostrzeżenie ICMPv6 (patrz rozdział 8.); jeżeli jest niezerowy, tunelowanie jest wykonywane, a w nowo utworzonym datagramie IPv6 wartość tego limitu (w polu danych opcji) zmniejszana jest o 1. W rezultacie wspomniany limit zachowuje się podobnie jak pola *Czas życia* (w nagłówku IPv4) i *Limit przeskoków* (w nagłówku IPv6), tyle że w odniesieniu do innej własności datagramu. Oczywiście, nieobecność opcji *Tunnel Encapsulation Limit* oznacza brak ograniczenia poziomu zagnieżdżenia tunelu.

5.3.1.4. Opcja Router Alert

Wystąpienie tej opcji oznacza, że datagram zawiera informacje wymagające przetwarzania przez router — podobnie jak w przypadku datagramu IPv4. Szczegółowy powód alarmowania routera zakodowany jest w 4-bajtowym polu danych opcji; znaczenie poszczególnych kodów opisane zostało na stronie [RTAOPTS].

5.3.1.5. Opcja Quick-Start

Opcja ta (w skrócie QS) wykorzystywana jest w połączeniu z eksperymentalną procedurą tzw. szybkiego startu (*Quick Start Procedure*) opisywaną w dokumencie [RFC4782]. Obsługiwana jest zarówno w ramach IPv4, jak i IPv6, jednak zaleca się ograniczenie jej stosowania do sieci prywatnych, poza kontekstem globalnego Internetu. Dane opcji zawierają zakodowaną szybkość transmisji wymaganą przez nadawcę, wartość QS TTL i inne informacje pomocnicze. Każdy router na trasie pakietu, który implementuje obsługę opisywanej opcji, zmniejsza o 1 wartość pola QS TTL i ewentualnie koryguje (zmniejsza) podaną wartość transmisji; routery nieimplementujące opcji zwyczajnie ją ignorują (vide wartość 00 w podpolu *Akcja*). Host docelowy po odebraniu datagramu wysyła do nadawcy komunikat zawierający m.in. różnicę między wartością w polu *Limit przeskoków* (*Czas życia* w IPv4) a QS TTL oraz skorygowaną ostatecznie wartość szybkości transmisji. Jeśli wspomniana różnica jest niezerowa, oznacza to istnienie na trasie datagramu routerów, które nie implementują opcji QS, w przeciwnym razie skorygowana wartość transmisji przyjmowana jest przez nadawcę jako obowiązująca na całej trasie (może być ona wyższa od tej, którą przyjąłby protokół TCP bez wykonywania procedury *Quick Start*).

5.3.1.6. Opcja CALIPSO

CALIPSO to skrót od *Common Architecture Label IPv6 Security Option* — „opcja wspólnej architektury oznaczania [datagramów] IPv6 etykietami bezpieczeństwa". Opcja ta, opisana w dokumencie [RFC5570], wykorzystywana jest w niektórych sieciach prywatnych do etykietowania datagramów IP wskaźnikami poziomu bezpieczeństwa, wraz z informacjami towarzyszącymi. Przeznaczona jest głównie do użytku w sieciach z wielopoziomową strukturą zabezpieczeń, wykorzystywanych w krytycznych zastosowaniach bankowych, militarnych i rządowych, gdzie poziom bezpieczeństwa danych musi być jawnie wskazany za pomocą specjalnej etykiety.

5.3.1.7. Opcja Home Address

Opcja ta związana jest z mechanizmem „mobilnego IP" (o którym piszemy w podrozdziale 5.5) obejmującym zestaw procedur umożliwiających dynamiczną zmianę adresów przyporządkowywanych urządzeniom mobilnym, bez przerywania trwających połączeń w warstwach wyższych. Jedną z koncepcji mobilnego IP jest „adres domowy" (*home address*) urządzenia, czyli adres IP stanowiący pochodną prefiksu sieciowego typowej lokalizacji tego urządzenia; gdy urządzenie się przemieszcza, może ono dynamicznie otrzymywać inne, tymczasowe adresy IP. Adres domowy urządzenia, utrzymywany w datagramach jako wartość opisywanej opcji, wykorzystywany jest w kontekście tego urządzenia jako swoista identyfikacja z innymi urządzeniami mobilnymi.

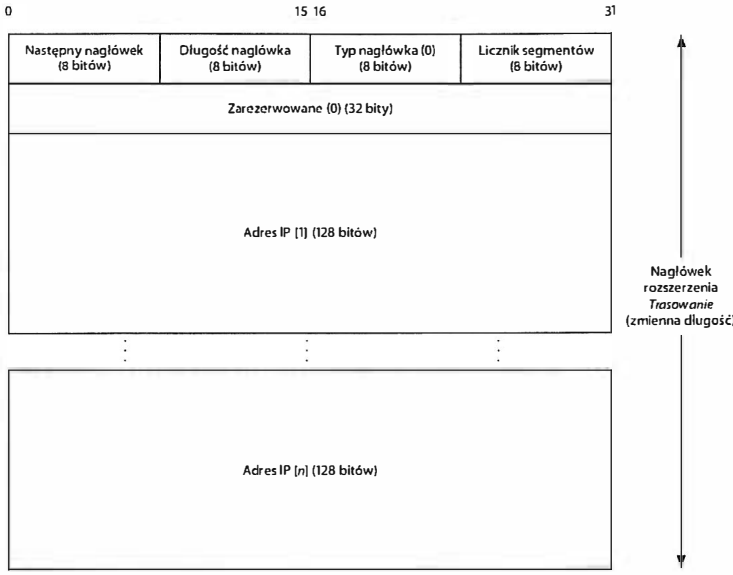
Jeśli opcja *Home address* jest używana, zawierający ją nagłówek rozszerzenia *Opcje docelowe* musi pojawić się po nagłówku *Trasowanie*, a przed nagłówkami *Fragmentacja*, *Uwierzytelnianie* i *Nagłówek ESP* (patrz rozdział 18.), o ile — oczywiście — wspomniane nagłówki w ogóle występują. Powrócimy do tej kwestii przy okazji szczegółowego omawiania mobilnego IP.

5.3.2. Nagłówek trasowania

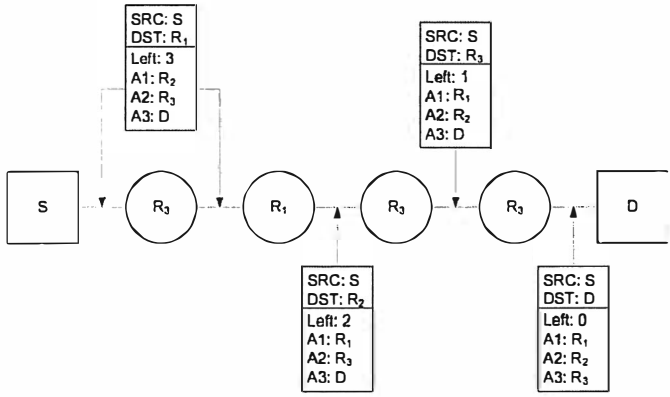
Nagłówek *Trasowanie* umożliwia nadawcy datagramu określenie listy (przynajmniej częściowej) adresów IP routerów, przez które datagram ten ma przejść na drodze do celu. Pierwotna wersja tego nagłówka, oznaczana umownie RH0, definiowana w sekcji 4.4 dokumentu [RFC2460], została na mocy dokumentu [RFC5095] wycofana z użycia jako wątpliwa pod względem bezpieczeństwa i ustąpiła miejsca wersji zwanej RH2, zdefiniowanej w związku z mobilnym IP. Rozpoczniemy od omówienia wersji RH0, wyjaśnimy przyczyny jej zarzucenia i pokażemy, czym różni się od niej wersja RH2. Format nagłówka RH0 widoczny jest na rysunku 5.8.

Centralną częścią nagłówka RH0 jest lista pozycji reprezentujących węzły, które odwiedzić musi datagram na swej trasie do celu. Standardowo pozycje te są adresami IP typu unicast rzeczonych węzłów; teoretycznie możliwe jest identyfikowanie węzłów w inny sposób, jest to jednak możliwość nieudokumentowana w standardzie IPv6 i nie będziemy jej rozważać. W polu *Typ nagłówka* znajduje się wartość 0 jako identyfikator wersji RH0 (wersja RH2 identyfikowana jest przez wartość 2). Wartość pola *Licznik segmentów* równa jest liczbie nieodwiedzonych jeszcze pozycji z listy. Liczba właściwych pozycji poprzedzona jest polem 32 zerowych bitów, ignorowanym przez routery.

W czasie wędrówki datagramu nagłówek *Trasowanie* nie jest przetwarzany do momentu osiągnięcia węzła, którego adres równy jest zawartości pola *Docelowy adres IP* w nagłówku podstawowym. Wówczas pole *Licznik segmentów* wykorzystywane jest do określenia pozycji (na liście adresów IP) zawierającej następny żądany adres na trasie. Następuje zamiana miejscami obu adresów oraz zmniejszenie o 1 pola *Licznik segmentów*. Przykład przedstawiony na rysunku 5.9 pozwoli lepiej zrozumieć ten mechanizm.



Rysunek 5.8. *Zdeprecjonowana wersja RH0 nagłówka Trasowanie jest uogólnieniem opcji Trasowanie źródłowe i Rejestracja trasy. Nadawca datagramu umieszcza w nagłówku listę adresów węzłów, które odwiedzić musi datagram na swej trasie do węzła docelowego. Lista może mieć charakter zamknięty (tzw. ścisłe trasowanie) lub elastyczny (tzw. luźne trasowanie) — w tym drugim przypadku trasa między sąsiednimi pozycjami nie musi być pokonana w postaci pojedynczego przeskoku. Pole Docelowy adres IP w nagłówku podstawowym modyfikowane jest przy każdym przeskoku tak, by wskazywać następny węzeł na trasie*



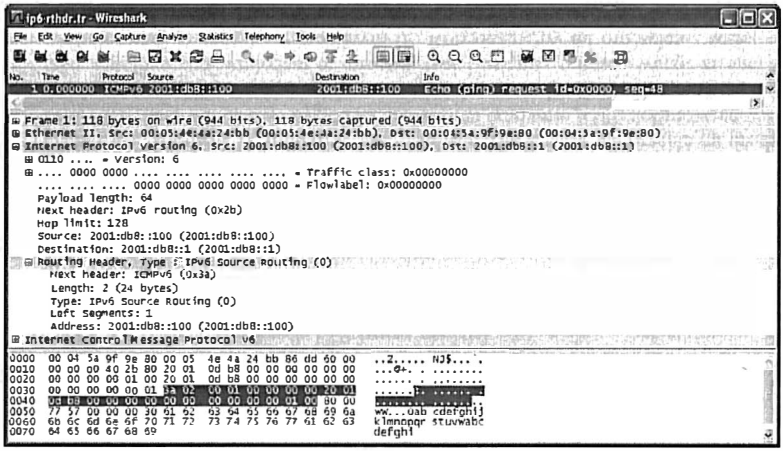
Rysunek 5.9. *Przykład użycia nagłówka trasowania w wersji RH0. Węzeł S wysyła datagram IPv6 do węzła D, wytyczając trasę prowadzącą przez węzły pośrednie R1, R2 i R3. Zauważmy, że adresy tych węzłów pojawiają się kolejno w polu Docelowy adres IP nagłówka podstawowego*

Na rysunku tym widzimy węzeł nadawczy **S** oraz datagram z nagłówkiem *Trasowanie*, w którym określono ciąg adresów $\mathbf{R_1}$, $\mathbf{R_2}$, $\mathbf{R_3}$ i **D**. W pole adresu docelowego (*DST*) wpisany zostaje adres R1, w pole *Licznik segmentów* — wartość 3 i datagram rusza w drogę. Router $\mathbf{R_0}$ nie napotyka swego adresu w polu *DST*, więc nagłówek *Trasowanie* pozostaje nienaruszony. Datagram dociera następnie do routera $\mathbf{R_1}$, pole *DST* zamienia się zawartością z pierwszą pozycją na liście, *Licznik segmentów* zostaje zmniejszony do 2 i datagram wędruje do routera $\mathbf{R_2}$. Tu sytuacja się powtarza: pole *DST* i przedostatnia pozycja zamieniają się miejscami, licznik segmentów przyjmuje wartość 1. Po dotarciu datagramu do $\mathbf{R_3}$ następuje kolejne, ostatnie powtórzenie scenariusza. Pole *DST* odzyskuje pierwotną zawartość, licznik segmentów zostaje wyzerowany, datagram zmierza ku węzłowi docelowemu **D**.

Opisany scenariusz możemy zaobserwować za pomocą programu Wireshark uwidaczniającego efekt „pingowania" wywołany za pomocą polecenia `ping6` w Windows XP; w Windows Vista i Windows 7 dostępne jest tylko standardowe polecenie `ping`, zapewniające obsługę IPv6.

```
C:\> ping6 -r -s 2001:db8::100 2001:db8::1
```

Powyższe polecenie powoduje wysyłanie żądań `ping` z adresu źródłowego `2001:db8::100` do adresu docelowego `2001:db8::1`; parametr `-r` powoduje dołączenie nagłówka RH0. Wychodzący pakiet żądania widoczny jest w oknie programu Wireshark (patrz rysunek 5.10).



Rysunek 5.10. *Polecenie ping powoduje pojawienie się pakietu żądania Echo protokołu ICMPv6. W polu Następny nagłówek nagłówka podstawowego IPv6 znajduje się wartość 0x2b (dziesiętnie 43), co zgodnie z tabelą 5.5 identyfikuje nagłówek Trasowanie; licznik segmentów równy jest 1, lista adresów zawiera tylko jedną pozycję 2001:db8::100. Następnym w kolejności nagłówkiem jest ICMPv6 (wartość 0x3a, dziesiętnie 58)*

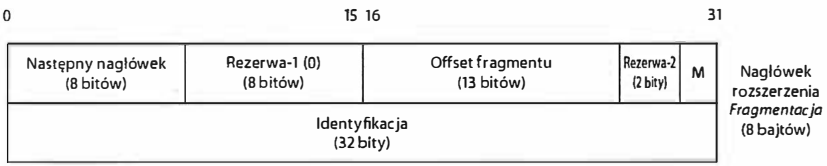
Komunikat `ping` ma postać pakietu żądania *Echo* protokołu ICMPv6 (patrz rozdział 8.). W datagramie IP komunikat ten poprzedzony jest nagłówkiem *Trasowanie* z wartością 0 w polu *Typ nagłówka*, mamy więc do czynienia z wersją RH0. Na liście adresów znajduje się tylko jedna pozycja — `2001:db8::100`.

Wspominaliśmy wcześniej, że wersja RH0 wycofana została z użytku jako niebezpieczna. Zauważmy otóż, że obecna w nagłówku lista adresów IP może być kształtowana w zasadzie dowolnie, m.in. nic nie stoi na przeszkodzie *wielokrotnemu użyciu tych samych adresów*. Aranżując odpowiednio owe powtórzenia, możemy doprowadzić do narastającego ruchu oscylacyjnego, co efektywnie równa się atakowi przeciążeniowemu (DoS). W wersji RH2 wspomniana lista adresów może zawierać *co najwyżej jedną pozycję*, a *Licznik segmentów* musi mieć wartość 0 — i jest to jedyna różnica w stosunku do wersji RH0, oprócz — oczywiście — wartości 2 w polu *Typ nagłówka*.

5.3.3. Nagłówek fragmentacji

Gdy rozmiar datagramu przekracza wartość PMTU dla ścieżki od nadawcy do hosta docelowego, datagram ten dzielony jest na **fragmenty**. O znaczeniu PMTU wspominaliśmy już w rozdziale 3., szczegółowo omawiamy ją w rozdziale 13., tu ograniczymy się tylko do przypomnienia, że protokół IPv6 narzuca dolne ograniczenie 1280 bajtów na jej wartość (zgodnie z sekcją 5. dokumentu [RFC2460]). W ramach protokołu IPv4 fragmentacji datagramu dokonywać mógł każdy host lub router, jeśli datagram ten był zbyt duży w kontekście MTU łącza prowadzącego do następnego węzła; informacja związana z fragmentacją obecna była w drugim 32-bitowym słowie nagłówka IPv4, obecnego w każdym fragmencie. Protokół IPv6 ogranicza poniekąd tę swobodę — fragmentowanie datagramu wykonywać może wyłącznie jego nadawca, a informacja opisująca szczegóły fragmentacji umiejscowiona jest w ramach odpowiedniego nagłówka rozszerzenia.

Nagłówek rozszerzenia *Fragmentacja* zawiera te same informacje o fragmentacji, które znajdowały się w nagłówku IPv4, lecz pole *Identyfikacja* rozszerzone zostało do 32 bitów, co umożliwia współegzystencję w sieci większej liczby pofragmentowanych pakietów. Format nagłówka *Fragmentacja* przedstawiono na rysunku 5.11.



Rysunek 5.11. *Nagłówek rozszerzenia Fragmentacja. Pole identyfikacyjne jest dwukrotnie większe niż w nagłówku IPv4, offset fragmentu oznacza przesunięcie fragmentu, liczone w jednostkach 8-bajtowych, względem początku części fragmentowalnej oryginalnego pakietu. Bit M jest wyzerowany w ostatnim fragmencie pakietu i ustawiony na 1 w pozostałych fragmentach*

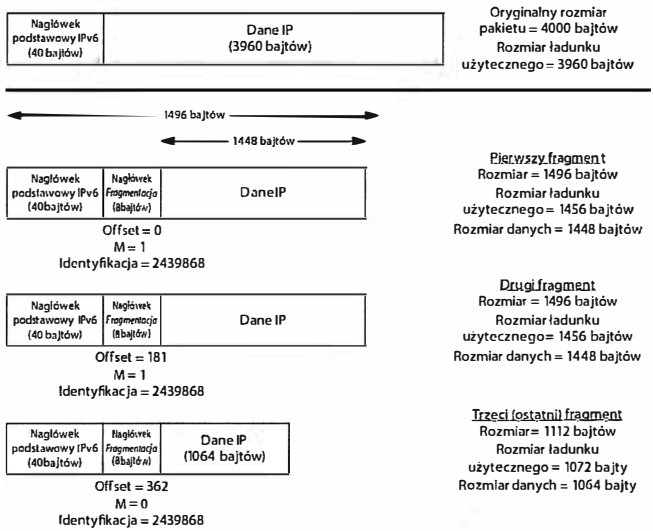
Dwa pola rezerwy zawierają zera i są ignorowane przez odbiorcę pakietu. *Offset fragmentu* określa umiejscowienie fragmentu w oryginalnym pakiecie, a konkretnie — przesunięcie tego fragmentu względem początku tzw. części fragmentowalnej, liczone w jednostkach 8-bajtowych. Zerowa wartość bitu *M* oznacza ostatni fragment pakietu.

W datagramie poddawanym fragmentacji (zwanym „oryginalnym datagramem") wyróżniamy dwie części: początkowa, *niefragmentowalna* część obejmuje nagłówek podstawowy oraz wszelkie nagłówki rozszerzeń podlegające przetwarzaniu przez węzły pośrednie (czyli wszystkie nagłówki do nagłówka *Trasowanie* włącznie albo nagłówek

opcji *Skok po skoku*, jeśli tylko on występuje w pakiecie). Pozostała, *fragmentowalna* część pakietu obejmuje nagłówek *Opcje docelowe*, nagłówki protokołów warstw wyższych i ładunek użyteczny.

Każdym fragment powstający z podziału datagramu oryginalnego rozpoczyna się kopią jego części niefragmentowalnej z tą różnicą, że w polu *Rozmiar ładunku użytecznego* znajduje się rozmiar fragmentu, nie rozmiar oryginalnego ładunku użytecznego[5]. Bezpośrednio za tą kopią znajduje się nagłówek *Fragmentacja*, z odpowiednio wypełnionym polem *Offset fragmentu* (równym zero w pierwszym fragmencie) i odpowiednio ustawionym bitem *M* (wyzerowanym w ostatnim fragmencie). Pole identyfikacyjne jest kopią pierwowzoru z części niefragmentowalnej.

Proces fragmentowania przykładowego datagramu przedstawiono poglądowo na rysunku 5.12. Datagram zawierający ładunek użyteczny o rozmiarze 3960 bajtów zostaje pofragmentowany w celu przetransmitowania przez ścieżkę, której PMTU wynosi 1500 (to wartość typowa dla Ethernetu). Rozmiar fragmentu nie może więc przekraczać 1500 bajtów, przy czym rozmiar danych fragmentu musi być wielokrotnością 8 bajtów. Odliczając 40 bajtów na nagłówek podstawowy i 8 bajtów na nagłówek *Fragmentacja* dostajemy limit 1452 bajtów na rozmiar danych, co po zaokrągleniu w dół do wielokrotności 8 daje 1448 bajtów. Ostatecznie maksymalny rozmiar pakietu równy jest 40 + 8 + 1448 = 1496 bajtów.



Rysunek 5.12. *Przykład podziału pakietu z ładunkiem użytecznym 3960 bajtów na trzy fragmenty o rozmiarze danych nie większym niż 1448 bajtów. We wszystkich fragmentach oprócz ostatniego bit M ma wartość 1. Offset fragmentu wyrażony jest w jednostkach 8-bajtowych, co dla drugiego i trzeciego fragmentu daje (odpowiednio) 181 · 8 = 1448 oraz 362 · 8 = 2896 bajtów. Schemat fragmentacji jest więc podobny do tego z IPv4, choć różni się od niego obecnością nagłówka rozszerzenia*

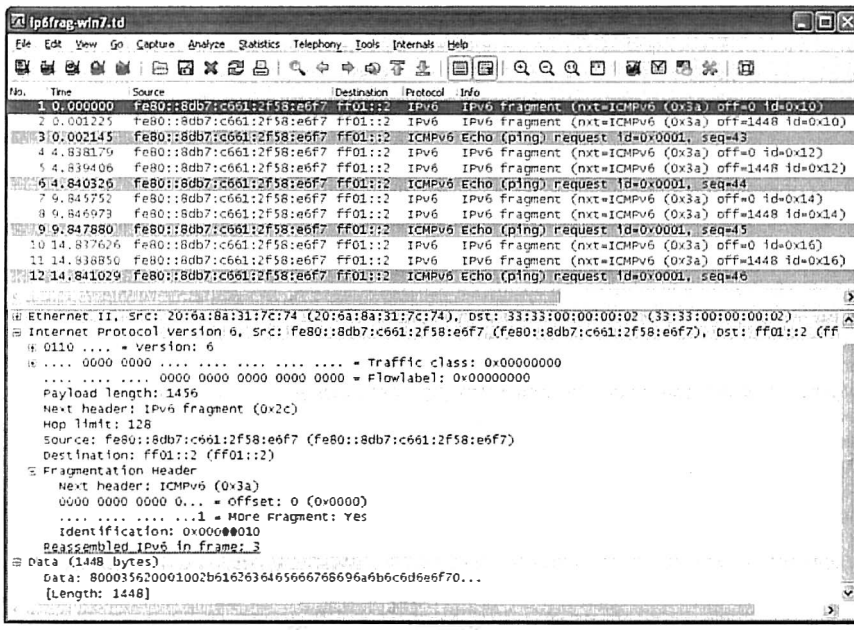
[5] Rozmiar ładunku użytecznego plus 8 bajtów nagłówka *Fragmentacja — przyp. tłum.*

Proces odwrotny do fragmentacji, czyli rekonstrukcja oryginalnego datagramu na podstawie fragmentów, nosi nazwę **defragmentacji** lub **reasemblacji**. Host docelowy musi — oczywiście — uprzednio otrzymać wszystkie fragmenty; podobnie jak w protokole IPv4, mogą one nadchodzić w dowolnej kolejności, niekoniecznie naturalnej, czyli według wzrastających offsetów (patrz rozdział 10.); do rozpoczęcia deasemblacji jest więc konieczne otrzymanie ostatniego fragmentu (rozpoznawanego na podstawie zerowej wartości bitu *M*) i wszystkich fragmentów poprzednich.

Fragmentację możemy zaobserwować w praktyce, inicjując wysyłanie datagramów IP za pomocą polecenia `ping` z odpowiednimi parametrami i podglądając transmitowane dane za pomocą programu Wireshark. W systemie Windows 7 wspomniane polecenie ma postać:

```
C:\> ping -l 3952 ff01::2
```

a jego konsekwencje wyglądają w oknie programu Wireshark podobnie do tych z rysunku 5.13.

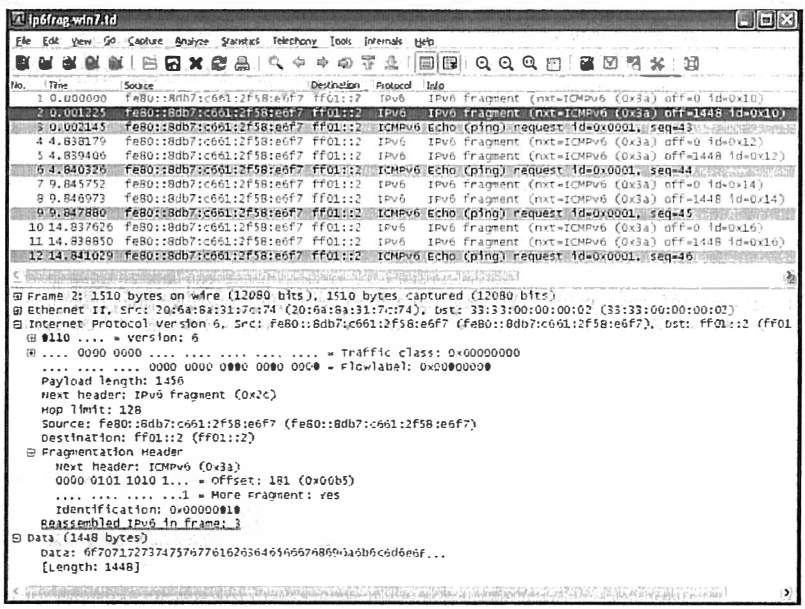


Rysunek 5.13. *Program ping generuje pakiety ICMPv6 (patrz rozdział 8.) zawierające 3960-bajtowy ładunek użyteczny IPv6. Pakiety te są fragmentowane w celu zdolności ich przetransmitowania przez łącze ethernetowe o wartości MTU 1500 bajtów*

Na rysunku widzimy pofragmentowany komunikat żądania Echo ICMPv6, wysyłany na adres docelowy multicast `ff01::2`. Fragmentacja jest konieczna, ponieważ parametr `-l` 3952 powoduje generowanie pakietów ICMPv6 niosących ładunek użyteczny o rozmiarze

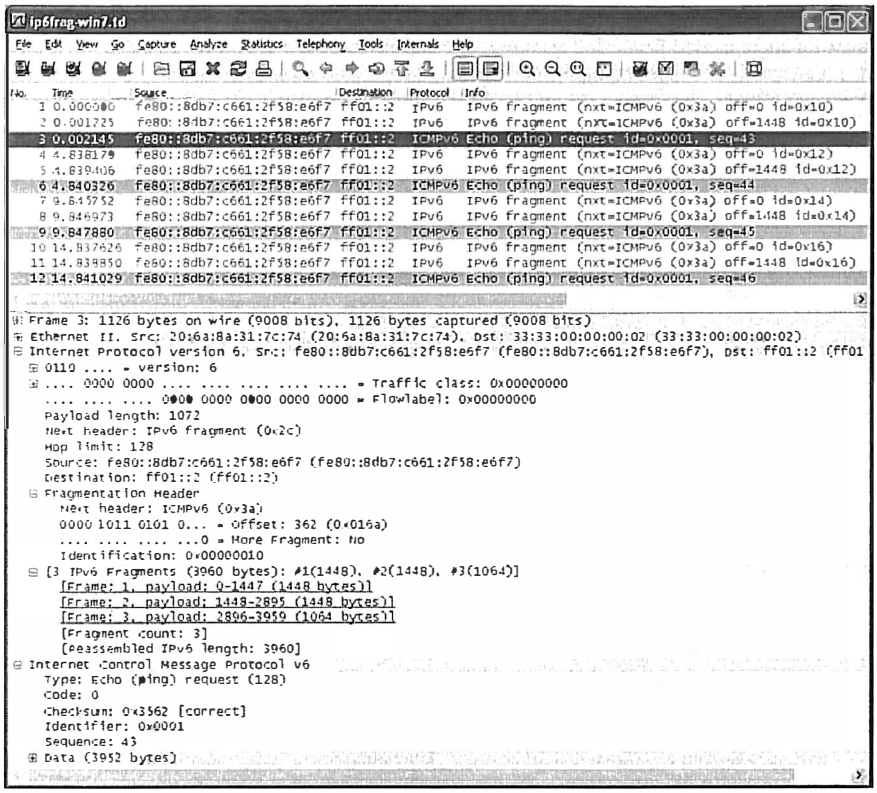
3952 bajtów. Razem z 8-bajtowym nagłówkiem ICMPv6 daje to pakiet ICMPv6 o wielkości 3960 bajtów. Pakiet ten staje się następnie ładunkiem użytecznym datagramu IPv6. Widoczny w nagłówku podstawowym IPv6 adres źródłowy jest adresem IPv6 lokalnym dla łącza. Dla określenia docelowego adresu multicast warstwy łącza danych wykorzystywana jest specyficzna dla IPv6 procedura mapująca, opisywana w rozdziale 9. Gdy wszystkie fragmenty danego pakietu dotrą do odbiornika, jakim jest program Wireshark, zostaną poskładane w całość, jaką jest oryginalny datagram IPv6.

Na rysunku 5.14 zobaczyć możemy szczegóły drugiego fragmentu. Zgodnie z oczekiwaniami, rozpoczyna się on podstawowym nagłówkiem IPv6, w którym wykazany jest ładunek użyteczny o rozmiarze 1448 bajtów, zaś pole *Następny nagłówek* zawiera kod 44 (0x2c) identyfikujący (zgodnie z tabelą 5.5) nagłówek rozszerzenia *Fragmentacja*. Kolejnym nagłówkiem w łańcuchu jest nagłówek protokołu ICMPv6, nie ma więc dalszych nagłówków rozszerzeń. Wartość 181 w polu *Offset fragmentu* umiejscawia początek tego fragmentu na bajcie przesuniętym o $181 \cdot 8 = 1448$ bajtów w stosunku do początku części fragmentowalnej oryginalnego datagramu. Ustawienie na 1 bitu *M* (sygnalizowane przez Wireshark jako wartość *Yes* zatytułowana *More Fragment*) oznacza, że wyświetlany fragment nie jest ostatnim fragmentem oryginalnego datagramu.



Rysunek 5.14. *Zaznaczony drugi fragment datagramu IPv6, zawierający 1448 bajtów ładunku użytecznego, w tym 8-bajtowy nagłówek rozszerzenia Fragmentacja. Obecność tego nagłówka jest świadectwem pofragmentowania datagramu przez nadawcę, wartość 181 w polu offsetu oznacza natomiast, że niniejszy fragment odzwierciedla część oryginalnego datagramu rozpoczynającą się na przesunięciu $181 \cdot 8 = 1448$ bajtów względem początku części fragmentowalnej. Wartość 1 bitu M oznacza, niniejszy fragment nie jest ostatni. Wszystkie fragmenty tego samego pakietu mają identyczną wartość w polu Identyfikacja, w tym przypadku 2*

Analogiczne obserwacje poczynić możemy w odniesieniu do trzeciego fragmentu (patrz rysunek 5.15). Z pola *Offset fragmentu* odczytujemy 362 · 8 = 2896 jako przesunięcie tego fragmentu względem początku części fragmentowalnej. W polu *Rozmiar ładunku użytecznego* widnieje wartość 1072; jeśli odejmiemy od niej 8 jako rozmiar nagłówka *Fragmentacja*, otrzymamy 1 064 jako rozmiar „netto" tegoż ładunku. Na podstawie offsetów i rozmiarów poszczególnych fragmentów Wireshark odtwarza ich chronologiczną aranżację, w sposób czytelny dla użytkownika.



Rysunek 5.15. *Zaznaczony ostatni fragment datagramu IPv6, zawierający 1072 bajty ładunku użytecznego, w tym 8-bajtowy nagłówek rozszerzenia Fragmentacja. Wartość 0 bitu M kwalifikuje niniejszy fragment jako ostatni, ponadto sumując jego offset (362 · 8 = 2896) i rozmiar „netto" ładunku użytecznego (bez uwzględniania nagłówka Fragmentacja) równy 1064 otrzymamy wartość 2896 + 1064 = 3960 jako rozmiar oryginalnego ładunku użytecznego (złożonego z 8-bajtowego nagłówka pakietu ICMPv6 i 3956 bajtów danych tego pakietu)*

Na podstawie dotychczasowego opisu fragmentacji datagramu IPv6 i na podstawie analizy prezentowanego przykładu możemy ocenić narzut — w postaci dodatkowych danych wymagających transmitowania — jaki wprowadza fragmentacja do protokołu IPv6. Po

pierwsze, w każdym fragmencie znajduje się kopia podstawowego nagłówka IPv6, zamiast więc 40 bajtów jednego egzemplarza tegoż nagłówka mamy teraz trzy egzemplarze, co oznacza narzut w ilości (3–1)*40 = 80 bajtów. W każdym fragmencie obecny jest ponadto 8-bajtowy nagłówek rozszerzenia *Fragmentacja*, co oznacza narzut 3*8 = 24 dodatkowych bajtów. Ponadto, ponieważ do przeniesienia fragmentów potrzebne będą trzy ramki ethernetowe (zamiast jednej), musimy uwzględnić 18-bajtowy narzut (14-bajtowy nagłówek + 4-bajtowa suma FCS) na każdą z dwóch dodatkowych ramek. Łącznie zatem warstwa łącza danych otrzyma dodatkowo 80+24+2*18 = 140 bajtów do przetransferowania.

5.4. Forwardowanie datagramów IP

Forwardowanie datagramów IP jest pod względem koncepcyjnym mało skomplikowane, szczególnie z perspektywy hosta. Jeśli węzeł docelowy przyłączony jest do hosta bezpośrednio (np. za pomocą łącza punkt-punkt) lub za pośrednictwem współdzielonej sieci (np. Ethernetu), datagram przesyłany jest bezpośrednio, bez użycia routera. W przeciwnym razie host wysyła datagram do **routera domyślnego** (*default router*), powierzając temu routerowi zadanie dostarczenia datagramu do celu. Ten prosty schemat funkcjonuje w przeważającej większości hostów.

W tym podrozdziale przeanalizujemy szczegóły tej prostej operacji, by następnie przyjrzeć się operacji, która już tak prosta nie jest. Zauważmy na wstępie, że większość współczesnych komputerów ma dwojakie oblicze, każdy z nich funkcjonować może zarówno jak typowy host, jak i w charakterze routera — wiele z domowych sieci wykorzystuje komputery PC przyłączone do Internetu właśnie jako routery (a także jako firewalle, którym poświęcamy rozdział 7.). Zasadniczą cechą odróżniającą (w kontekście IP) host od routera jest traktowanie „obcych" datagramów: host, w przeciwieństwie do routera, nigdy nie zajmuje się forwardowaniem datagramów, których sam nie wytworzył. W naszym ogólnym schemacie protokół IP otrzymywać może datagramy od innych protokołów funkcjonujących w tym samym komputerze (m.in. TCP i UDP) bądź z interfejsu sieciowego.

Wykonywane przez warstwę IP forwardowanie datagramów wiąże się z utrzymywaniem w pamięci hosta lub routera pomocniczych danych, zwanych popularnie **tablicą trasowania** (*routing table*) lub **tablicą forwardowania** (*forwarding table*). Gdy datagram zostaje odebrany przez interfejs sieciowy, protokół IP sprawdza najpierw, czy zawarty w tym datagramie docelowy adres IP jest adresem własnym węzła, w którym protokół ten funkcjonuje (czyli jednym z adresów przypisanych interfejsom sieciowym tegoż węzła), bądź „pasującym" do węzła adresem multicast lub broadcast. Jeśli tak, datagram dostarczany jest do modułu implementującego protokół wskazywany w polu *Protokół* (w IPv6 w polu *Następny nagłówek*) w nagłówku tegoż datagramu. W przeciwnym razie datagram potraktowany zostać może w dwojaki sposób:

- jeśli warstwa IP skonfigurowana jest do pełnienia funkcji routera, datagram jest forwardowany, czyli przetwarzany według scenariusza opisywanego w punkcie 5.4.2,
- jeśli warstwa IP nie implementuje funkcji routera, datagram jest odrzucany; w niektórych sytuacjach do nadawcy wysyłany jest komunikat protokołu ICMP z informacją o wystąpieniu błędu.

5.4.1. Tablica forwardowania

Specyfikacja protokołu IP nie precyzuje szczegółów danych składających się na tablicę forwardowania, pozostawiając w tym zakresie swobodę autorom konkretnych implementacji. Reguły protokołu IP nieuchronnie prowadzą jednak do struktury, która (przynajmniej koncepcyjnie) powinna być wektorem pozycji składających się z czterech następujących pól. Oto one.

- **Przeznaczenie**, czyli 32-bitowa (dla IPv4) lub 128-bitowa (dla IPv6) wartość spełniająca wraz z polem maski (patrz następna pozycja) funkcje *dopasowywania* adresu docelowego (szczegóły już za chwilę). Może to być np. wartość zerowa, odpowiadająca „trasie domyślnej" reprezentującej wszystkie możliwe miejsca przeznaczenia, albo pełny adres IP w przypadku „trasy hosta" reprezentującej pojedyncze miejsce przeznaczenia.
- **Maska**, czyli ciąg bitów o rozmiarze takim samym jak pole **Przeznaczenie**. Zostaje pomnożona bitowo (AND) z docelowym adresem IP zawartym w datagramie, po czym następuje próba dopasowania wyniku tego mnożenia do któregoś z pól **Przeznaczenie** w tablicy forwardowania.
- **Adres następnego przeskoku**, czyli adres IP następnego węzła (routera lub hosta), do którego przekazany zostanie datagram.
- **Identyfikator interfejsu** wykorzystywany przez warstwę IP do wyboru interfejsu, który powinien dostarczyć datagram do następnego przeskoku; może to być np. karta ethernetowa (tradycyjna lub bezprzewodowa) albo interfejs PPP skojarzony z portem szeregowym. Gdy węzeł wysyłający datagram jest jednocześnie jego twórcą, pole to może posłużyć do określenia, który z adresów IP powinien zostać wpisany do nagłówka datagramu jako adres źródłowy (do tej kwestii powrócimy w podpunkcie 5.6.2.1).

Forwardowanie pakietów IP odbywa się na zasadzie „skok po skoku": jak można się zorientować z dotychczasowego opisu, host ani router nie zawierają informacji o kompletnej trasie datagramu, aż do węzła docelowego, lecz jedynie informację konieczną do właściwego wyboru następnego węzła na tej trasie (wyjątkiem jest — oczywiście — sytuacja, gdy następny węzeł jest jednocześnie docelowym, bezpośrednio połączonym z bieżącym). Węzeł „właściwie wybrany" to węzeł z założenia znajdujący się „bliżej" węzła docelowego na wspomnianej trasie i połączony z węzłem bieżącym w sposób umożliwiający bezpośrednie przekazanie datagramu. Od trasy datagramu wymaga się natomiast, by była pozbawiona cykli, w przeciwnym razie datagram, zamiast dotrzeć do celu, mógłby krążyć po sieci, aż do wyczerpania się limitu ustalonego przez pole *Czas życia* (w IPv6 pole *Limit przeskoków*). Zapewnienie poprawnej postaci tablic forwardowania (m.in. we wspomnianych aspektach) jest zadaniem **protokołów trasowania** (*routing protocols*) — RIP, OSPF, BGP czy IS-IS, że ograniczymy się do wymienienia tylko kilku najbardziej znanych (więcej informacji na temat protokołów tej kategorii znajdą czytelnicy np. w publikacji [DC05]).

5.4.2. Szczegóły forwardowania

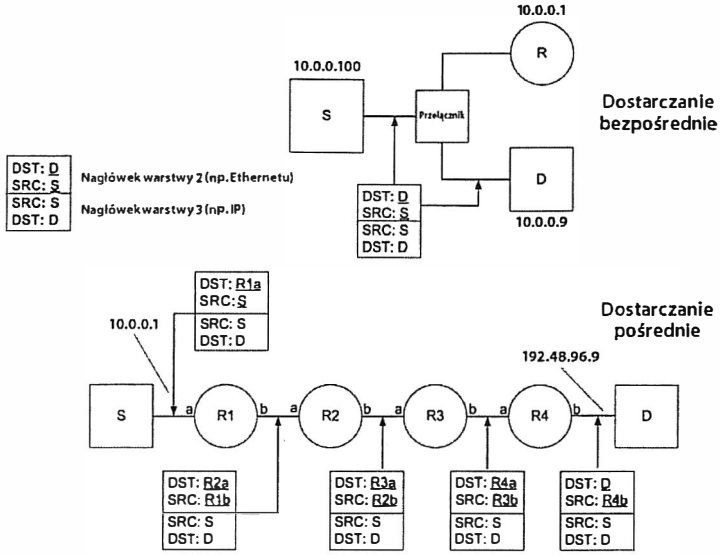
Gdy warstwa IP implementowana przez host lub router zamierza wysłać datagram IP do następnego przeskoku, rozpoczyna od konfrontacji adresu docelowego tegoż datagramu (oznaczmy ten adres przez D) z kolejnymi pozycjami swej tablicy forwardowania. Pozycję o numerze j we wspomnianej tablicy uznaje się za *pasującą* do adresu D, jeśli koniunkcja bitowa (AND) adresu D i pola *Maska* (m_j) na tej pozycji da wynik równy zawartości pola *Przeznaczenie* (d_j), czyli gdy $D \wedge m_j = d_j$. Spośród wszystkich pozycji pasujących do adresu D wybrana zostaje ostatecznie ta, której pole m_j zawiera najwięcej bitów jedynkowych. Powyższy algorytm nosi nazwę *dopasowania do najdłuższego prefiksu* (*longest prefix match*), bo maski w tablicy forwardowania są maskami podsieci, a maska podsieci stanowi (jak wyjaśnialiśmy w rozdziale 2.) konkatenację ciągu (być może pustego) bitów jedynkowych z ciągiem (być może pustym) bitów zerowych. Wyjaśnia to związek z długością prefiksu i liczbą „jedynek" w masce.

Jeżeli w tablicy forwardowania nie zostanie odnaleziona przynajmniej jedna pozycja pasująca do adresu D, datagram uznawany jest za *niedostarczalny* (*undeliverable*). Jeśli sytuacja taka ma charakter lokalny — niemożliwość forwardowania datagramu stwierdzona została przez host, który datagram ów wygenerował — odnośna aplikacja otrzymuje standardowy komunikat w rodzaju „Host nieosiągalny". Gdy opisana sytuacja wydarzy się w obrębie routera, zostaje wysłany odpowiedni komunikat ICMP do hosta będącego nadawcą datagramu.

Powróćmy jeszcze do dopasowywania adresu D do pozycji tablicy forwardowania: może się zdarzyć, że na kilku pozycjach stwierdzona zostanie największa liczba bitów jedynkowych w masce i żadna z nich nie zasługuje tym samym na miano „najlepszej". Może się tak zdarzyć, jeśli dla hosta określono kilka domyślnych routerów, bo w sieci zastosowano multihoming, czyli przyłączenie do kilku dostawców Internetu. Standard IP nie określa żadnej szczególnej zasady rozstrzygania takich remisów, najczęściej więc implementacje wybierają pierwszą z kandydatur. Bardziej inteligentna strategia wyboru mogłaby realizować podział ruchu między kilka tras, być może z uwzględnieniem ich oczekiwanego obciążenia. Jak dowodzą analizy przedstawione w publikacji [THL06], multihoming może być korzystny nie tylko dla dużych przedsiębiorstw, lecz także dla użytkowników domowych.

5.4.3. Przykłady

Zilustrujmy konkretnymi przykładami opisane reguły forwardowania, a szczególnie dopasowywanie do najdłuższego prefiksu w tablicach trasowania. Przeanalizujemy dwa przypadki dostarczania datagramu: lokalne — gdzie host wysyłający i docelowy znajdują się w tej samej sieci (tzw. **dostarczanie bezpośrednie** — *direct delivery*) — oraz zdalne, angażujące serię routerów (zwane **dostarczaniem pośrednim** — *indirect delivery*). Oba schematy zilustrowano na rysunku 5.16.



Rysunek 5.16. *Dostarczanie bezpośrednie nie wymaga użycia routera — datagram IP enkapsulowany jest w ramce warstwy łącza danych, do której wprost wpisywane są adresy sprzętowe źródłowy i docelowy. Dostarczanie pośrednie wiąże się z wykorzystaniem routera — dane forwardowane są do routera przy użyciu jego adresu sprzętowego jako docelowego w ramce. Adres IP routera nie pojawia się w datagramie IP (chyba że router jest węzłem nadawcą lub węzłem docelowym albo użyta została opcja Trasowanie źródłowe*

5.4.3.1. Dostarczanie bezpośrednie

Rozpoczniemy od przykładu dostarczania bezpośredniego, zilustrowanego w górnej części rysunku 5.16. Nasz host **S**, zarządzany przez system Windows XP, posiada adres IPv4 10.0.0.100 oznaczony tu *S*, a jego karta sieciowa posiada adres MAC, który oznaczymy *S* (podkreślone); host ten wysyła właśnie datagram IP do hosta linuksowego **D** o adresie IP równym 10.0.0.9, który oznaczamy przez *D*, i adresie MAC, który oznaczamy przez *D* (podkreślone). Oba hosty połączone są za pomocą przełącznika. Tablica trasowania utrzymywana przez warstwę IP hosta **S** ma postać przedstawioną w tabeli 5.8.

Tabela 5.8. *Tablica trasowania w hoście S zawiera dwie pozycje. Hostowi S przydzielony został adres 10.0.0.100/25. Adresy docelowe z przedziału od 10.0.0.1 do 10.0.0.126 dopasowywane są do drugiej pozycji (jako zawierającej dłuższy prefiks) i stanowią podstawę dostarczania bezpośredniego datagramów. Pozostałe adresy docelowe pasują wyłącznie do pierwszej pozycji, prowadzącej do routera R o adresie 10.0.0.1*

Przeznaczenie (di)	Maska (mi)	Następny przeskok	Interfejs
0.0.0.0	0.0.0.0	10.0.0.1	10.0.0.100
10.0.0.0	255.255.255.128	10.0.0.100	10.0.0.100

Dopasowywanie adresu *D* równego 10.0.0.9 daje wynik pozytywny dla obu pozycji tablicy; w pierwszej masce liczba bitów jedynkowych wynosi 0, w drugiej 25, zdecydowanie więc wybrana zostaje druga pozycja, w której adres następnego przeskoku 10.0.0.100 równy jest adresowi hosta **S**; jest to jednocześnie sygnał dla hosta **S**, że datagram powinien zostać dostarczony w sposób bezpośredni, bez używania jakichkolwiek routerów.

Datagram zostaje więc poddany przez host **S** enkapsulacji w ramce warstwy łącza danych, przeznaczonej dla hosta **D**. Jeśli adres *D* jest dla hosta **S** nieznany, musi zostać rozpoznany przez odpowiedni protokół — w wersji IPv4 jest to protokół ARP, opisywany w rozdziale 4., w wersji IPv6 protokół ICMPv6, a dokładniej jego komunikaty *Neighbor Solicitation,* którymi zajmiemy się w rozdziale 8. Adres *D* wpisany zostaje następnie w pole *DST* wspomnianej ramki — w procesie jej dostarczania przełącznik wykorzystuje wyłącznie adresy warstwy łącza danych, abstrahując zupełnie od adresów IP w enkapsulowanym datagramie.

5.4.3.2. Dostarczanie pośrednie

Przeanalizujmy teraz przypadek przedstawiony w dolnej części rysunku 5.16. Z naszego hosta wysyłany jest datagram IP na adres 192.48.96.9, który na swej trasie napotka cztery routery pośredniczące. Jedynie pierwsza pozycja w tablicy trasowania (ta z zerową maską) pasuje do wspomnianego adresu docelowego, jako następny przeskok wykazywany jest tu adres 10.0.0.1, przyporządkowany „stronie a" routera R1. Sytuacja taka jest typowa dla większości sieci domowych.

Jak pamiętamy, w przypadku dostarczania bezpośredniego adresy IP w nagłówku datagramu — źródłowy i docelowy — odpowiadają adresom (odpowiednio) węzła nadawcy i węzła docelowego i podobnie jest z adresami sprzętowymi (np. ethernetowymi). W przypadku dostarczania pośredniego tak samo ma się rzecz z adresami IP, lecz już nie z adresami sprzętowymi: adresy źródłowy i docelowy w ramce odnoszą się do węzłów wyznaczających kolejny odcinek jej trasy; docelowy adres ethernetowy w ramce wychodzącej z hosta S wskazuje interfejs sieciowy po „stronie a" routera **R1**, ten opatrzony adresem IP 10.0.0.1. „Przeliczeniem" adresu IP na adres sprzętowy zajmuje się jeden z wymienionych wcześniej protokołów (ARP albo ICMPv6) działających w sieci łączącej host S z routerem **R1**. Gdy tylko host S otrzyma odpowiedź od routera **R1**, wyśle do niego ramkę enkapsulującą datagram IP; przetworzenie tej ramki odbędzie się wyłącznie na podstawie adresów warstwy łącza danych (a konkretnie — docelowego adresu ethernetowego).

Router **R1** po otrzymaniu wspomnianej ramki wyłuskuje z niej rzeczony datagram IP i przystępuje do przeglądania swej tablicy trasowania w celu dopasowania adresu docelowego 192.48.96.9 odczytanego z nagłówka datagramu. Zakładamy, że tablica trasowania zawiera dwie pozycje, jak w tabeli 5.9.

Router **R1** po stwierdzeniu, że adres docelowy datagramu (192.48.96.9) nie jest jego własnym adresem IP, przystępuje do forwardowania otrzymanego datagramu. Pole *Następny przeskok* dopasowanej pozycji wskazuje na adres 70.231.159.254, przypisany do „strony a" routera **R2** — jest to adres sieci o (cokolwiek skomplikowanej) nazwie adsl-70-231-159-254.dsl.snfc21.sbcglobal.net. Ponieważ jest to globalny adres internetowy, a zawarty w datagramie adres źródłowy IP jest lokalnym adresem jego własnej

Tabela 5.9. *Tabela forwardowania w routerze R1 wskazuje na potrzebę wykonania translacji adresów (NAT). W wyniku tej translacji prywatny, lokalny adres routera 10.0.0.1 odwzorowany zostaje na globalny adres 70.231.132.85, w wyniku czego datagramy generowane w sieci 10.0.0.0/25 widoczne są w Internecie jako wysyłane z adresu 70.231.132.85*

Przeznaczenie (di)	Maska (mi)	Następny przeskok	Interfejs	Komentarz
0.0.0.0	0.0.0.0	70.231.159.254	70.231.132.85	NAT
10.0.0.0	255.255.255.128	10.0.0.100	10.0.0.1	NAT

sieci, router **R1** musi zastąpić go adresem globalnym, wykonując procedurę tłumaczenia adresów sieciowych (*Network Address Translation*, w skrócie NAT) opisywaną szczegółowo w rozdziale 7. W wyniku wykonania tej procedury adres lokalny 10.0.0.100 zastąpiony zostaje przez globalny adres 70.231.132.85, reprezentujący odtąd „stronę b" routera R1.

Datagram IP po dotarciu do routera **R2** poddawany jest przetwarzaniu podobnemu do tego z **R1**, z wyjątkiem procedury NAT, która jest zbędna, jako że oba adresy IP w nagłówku datagramu są adresami globalnymi. Adres docelowy w nagłówku datagramu nie jest adresem własnym routera **R2**, więc datagram jest forwardowany; tym razem jednak router dysponuje większą liczbą możliwych tras (zamiast jednej trasy domyślnej), zależnie od sposobu połączenia routera z Internetem i lokalnych założeń polityki bezpieczeństwa.

W protokole IPv6 forwardowanie datagramów odbywa się podobnie, z dwiema istotnymi różnicami. Po pierwsze, przeliczanie adresów IP na adresy fizyczne jest zadaniem komunikatów *Neighbor Solicitation*, które (jako część protokołu ICMPv6) opiszemy w rozdziale 8. Po drugie, protokół IPv6 definiuje nową kategorię adresów — adresy lokalne dla łącza, rozpoczynające się od prefiksu f380::/10 (patrz rozdział 2.); fakt ten może wymagać interwencji użytkownika (administratora) w przypadku hostów *multihomed*, w celu wskazania konkretnego interfejsu, jaki ma być używany do wysyłania datagramów kierowanych na adresy tej grupy.

Poniższy przykład ilustruje wykorzystywanie adresów lokalnych dla łącza w systemie Windows XP; zakładamy, że protokół IPv6 został w tym systemie zainstalowany i działa prawidłowo:

```
C:> ping6 fe80::204:5aff:fe9f:9e80

Badanie fe80::204:5aff:fe9f:9e80 z użyciem 32 bajtów danych:

Brak trasy do miejsca docelowego.
  Określ poprawny identyfikator zakresu lub użyj parametru -s do określenia adre
su źródłowego.

...

Statystyka badania dla fe80::204:5aff:fe9f:9e80:
    Pakiety: Wysłane = 4, Odebrane = 0, Utracone = 4 (100% utraconych),
```

Niepowodzenie wynika z konieczności wskazania konkretnego interfejsu, przeznaczonego do wysyłania datagramów produkowanych przez program ping6; wskazanie to

może mieć formę konkretnego adresu IP lub zakresu. Skorzystamy z drugiej możliwości, wskazując jawnie numer interfejsu jako przyrostek %6 dodany do zasadniczego adresu:

```
C:> ping6 fe80::204:5aff:fe9f:9e80%6

Badanie fe80::204:5aff:fe9f:9e80%2
z fe80::5efe:192.168.131.67%2 z użyciem 32 bajtów danych:

Odpowiedź z fe80::5efe:192.168.131.67%6: bajtów=32 czas=1 ms
Odpowiedź z fe80::5efe:192.168.131.67%6: bajtów=32 czas=1 ms
Odpowiedź z fe80::5efe:192.168.131.67%6: bajtów=32 czas=1 ms
Odpowiedź z fe80::5efe:192.168.131.67%6: bajtów=32 czas=1 ms

Statystyka badania dla fe80::204:5aff:fe9f:9e80%2:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0 (0% straty),
Szacunkowy czas błądzenia pakietów w millisekundach:
    Minimum = 1 ms, Maksimum = 1 ms, Czas średni = 1 ms
```

Trasę datagramu IP prześledzić można za pomocą linuksowego programu `traceroute` (jego windowsowy odpowiednik ma nazwę `tracert` i nieco inny zestaw opcji); użycie parametru `-n` blokuje wyświetlanie nazw DNS przyporządkowanych wyświetlanym adresom:

```
Linux% traceroute -n ftp.uu.net
traceroute to ftp.uu.net (192.48.96.9), 30 hops max, 38 byte packets
 1  70.231.159.254 9.285 ms 8.404 ms 8.887 ms
 2  206.171.134.131 8.412 ms 8.764 ms 8.661 ms
 3  216.102.176.226 8.502 ms 8.995 ms 8.644 ms
 4  151.164.190.185 8.705 ms 8.673 ms 9.014 ms
 5  151.164.92.181 9.149 ms 9.057 ms 9.537 ms
 6  151.164.240.134 9.680 ms 10.389 ms 11.003 ms
 7  151.164.41.10 11.605 ms 37.699 ms 11.374 ms
 8  12.122.79.97 13.449 ms 12.804 ms 13.126 ms
 9  12.122.85.134 15.114 ms 15.020 ms 13.654 ms
     MPLS Label=32307 CoS=5 TTL=1 S=0
10  12.123.12.18 16.011 ms 13.555 ms 13.167 ms
11  192.205.33.198 15.594 ms 15.497 ms 16.093 ms
12  152.63.57.102 15.103 ms 14.769 ms 15.128 ms
13  152.63.34.133 77.501 ms 77.593 ms 76.974 ms
14  152.63.38.1 77.906 ms 78.101 ms 78.398 ms
15  207.18.173.162 81.146 ms 81.281 ms 80.918 ms
16  198.5.240.36 77.988 ms 78.007 ms 77.947 ms
17  198.5.241.101 81.912 ms 82.231 ms 83.115 ms
```

Widzimy listę przeskoków na trasie od komputera wykonującego program `traceroute` do węzła docelowego *ftp.uu.net* (`192.48.96.9`). Program `traceroute` opiera swe działanie na generowaniu kombinacji datagramów UDP z sukcesywnie zwiększaną wartością pola *Czas życia* (w IPv6 *Limit przeskoków*) i komunikatów ICMP (wykorzystywanych do wykrywania sytuacji, gdy w danym węźle wyczerpuje się limit czasu życia (przeskoków) datagramu — w ten właśnie sposób wykrywane jest istnienie węzła na trasie). Dla każdej wartości *Czasu życia/Limitu przeskoków* wysyłane są trzy datagramy UDP, w celu trzykrotnego pomiaru czasu wędrówki datagramu od nadawcy do każdego z węzłów.

Standardowo program `traceroute` przetwarza tylko informację związaną z protokołem IP, na listingu widzimy jednak wiersz

```
MPLS Label=32307 CoS=5 TTL=1 S=0
```

Jest on konsekwencją zastosowania techniki MPLS (*MultiProtocol Label Switching* — wieloprotokołowe przełączanie etykiet) opisywanej w dokumencie [RFC3031] i wiążącej z pakietami specjalne etykiety, stanowiące podstawę forwardowania tych pakietów (przez routery obsługujące MPLS). Etykieta sygnalizowana w powyższym wierszu posiada identyfikator 32307, została zakwalifikowana do klasy usług 5, limit przeskoków określono na 1, a komunikat reprezentowany przez etykietę nie znajduje się na dnie stosu MPLS (S=0, patrz [RFC4950]). Współdziałanie MPLS z protokołem ICMP opisane jest w dokumencie [RFC4950], natomiast ich łączenie z pakietami IPv4 (na zasadzie opcji IP) opisano w [RFC6178]. Wielu operatorów sieci wykorzystuje MPLS na potrzeby kontrolowania dróg przepływu poszczególnych części ruchu.

5.4.4. Dyskusja

Przedstawione typowe przykłady pozwalają na sformułowanie kilku spostrzeżeń związanych z forwardowaniem datagramów IP opatrzonych adresami docelowymi typu unicast.

1. Większość hostów i routerów posiada w swych tablicach trasowania pozycję o zerowej masce i zerowym adresie wynikowym, reprezentującą kolejny przeskok na domyślnej trasie pakietu. Istotnie, typowy host lub router znajdujący się na styku Internetu z siecią lokalną kieruje na domyślną trasę wszystkie pakiety nieprzeznaczone dla własnej sieci, posiada bowiem tylko jeden interfejs łączący go z Internetem.
2. Adresy IP źródłowy i docelowy, zawarte w nagłówku datagramu, nie ulegają zmianie na trasie pakietu, z wyjątkiem stosowania opcji *Trasowanie źródłowe* lub procedury NAT. Decyzje dotyczące trasowania datagramu podejmowane są przez hosty i routery na podstawie docelowego adresu IP.
3. Datagramy IP enkapsulowane są w ramkach warstwy łącza danych, w których adresy docelowe (o ile występują) identyfikują zawsze najbliższy przeskok na trasie, zmieniają się więc przy kolejnych przeskokach. W przedstawionych przykładach adresy warstwy łącza danych obecne były w nagłówkach ramek ethernetowych, lecz już nie w ramkach łącza DSL. W protokole IPv4 przeliczanie adresów IP na adresy sprzętowe jest zadaniem protokołu ARP, w protokole IPv6 zadanie to spełniają komunikaty *Neighbor Solicitation* protokołu ICMPv6.

5.5. Mobilny IP

Opisywany dotychczas model forwardowania datagramów IP posiada ważną cechę charakterystyczną: jest nią niezmienność wzajemnej lokalizacji węzłów sieci — każdy host współdzieli ten sam prefiks z najbliższymi hostami i routerami. Zmieniając niespodziewanie adres IP węzła i jednocześnie pozostawiając ów węzeł przyłączony do sieci na poziomie warstwy łącza danych, ryzykujemy załamanie się połączeń nawiązanych w warstwach wyższych (np. połączeń TCP), bo połączenia te wciąż bazują na starym adresie IP wspomnianego węzła i poprawne trasowanie związanych z nimi datagramów może okazać się niemożliwe. Problem ten nabiera poważnego wymiaru użytkowego w przypadku węzłów mobilnych, a wieloletnie (teraz już — kilkudziesięcioletnie) wysiłki projektantów zmierzające do jego rozwiązania zaowocowały skonstruowaniem mechanizmu

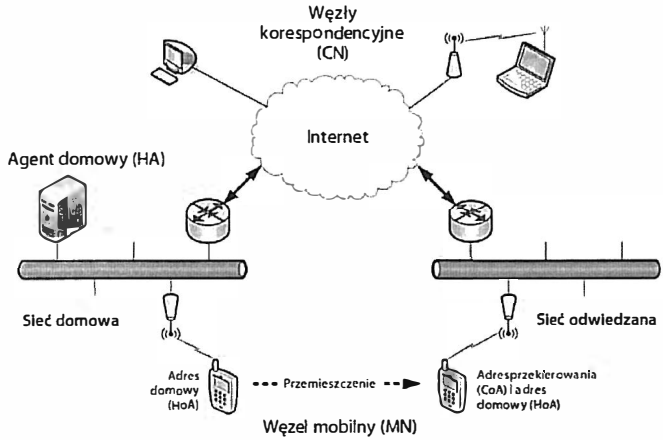
mobilnego IP (*Mobile IP*) (propozycje innych protokołów tego rodzaju przedstawione są w dokumencie [RFC6301]). Mobilny IP istnieje w wersjach dla obu protokołów IPv4 ([RFC5944]) oraz IPv6 ([RFC6275]), skoncentrujemy się jednak wyłącznie na wersji dla IPv6 (oznaczanej dalej skrótowo MIPv6) jako bardziej elastycznej, łatwiejszej do objaśnienia i bardziej rozpowszechnionej, zwłaszcza w nowszych urządzeniach mobilnych, głównie smartfonach. Oczywiście, nie sposób w tak zwięzłych ramach przedstawić wszystkich istotnych jej cech, dlatego ograniczymy się do przedstawienia jedynie podstawowych koncepcji i zasad, polecając zainteresowanym czytelnikom literaturę dedykowaną tematowi, np. publikację [RC05].

Mobilny IP opiera się na koncepcji **sieci domowej** hosta (*home network*) — jest to sieć, w ramach której host przebywa przez większość czasu i którą może opuszczać, przemieszczając się do innych sieci. Komunikacja hosta z jego siecią domową odbywa się zgodnie z opisanymi wcześniej algorytmami. Gdy host opuszcza swą sieć domową, zapamiętuje swój „domowy" adres IP i (przy użyciu różnych trików i specjalnych algorytmów forwardowania) prezentuje się pod tym adresem nowej sieci i wszystkim hostom, z którymi nawiązuje komunikację — tak jakby nadal znajdował się w swej domowej sieci. Podstawowym elementem tego schematu jest specjalny rodzaj routera, zwany **agentem domowym** (*home agent*), wspomagający trasowanie związane z mobilnymi węzłami.

Stopień złożoności MIPv6 uprawnia do uznania go za odrębny protokół. Złożoność ta objawia się głównie w postaci komplikacji mechanizmu komunikatów sygnalizacyjnych i ich zabezpieczania. Komunikaty te wykorzystują różne formy nagłówka rozszerzenia *Mobilność*, identyfikowanego (zgodnie z tabelą 5.5) wartością 135 w polu *Następny nagłówek*. Występuje on w wielu (obecnie 17) odmianach, szczegółowo opisanych na stronie [MP]. Omawiając model komunikacji mobilnego IP, skoncentrujemy się na podstawowych komunikatach opisywanych w [RFC6275]; inne komunikaty wykorzystywane są m.in. do implementowania „szybkich urządzeń podręcznych" (*fast handovers*) ([RFC5568]), do zmiany agenta domowego ([RFC5142]) i do celów eksperymentalnych ([RFC5096]). Rozpoczniemy od przedstawienia ogólnego modelu komunikacji MIPv6 i związanej z nim terminologii.

5.5.1. Model podstawowy — tunelowanie dwukierunkowe

Na rysunku 5.17 widoczne są podstawowe elementy infrastruktury MIPv6; większość terminologii odnosi się również do MIPv4 ([RFC5944]). I tak, host który może się przemieszczać, nazywamy **węzłem mobilnym** (*Mobile Node* — MN), zaś węzły, z którymi może się on komunikować, określane są mianem **węzłów korespondencyjnych** (*Correspondent Nodes* — CN). MN opatrzony zostaje adresem IP utworzonym na bazie prefiksu IP jego sieci domowej — jest to jego **adres domowy** (*Home Address* — HoA). Gdy MN przemieści się do innej sieci, otrzymuje dodatkowy adres, zwany **adresem przekierowania** (*Care-of Address* — CoA). W opisywanym modelu podstawowym komunikacja CN z MN trasowana jest za pomocą **agenta domowego** (*Home Agent* — HA) — to specjalny rodzaj routera, równorzędny w infrastrukturze sieciowej z innymi ważnymi systemami (np. routerami i serwerami WWW). Skojarzenie adresów HoA i CoA węzła mobilnego nazywamy **wiązaniem** (*binding*) tego węzła.



Rysunek 5.17. *Mobilny IP umożliwia utrzymywanie operatywności węzłów, mimo ich przemieszczania się między sieciami. Agent domowy węzła pośredniczy w jego komunikacji z innymi węzłami (w wariancie podstawowym), a w wersji zoptymalizowanej organizuje bezpośrednie skojarzenie węzła mobilnego z węzłem korespondentem*

W podstawowym modelu komunikacji węzły CN nie angażują protokołu MIPv6. Odmianą tego modelu jest sytuacja, gdy mobilna jest całą sieć, wraz ze swym routerem (mechanizm taki, o sugestywnej nazwie „NEMO" — od *Network Mobility* — opisywany jest w dokumencie [RFC3963]). Gdy MN (lub mobilny router sieci) przyłączony zostaje do nowej sieci, otrzymuje CoA i komunikuje go HA za pośrednictwem komunikatu **aktualizacja wiązania** (*binding update*); w odpowiedzi HA przesyła do MN komunikat **potwierdzenie wiązania** (*binding acknowledgment*). Odtąd ruch między MN a jego CN odbywać się będzie za pośrednictwem wspomnianego HA, przy użyciu dwustronnej formy tunelowania pakietów IPv6 (patrz [RFC2473]), zwanej **tunelowaniem dwukierunkowym** (*bidirectional tunneling*). Wymieniane w związku z tym komunikaty chronione są za pomocą protokołu IPsec w wersji z nagłówkiem ESP (*Encapsulating Security Payload* — patrz rozdział 18.). Stanowi to zabezpieczenie przed możliwością nawiązania skojarzenia z HA przez intruzywną stację, podszywającą się pod legalny MN.

5.5.2. Optymalizacja trasy (RO)

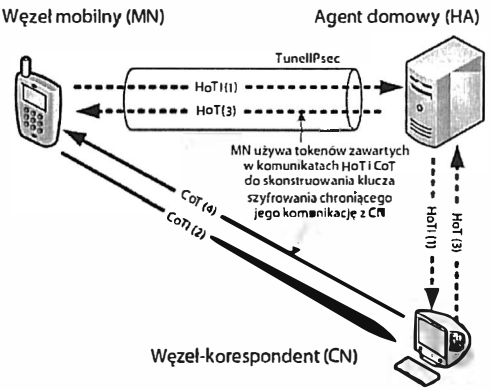
Tunelowanie dwukierunkowe jest stosunkowo nieskomplikowaną techniką realizacji MIPv6 w komunikacji z węzłami CN, które mogą nawet nie implementować MIP, owa prostota może jednak przekładać się na znaczną nieefektywność komunikacji: szczególnie kuriozalna jest sytuacja, gdy MN utrzymuje komunikację z pobliskimi CN, lecz znajduje się w znacznym oddaleniu od swego HA, który *musi* w tej komunikacji pośredniczyć. Wyeliminowanie tego mankamentu jest zadaniem procesu zwanego **optymalizacją trasy** (*route optimization*, w skrócie RO); proces ten musi być wspierany zarówno przez MN, jak i wszystkie CN, z którymi ten się komunikuje. Jak za chwilę pokażemy, metody zapewniające użyteczność i bezpieczeństwo RO są dość skomplikowane, ograniczymy się

więc tylko do naszkicowania podstawowych operacji RO, odsyłając czytelników zainteresowanych szczegółami do lektury dokumentów [RFC6275] i [RFC4866]; racjonalizacja RO — czyli uzasadnienie wyboru takich, a nie innych rozwiązań szczegółowych — zawarta jest w dokumencie [RFC4225].

Podstawową ideą RO jest **rejestracja korespondencyjna**, czyli proces, w ramach którego MN uwierzytelnia się wobec CN, jednocześnie informując je o swym aktualnym adresie CoA — to wszystko w celu nawiązania komunikacji bezpośredniej, nieangażującej HA. Z tego punktu widzenia, funkcjonowanie RO można podzielić na dwa podprocesy: jeden z nich zajmuje się wspomnianą rejestracją korespondencyjną i w efekcie nawiązaniem skojarzenia między MN a CN, drugi natomiast zajmuje się wymianą datagramów między MN a skojarzonymi z nim CN.

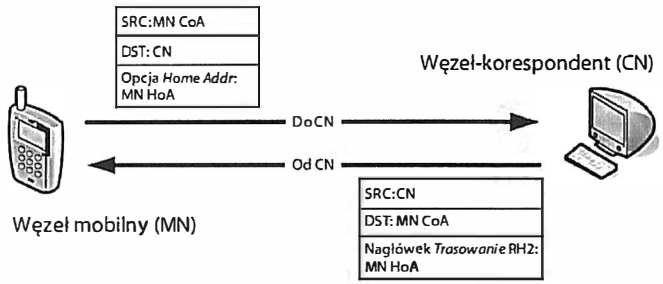
W ramach rejestracji korespondencyjnej MN uwiarygodnia się najpierw wobec każdego CN, wykonując **procedurę trasowalności powrotnej** (*Return Routability Procedure*, w skrócie RRP). Komunikaty wymieniane w ramach RRP nie są chronione przez IPsec (ochrona taka miała miejsce w przypadku komunikacji MN ze swym HA), ponieważ funkcjonowanie IPsec w tym kontekście byłoby (zdaniem projektantów) dość zawodne — co prawda, od każdej implementacji protokołu IPv6 wymaga się zaimplementowania IPsec, jednak nie ma obowiązku jego używania. I choć RRP nie zapewnia ochrony tak solidnej jak IPsec, to jednak jest od niego znacznie prostsza i wychodzi naprzeciw większości problemów bezpieczeństwa, jakie rozważali projektanci MIP.

Procedura RRP obejmuje wymianę następujących *komunikatów mobilności*: HoTI (*Home Test Init*), HoT (*Home Test*), CoTI (*Care-of Test Init*) oraz CoT (*Care-of Test*). Ich zadaniem jest weryfikacja przez CN osiągalności MN zarówno na podstawie adresu domowego (komunikaty HoTI i HoT), jak i adresu CoA (komunikaty CoTI i CoT). Schemat działania RRP przedstawiono na rysunku 5.18.



Rysunek 5.18. *Wymiana komunikatów w ramach procedury trasowalności powrotnej (RRP) przygotowującej zoptymalizowaną, bezpośrednią komunikację między MN a CN; CN zostaje poinformowany o dostępności MN pod obydwoma adresami: domowym i przekierowania. Linie przerywane reprezentują komunikaty wymieniane za pośrednictwem agenta domowego; cyfry oznaczają kolejność wysyłania komunikatów, choć komunikaty (1) i (2) mogą zostać wysłane przez MN równocześnie*

Na rysunku tym widzimy banalny przypadek jednego MN i jednego CN. Procedurę RRP rozpoczyna MN, wysyłając do CN dwa komunikaty: HoTI (za pośrednictwem HA) oraz CoTI (bezpośrednio do CN). CN po otrzymaniu obu komunikatów (w dowolnej kolejności) odpowiada wysłaniem do MN dwóch komunikatów: HoT (za pośrednictwem wspomnianego HA) oraz CoT (bezpośrednio do MN). Komunikaty te niosą losowe ciągi znaków, zwane **tokenami**, służące do skonstruowania klucza kryptograficznego używanego do uwierzytelniania MN wobec CN w ramach procedury ich kojarzenia. Po zakończeniu procedury węzły MN i CN mogą komunikować się w sposób bezpośredni, jak na rysunku 5.19 — trasa wymienianych między nimi datagramów została zoptymalizowana. RO ukazuje swe drugie oblicze, jakim jest metoda bezpośredniego komunikowania się MN i CN, zilustrowana w uproszczeniu na rysunku 5.19.



Rysunek 5.19. *Po nawiązaniu skojarzenia MN i CN wymieniają dane w sposób bezpośredni. Pakiety wędrujące od MN do CN wykorzystują opcję docelową Adres domowy, pakiety wędrujące w kierunku przeciwnym posiadają nagłówek Trasowanie w wersji RH2*

Po pomyślnym przeprowadzeniu kojarzenia dane między MN i CN przepływać mogą w sposób bezpośredni, wolne od nieefektywności właściwej tunelowaniu dwukierunkowemu. Komunikacja ta organizowana jest z użyciem *Opcji docelowych* IPv6 dla datagramów przepływających od MN do CN oraz nagłówka *Trasowanie* RH2 dla datagramów płynących w kierunku przeciwnym; w nagłówku tym adres HoA węzła MN wskazywany jest jawnie jako następny (jedyny) przeskok.

Pakiety wysyłane przez CN zawierają w polu *Adres źródłowy IP* adres CoA węzła MN, co pozwala na uniknięcie problemu polegającego na odrzucaniu pakietów w ramach tzw. **filtrowania wprowadzającego** (*ingress filtering*) (patrz [RFC2827]) — odrzucanie takie mogłoby następować, gdyby w roli adresu źródłowego datagramów wykazywany był adres HoA węzła MN. Notabene adres ten figuruje w pakiecie, lecz znajduje się w polu danych opcji *Home Address* i jako taki nie jest interpretowany (ani modyfikowany) przez routery.

Węzeł CN wysyła więc pakiety przeznaczone dla MN na jego adres CoA (bo taki odczytał z pola *Adres źródłowy IP* w nagłówku podstawowym). Węzeł MN po pomyślnym odebraniu pakietu od CN wykonuje ciekawy trik, polegający na wpisaniu w pole *Adres docelowy IP* tegoż pakietu swego adresu HoA, odczytanego z nagłówka RH2. Tak spreparowany pakiet przekazywany jest pozostałym protokołom stosu TCP/IP implementowanym w węźle MN, wskutek czego protokoły traktują ów pakiet tak, jakby wysłany został na adres HoA, nie CoA, węzła MN.

5.5.3. Dyskusja

Mobilny IP zaprojektowany został z myślą o obsłudze (do pewnego stopnia) łączności mobilnej, w warunkach której adres urządzenia może się dynamicznie zmieniać, lecz zarazem urządzenie to musi pozostać połączone z warstwą łącza danych. Problem ten w mniejszym stopniu dotyczy komputerów przenośnych, które przed przemieszczeniem są zwykle wyłączane, usypiane lub hibernowane[6], jest natomiast typowy dla urządzeń podręcznych (głównie smartfonów), na których uruchamiane są aplikacje czasu rzeczywistego (np. VoIP). Podejmowane są więc różne wysiłki zmierzające do minimalizowania czasu kojarzenia węzłów, odnotować na tym polu należy przede wszystkim technologię **szybkich urządzeń podręcznych** (*fast handovers*) opisywaną w [RFC5568], rozszerzenie MIPv6 nazywane **hierarchicznym MIPv6** (HMIPv6) i opisywane w [RFC5380], a także użycie serwera proxy do odciążenia MN od przetwarzania komunikatów sygnalizacyjnych (technika ta, nazywana po prostu „proxy MIPv6" lub „PMIPv6", opisywana jest w [RFC5213]).

5.6. Przetwarzanie datagramów IP przez host

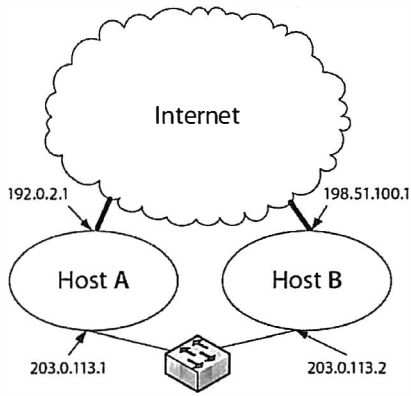
Chociaż routery zasadniczo wolne są od obowiązku wpisywania źródłowego i docelowego adresu IP do przetwarzanych pakietów, host (zarówno ten wysyłający, jak i odbierający pakiety) musi troszczyć się o zawartość tych pól. Przykładowo aplikacja w rodzaju przeglądarki WWW może łączyć się z serwerem posiadającym kilka adresów IP, podobnie komputer kliencki uruchamiający tę przeglądarkę może być także identyfikowany przez wiele adresów IP. Rodzi to m.in. naturalny problem określenia, którego z dostępnych adresów IP (czy nawet — której wersji protokołu IP) należy użyć do wysłania konkretnego datagramu; podobnym problemem jest rozstrzyganie o legalności pakietu, który dotarł do hosta przez „nie ten" interfejs, czyli interfejs nieprzeznaczony do obsługi adresu docelowego figurującego w nagłówku datagramu.

5.6.1. Modele hosta

Gdy do hosta dociera datagram opatrzony adresem docelowym unicast identycznym z którymś z adresów IP tegoż hosta, decyzja o podjęciu przetwarzania odebranego datagramu wydaje się być oczywista; w rzeczywistości jest ona uzależniona od *modelu hosta* przyjętego w systemie odbierającym wspomniany datagram (patrz [RFC1122]) i wbrew pozorom oczywista często nie jest, szczególnie w systemach *multihomed*. Istnieją dwa modele hostów: **silny** (*strong*) i **słaby** (*weak*) — jak można oczekiwać, bardziej restrykcyjny jest model silny. Istotnie, w modelu silnym protokoły stosu TCP/IP otrzymują datagram odebrany przez host tylko wtedy, gdy zawarty w tym datagramie adres docelowy zgodny jest z którymś z adresów IP przypisanych do interfejsu, za pośrednictwem którego ów datagram do hosta dotarł. W modelu słabym kontrola ta jest mniej rygorystyczna, obejmuje jedynie zgodność adresu docelowego datagramu z którymś z lokalnych adresów hosta, kwestia interfejsu dostarczającego datagram jest nieistotna. Analogicznie ma się rzecz z wysyłaniem datagramów: w modelu silnym adres źródłowy w wysyłanym datagramie musi zgadzać się z którymś z adresów przypisanych do interfejsu realizującego wysyłanie, w modelu słabym nie ma takiego wymogu.

[6] Niekoniecznie, coraz więcej komputerów wykorzystuje szerokopasmowe modemy komórkowe — *przyp. tłum.*

Na rysunku 5.20 przedstawiono jedną z typowych sytuacji, w której wybór odpowiedniego modelu hosta staje się kwestią bardzo istotną. Dwa hosty — **A** i **B** — połączone są w dwojaki sposób: za pomocą Internetu oraz za pośrednictwem sieci lokalnej. Jeśli host **A** skonfigurowany jest do pracy według modelu silnego, przychodzące doń z Internetu datagramy o adresie docelowym `203.0.113.1` będą odrzucane, taki sam los spotykać będzie pakiety o adresie docelowym `192.0.2.1` przychodzące do hosta **A** przez sieć lokalną. Jeśli host **B** skonfigurowany jest na pracę według modelu słabego, może zdecydować o wysłaniu datagramu na adres `192.0.2.1` przez interfejs prowadzący do sieci lokalnej, jako zapewniający bardziej efektywną trasę (w porównaniu z trasą przez Internet). W rezultacie pakiet, legalnie wysłany przez host **B** do hosta **A**, zostanie przez ten ostatni odrzucony — ze względu na stosowany model operacyjny. Nie sposób w tym kontekście nie zadać fundamentalnego pytania — pytania o sam sens istnienia modelu silnego.



Rysunek 5.20. *Hosty mogą być połączone w różny sposób, przy użyciu różnych interfejsów, konieczne jest więc dla każdego z hostów właściwe określenie adresów źródłowego i docelowego w wysyłanych datagramach. Wybór wspomnianych adresów dokonywany jest przez host na podstawie jego tablicy forwardowania, algorytmu rankingowego (patrz [RFC3484]) i stosowanego modelu (silnego albo słabego)*

Ów sens da się sprowadzić do wiele mówiącego, magicznego słowa — bezpieczeństwo. Załóżmy, że złośliwy internauta wprowadził do Internetu pakiet przeznaczony dla adresu `203.0.113.2` i zawierający wartość (oczywiście, fikcyjną) `203.0.113.1` w polu adresu źródłowego. Działająca w hoście **B** aplikacja odbiera ów pakiet, wierząc, że utworzony i nadany został przez host **A**. Jeśli w gestii wspomnianej aplikacji spoczywają decyzje w kwestii np. kontroli dostępu, podejmowane na podstawie adresu źródłowego pakietu, fatalne konsekwencje podstępu zastosowanego przez intruza nietrudno sobie wyobrazić. Gdyby host **B** funkcjonował w trybie modelu silnego, nie przyjąłby rzeczonego pakietu na interfejsie prowadzącym do Internetu.

Większość systemów operacyjnych umożliwia jawny wybór modelu hosta; w systemie Windows (w wersji Vista i 7) model silny jest domyślny dla odbierania i wysyłania datagramów, w obu wersjach IPv4 i IPv6. Użytkownik może jednak zmieniać ten domyślny stan, włączając lub wyłączając model słaby za pomocą polecenia o postaci

```
C:\> netsh interface ipv<x> set interface <nazwa> <kierunek> <stan>
```

Parametr <*stan*> określa włączenie (`enabled`) albo wyłączenie (`disabled`) słabego modelu, <*x*> jest numerem wersji protokołu IP (4 albo 6), <*nazwa*> jest nazwą interfejsu, <*kierunek*> określa ustawianie słabego modelu dla wysyłania (`weakhostsend`) albo odbierania (`weakhostreceived`) datagramów.

W Linuksie domyślnym modelem jest model słaby, w systemach linii BSD (włącznie z Mac OS X) — model silny.

5.6.2. Selekcja adresów

Nieodłączną czynnością towarzyszącą wysyłaniu przez host datagramu IP jest właściwe ustawienie pól *Adres źródłowy* i *Adres docelowy* w nagłówku tego datagramu. W niektórych sytuacjach adres źródłowy określony jest jednoznacznie bądź to przez aplikację, bądź też przez fakt, że wysyłany datagram jest odpowiedzią na inny, wcześniej odebrany w ramach tego samego połączenia (np. połączenia TCP — w rozdziale 13. zajmiemy się problematyką zarządzania adresami przez protokół TCP).

W nowoczesnych implementacjach protokołu IP wybór wspomnianych adresów jest zadaniem zaawansowanych **procedur selekcyjnych**. Dawno temu, u zarania Internetu, typowy host wykorzystywał tylko jeden adres IP na potrzeby zewnętrznej komunikacji, więc procedury te miały charakter oczywisty. Sytuacja zmieniła się diametralnie w związku z powszechnym wykorzystywaniem wielu adresów IP dla jednego interfejsu, no i — oczywiście — w związku z pojawieniem się IPv6, zgodnie z regułami którego równoczesne wykorzystywanie, dla danego interfejsu, wielu adresów o zróżnicowanym zakresie jest na porządku dziennym. Sytuacja komplikuje się dodatkowo w przypadku hostów implementujących tzw. dualny stos TCP/IP, czyli oba protokoły IPv4 i IPv6 (patrz [RFC4213]); niewłaściwy wybór adresów może prowadzić do różnych niepożądanych efektów ubocznych: asymetrycznego trasowania, nieuzasadnionego filtrowania lub odrzucania pakietów z innych przyczyn. Skuteczne rozwiązanie tych problemów stanowi nie lada wyzwanie dla projektantów.

W przypadku hostów implementujących wyłącznie IPv4 sprawy nie są aż tak skomplikowane. Dla implementacji IPv6 reguły domyślnego wyboru adresów zawarte są w dokumencie [RFC3484], aplikacje powinny mieć możliwość zmiany domyślnych ustawień za pomocą odpowiednich funkcji API (czyhające w związku z tym pułapki są przedmiotem rozważań dokumentu [RFC5220]). Generalnie rzecz biorąc, wspomniane reguły domyślnie zalecają wybór pary adresów „źródłowy-docelowy" o tym samym zakresie, preferowanie węższych zakresów kosztem szerszych dla adresów docelowych, zaś dla adresów źródłowych preferowanie adresów publicznych kosztem tymczasowych; w komunikacji mobilnej adresy domowe preferowane są kosztem adresów przekierowania. Ostatnie słowo w tej kwestii należy do administratora, który ma możliwość zastępowania reguł domyślnych własną polityką selekcyjną — nie będziemy się jednak zajmować tą kwestią, jako specyficzną dla konkretnego środowiska.

Działanie procedury selekcyjnej sterowane jest przez **tablicę założeń**, obecną (przynajmniej koncepcyjnie) w każdym hoście i konfigurowaną na podstawie parametrów określonych przez administratora. Tablica założeń sugerowana przez dokument [RFC3484] — o ile jej zawartość nie stoi w sprzeczności z założeniami administracyjnymi — widoczna jest jako tabela 5.10. Podobnie jak w przypadku tablicy trasowania (patrz punkt 5.4.2),

wybór jednej z pozycji odbywa się na zasadzie najdłuższego pasującego prefiksu. Dla każdego adresu (prefiksu) x określone jest pierwszeństwo wyboru $P(x)$ — im większa jego wartość, tym większa preferencja w wyborze. Przyporządkowywane poszczególnym pozycjom etykiety $L(x)$ służą do grupowania adresów o podobnym typie: jeśli dla pary adresów x i y spełniony jest warunek $L(x) = L(y)$, para ta jest preferowana w wyborze do roli adresów „źródłowy-docelowy".

Tabela 5.10. *Domyślna tablica założeń, sugerowana przez [RFC3484]. Większa wartość pierwszeństwa oznacza większą preferencję przy wyborze*

Prefiks	Pierwszeństwo P(x)	Etykieta L(x)
::1/128	50	0
::/0	40	1
2002::/16	30	2
::/96	20	3
::ffff:0:0/96	10	4

Procedura selekcyjna wykorzystuje ponadto następujące własności określone dla danego adresu lub pary adresów.

- CPL(A, B) to długość (w bitach) wspólnego prefiksu (*common prefix length*), czyli najdłuższego ciągu identycznych bitów lewostronnych, adresów IPv6 A i B.
- S(A) to zakres (*scope*) adresu A odwzorowany na wartość numeryczną; większa wartość odpowiada szerszemu zakresowi, jeśli więc A jest adresem lokalnym dla łącza, a B jest adresem globalnym, to $S(A) < S(B)$.
- M(A) to wynik mapowania adresu IPv4 A na adres IPv6. Ponieważ zakres adresu IPv4 jest funkcją jego wartości, więc konieczne jest honorowanie przez implementację następującej relacji:

```
S(M(169.254.x.x)) = S(M(127.x.x.x)) < S(M(<jakikolwiek adres prywatny IPv4>))
↪< S(M(jakikolwiek inny adres IPv4))
```

- $\Lambda(A)$ jest czasem życia adresu A; jeśli A jest adresem przestarzałym, niezalecanym do użytku (*deprecated*), a B adresem zalecanym (*preferred*), to $\Lambda(A) < \Lambda(B)$,
- W kontekście mobilnego IP: predykat H(A) ma wartość `true`, jeśli A jest adresem domowym, i `false` w przeciwnym razie, natomiast predykat C(A) ma wartość `true`, jeśli A jest adresem przekierowania, i `false` w przeciwnym razie.

5.6.2.1. Algorytm selekcji adresu źródłowego

Oznaczmy przez CS(D) zbiór *adresów kandydackich*, czyli potencjalnie możliwych adresów IP bazujących na określonym przeznaczeniu D. Z definicji do zbioru tego nie należą adresy anycast, multicast i adres nieokreślony. W zbiorze CS(D) definiuje się następujące funkcje.

- $R_D(A)$ oznacza *ranking* adresu A w zbiorze CS(D): jeśli $R_D(A) > R_D(B)$, to *ranking* adresu A jest większy niż ranking adresu B, co zapisujemy jako $R_D(A)$ *> $R_D(B)$, a co oznacza, że A preferowany jest względem B jako adres źródłowy dla osiągnięcia hosta o adresie D.

- I(D) oznacza interfejs wybrany dla forwardowania datagramu w kierunku przeznaczenia D, na zasadzie dopasowania względem najdłuższego prefiksu (o czym pisaliśmy w punkcie 5.4.2).
- @(I) jest zbiorem adresów IP przypisanych do interfejsu I.
- T(A) jest predykatem przyjmującym wartość `true`, jeśli A jest adresem tymczasowym (patrz rozdział 6.), i `false` w przeciwnym razie.

W oparciu o powyższe funkcje w zbiorze CS(D) definiowana jest relacja częściowego porządku, w kolejności stosowania następujących reguł dla dwóch dowolnych adresów A i B.

1. Preferowanie tożsamości adresów: jeśli $A = D$ i $B \neq D$, to $R_D(A)$ *> $R_D(B)$, i vice versa: jeśli $A \neq D$ i $B = D$, to $R_D(B)$ *> $R_D(A)$.
2. Preferowanie szerszego zakresu: jeśli $S(A) < S(B)$ i $S(A) < S(D)$, to $R_D(B)$ *> $R_D(A)$, w przeciwnym razie $R_D(A)$ *> $R_D(B)$. I vice versa: jeśli $S(B) < S(A)$ i $S(B) < S(D)$, to $R_D(A)$ *> $R_D(B)$, w przeciwnym razie $R_D(B)$ *> $R_D(A)$.
3. Unikanie adresów przestarzałych — w sytuacji gdy $S(A) = S(B)$: jeśli $\Lambda(A) < \Lambda(B)$, to $R_D(A)$ *> $R_D(B)$, w przeciwnym razie $R_D(A)$ *> $R_D(B)$.
4. Preferowanie adresów domowych ($\wedge$ oznacza koniunkcję, $\neg$ oznacza zaprzeczenie):

   a) jeśli $H(A) \wedge C(A) \wedge \neg (H(B) \wedge C(B))$, to $R_D(A)$ *> $R_D(B)$

   b) jeśli $H(B) \wedge C(B) \wedge \neg (H(A) \wedge C(A))$, to $R_D(B)$ *> $R_D(A)$

   c) jeśli $H(A) \wedge \neg C(A) \wedge \neg H(B) \wedge C(B)$, to $R_D(A)$ *> $R_D(B)$

   d) jeśli $H(B) \wedge \neg C(B) \wedge \neg H(A) \wedge C(A)$, to $R_D(B)$ *> $R_D(A)$
5. Preferowanie interfejsów dla ruchu wychodzącego: jeśli $A \in @(I(D)$ i $B \notin @(I(D)$, to $R_D(A)$ *> $R_D(B)$. I vice versa: jeśli $A \notin @(I(D)$ i $B \in @(I(D)$, to $R_D(B)$ *> $R_D(A)$
6. Preferowanie identyczności etykiet: jeśli $L(A) = L(D)$ i $L(B) \neq L(D)$, to $R_D(A)$ *> $R_D(B)$. I vice versa: jeśli $L(A) \neq L(D)$ i $L(B) = L(D)$, to $R_D(B)$ *> $R_D(A)$
7. Preferowanie adresów trwałych: jeśli $T(A)$ i $\neg T(B)$, to $R_D(B)$ *> $R_D(A)$. I vice versa: jeśli $T(B)$ i $\neg T(A)$, to $R_D(A)$ *> $R_D(B)$.
8. Preferowanie dłuższego wspólnego prefiksu: jeśli $CPL(A, D) > CPL(B, D)$, to $R_D(A)$ *> $R_D(B)$. I vice versa: jeśli $CPL(B, D) > CPL(A, D)$, to $R_D(B)$ *> $R_D(A)$.

Stosowanie powyższych reguł — w wymienionej kolejności — powinno (choć niekoniecznie) doprowadzić ostatecznie do wyboru adresu maksymalnego względem relacji *>, czyli takiego adresu Ψ, że $R_D(\Psi)$ *> $R_D(x)$ dla każdego $x \in CS(D) - \{\Psi\}$. Adres ten tworzy jednoelementowy zbiór oznaczany przez Q(D) i stanowiący podstawę dla algorytmu wyboru adresu docelowego. Jeśli zaprezentowany ciąg reguł nie doprowadza do wybrania adresu o maksymalnej randze, zbiór Q(D) jest zbiorem pustym.

5.6.2.2. Algorytm selekcji adresu docelowego

W podobny sposób przeprowadzana jest selekcja adresu docelowego, identyfikującego w datagramie jego przeznaczenie. Niech Q(x) oznacza rezultat wyboru adresu docelo-

wego dla przeznaczenia x, zgodnie z algorytmem opisanym w poprzednim podpunkcie — czyli zbiór jednoelementowy albo zbiór pusty. Na użytek algorytmu wyboru adresu docelowego wprowadzamy kolejne oznaczenia:

- $U(B)$ jest predykatem przyjmującym wartość `true`, jeśli adres B jest nieosiągalny z poziomu bieżącego hosta;
- $E(B)$ jest predykatem przyjmującym wartość `true`, jeśli adres A jest osiągalny za pomocą pewnego „enkapsulowanego transportu" (np. tunelowanego trasowania).

Analogicznie do przypadku selekcji adresu źródłowego, poniższy ciąg reguł, definiujących częściowy porządek między adresami kandydackimi tworzącymi zbiór $SD(S)$, zdefiniowano z intencją wyłonienia adresu o największym rankingu R_S.

1. Unikanie bezużytecznych adresów: jeśli $U(B)$ ma wartość `true` lub $Q(B)$ jest zbiorem pustym, to $R_S(A) *> R_S(B)$; analogicznie, jeśli $U(A)$ ma wartość `true` lub $Q(A)$ jest zbiorem pustym, to $R_S(B) *> R_S(A)$.
2. Preferowanie zgodnych zakresów: jeśli $S(A) = S(Q(A))$ i $S(B) \neq S(Q(B))$, to $R_S(A) *> R_S(B)$; i vice versa — jeśli $S(A) \neq S(Q(A))$ i $S(B) = S(Q(B))$, to $R_S(B) *> R_S(A)$.
3. Unikanie adresów przestarzałych: jeśli $\Lambda(Q(A)) < \Lambda(Q(B))$, to $R_S(B) *> R_S(A)$. I vice versa: jeśli $\Lambda(Q(A)) > \Lambda(Q(B))$, to $R_S(A) *> R_S(B)$.
4. Preferowanie adresów domowych:

   a) jeśli $H(Q(A)) \wedge C(Q(A)) \wedge \neg (C(Q(B)) \wedge H(Q(B)))$, to $R_S(A) *> R_S(B)$

   b) jeśli $H(Q(B)) \wedge C(Q(B)) \wedge \neg (C(Q(A)) \wedge H(Q(B)))$, to $R_S(B) *> R_S(A)$

   c) jeśli $H(Q(A)) \wedge C(Q(A)) \wedge \neg H(Q(B)) \wedge H(Q(B))$, to $R_S(A) *> R_S(B)$

   d) jeśli $H(Q(B)) \wedge C(Q(A)) \wedge \neg H(Q(A)) \wedge H(Q(A))$, to $R_S(B) *> R_S(A)$
5. Preferowanie identyczności etykiet: jeśli $L(Q(A)) = L(A)$ i $L(Q(B)) \neq L(B)$, to $R_S(A) *> R_S(B)$. I vice versa: jeśli $L(Q(A)) \neq L(A)$ i $L(Q(B)) = L(B)$, to $R_S(B) *> R_S(A)$.
6. Preferowanie większego pierwszeństwa: jeśli $P(A) > P(B)$, to $R_S(A) *> R_S(B)$. I vice versa: jeśli $P(A) < P(B)$, to $R_S(B) *> R_S(A)$.
7. Preferowanie natywnego transportu: jeśli $E(A) \wedge \neg E(B)$, to $R_S(B) *> R_S(A)$; analogicznie, jeśli $\neg E(A) \wedge E(B)$, to $R_S(A) *> R_S(B)$.
8. Preferowanie węższego zakresu: jeśli $S(A) < S(B)$, to $R_S(A) *> R_S(B)$, w przeciwnym razie $R_S(B) *> R_S(A)$.
9. Preferowanie dłuższego wspólnego prefiksu: jeśli $CPL(A, Q(A)) > CPL(B, Q(B))$, to $R_S(A) *> R_S(B)$. I vice versa: jeśli $CPL(A, Q(A)) < CPL(B, Q(B))$, to $R_S(B) *> R_S(A)$.
10. Jeżeli nie ma zastosowania żadna z powyższych reguł, to o względnym porządku adresów decyduje ich kolejność na oryginalnej liście: jeśli A występuje na pozycji wcześniejszej niż B, to $R_S(A) *> R_S(B)$, w przeciwnym razie $R_S(B) *> R_S(A)$.

Tak jak przy wyborze adresu źródłowego, tak i tym razem celem powyższych reguł jest wyłonienie kandydata o największym rankingu spośród kandydatur składających się na zbiór $SD(S)$ dla danego źródła S. W odróżnieniu jednak od wyboru adresu źródłowego,

tym razem przedstawiona procedura zawsze kończy się powodzeniem, gdy bowiem zawiodą wszystkie inne kryteria, ostatecznym kryterium wyboru jest oryginalna pozycja kandydata na liście. Niektóre z operacji przedstawionej procedury kryją pewne pułapki, przykładowo w kroku 9. może wystąpić zjawisko „karuzeli DNS" (*DNS round-robin*) opisywane w rozdziale 11. Pułapki te stały się przesłanką do zrewidowania oryginalnej specyfikacji RFC3484 — poprawka [RFC3484-revise] wprowadza kilka zmian i nowości, m.in. uwzględnia w przedstawionej procedurze rankingowej tzw. unikatowe lokalne adresy IPv6 unicast (*Unique Local IPv6 Unicast Addresses*, w skrócie ULA), opisywane w [RFC4193], rozpoczynające się od prefiksu `fc00::/7`. Są one globalnie rozpoznawalne, lecz ich używanie ograniczone jest do wybranej (prywatnej) sieci lub miejsca sieciowego.

5.7. Ataki wykorzystujące protokół IP

W ciągu swego dość długiego już żywota Internet doświadczył wielu ataków dokonywanych za pośrednictwem protokołu IP; większość z nich sprowadzała się do nadużywania opcji IPv4 lub eksploatacji błędów tkwiących w implementacjach specjalizowanych operacji, m.in. reasemblacji fragmentów datagramu. Niesolidność wielu implementacji protokołu IP we wczesnych routerach powodowała, że bardzo łatwo można było paraliżować działanie tych routerów przez generowanie „bezsensownych" datagramów, np. zawierających kuriozalne wartości w polach *Wersja* czy *IHL*. Na szczęście, współczesne implementacje cechują się wystarczającym stopniem „idiotoodporności", by radzić sobie niezawodnie z próbami podstępów tego rodzaju, poza tym współczesne routery wykorzystywane w Internecie generalnie ignorują obecność opcji IP w datagramie. I choć nie sposób wymagać absolutnej bezbłędności od jakiegokolwiek oprogramowania, wskutek czego w oprogramowaniu routerów także pojawiają się luki zabezpieczeń, to jednak są one sukcesywnie eliminowane, m.in. w dokumentach [RFC1858] i [RFC3128] opisywane są techniki przeciwdziałania atakom wykonywanym przy użyciu mechanizmów reasemblacji datagramów. Twórcy exploitów mają więc coraz bardziej pod przysłowiową górkę.

Wiele nadużyć odbywa się obecnie za pomocą fabrykowania (*spoofing*) adresów IP w sytuacji, gdy datagramy nie są chronione przez szyfrowanie ani uwierzytelnianie. Ponieważ wiele wczesnych mechanizmów kontroli dostępu opierało się na adresach źródłowych IP, które, jak wiadomo, można fabrykować bez większego wysiłku, więc furtka nieuprawnionego dostępu w wielu systemach stała otworem dla intruzów, szczególnie gdy ci łączyli proste „podrabianie" adresów z innymi technikami, np. opcją *Trasowanie źródłowe*. W rezultacie zdalny komputer, próbujący wtargnąć do prywatnej sieci, postrzegany był jako uczestnik tejże sieci (a często nawet utożsamiany był z hostem, którego usługi zamierzał wyłudzać). I chociaż podrabianie adresów IP wciąż jest źródłem potencjalnych zagrożeń, opracowano wiele skutecznych mechanizmów poważnie ograniczających te zagrożenia, m.in. **filtrowanie wprowadzające** (*ingress filtering*), w ramach którego dostawcy Internetu weryfikują adresy źródłowe w datagramach przychodzących od klientów pod kątem zgodności z przydzielonymi tym klientom prefiksami IP.

Ponieważ IPv6 i mobilny IP są technologiami relatywnie młodymi w porównaniu z IPv4, trudno jeszcze mówić o jakimś systematycznym rozpoznaniu ich podatności na ataki wynikające ze specyfiki ich konstrukcji czy implementacji. Trzeba przyznać, że w obli-

czu nowego mechanizmu nagłówków rozszerzeń, znacznie bardziej elastycznego od opcji IPv4, potencjalni intruzi zyskują nowe pole do popisu. Wspominaliśmy już o ataku DoS realizowanym za pomocą oscylacji datagramów wywołanej odpowiednią konstrukcją nagłówka RH0 (patrz punkt 5.3.2), co stało się powodem zarzucenia jego obsługi w nowych implementacjach na rzecz poprawionej wersji RH2. Wersja ta jest jednak nieodporna (podobnie jak jej poprzednik) na podrabianie adresu IP, co stawia szczególne wyzwanie przed konstruktorami firewalli i konfigurującymi je administratorami. Zauważmy, że banalne odrzucanie datagramów zawierających nagłówki rozszerzeń jest pomysłem chybionym, bo uniemożliwiałoby funkcjonowanie mobilnego IP, a poważnie ograniczało wiele innych funkcji protokołu IPv6.

5.8. Podsumowanie

Rozdział ten poświęciliśmy protokołowi IP w dwóch najczęściej używanych wersjach — IPv4 i IPv6. Rozpoczęliśmy od opisu nagłówków w datagramach obu wersji; nagłówki te są całkowicie różne od siebie i muszą być przetwarzane w zupełnie inny sposób, jedynym ich wspólnym elementem jest pole *Wersja*, zajmujące pierwsze 4 bity i służące (zgodnie z nazwą) do rozróżniania obu wersji. Struktura nagłówka IPv6 jest wynikiem optymalizacji podyktowanych spostrzeżeniami nabytymi w związku z wieloletnim stosowaniem wersji IPv4: usunięto z nagłówka opcje, wykorzystywane selektywnie i raczej rzadko, dzięki czemu nagłówek zyskał ustaloną strukturę i ustalony rozmiar. Mimo czterokrotnego wydłużenia adresu IP, rozmiar nagłówka zwiększył się tylko dwukrotnie.

Nagłówek IP (w obu wersjach) zawierał pole charakteryzujące typ ruchu (w IPv6 nazywany klasą usługi) związany z danym datagramem. Wobec nikłej jego użyteczności, przedefiniowano po kilku latach jego znaczenie, dedykując je różnicowaniu usług świadczonych przez Internet, czyli zapewnieniu niektórym rodzajom usług większej wydajności w porównaniu z wariantem standardowym. Stopień wykorzystywania tej możliwości uwarunkowany jest aspektami nie tyle technicznymi, ile odpowiednim modelem biznesowym, obejmującym m.in. rozsądny i sprawiedliwy system opłat za uzyskiwane korzyści.

Forwardowanie IP to proces transportu datagramów przez sieć — prostą lub wieloskokową. Z wyjątkiem przypadków specjalnych, forwardowanie realizowane jest „skok po skoku". Docelowy adres IP w datagramie pozostaje niezmienny na całej jego trasie, zmienia się natomiast na poszczególnych przeskokach docelowy adres warstwy łącza danych w ramkach enkapsulujących datagram. Wybór następnego przeskoku dokonywany jest na bazie tablic trasowania (zwanych także tablicami forwardowania) i procedury dopasowania zgodnie z regułą „najdłuższego pasującego prefiksu". W najprostszym przypadku tablica taka zawiera tylko jedną pozycję, określającą domyślną trasę dla wszystkich forwardowanych pakietów.

Na bazie zestawu specjalnych protokołów sygnalizacyjnych oraz zabezpieczających zaprojektowano i zrealizowano odmianę protokołu IP przeznaczoną dla urządzeń mobilnych, zwaną mobilnym IP. Z mobilnym węzłem skojarzone są dwa adresy: domowy, wywodzący się z jego macierzystej sieci, oraz adres przekierowania, wywodzący się z sieci, w obrębie której węzeł ten aktualnie przebywa. W wersji podstawowej komunikacja węzła mobilnego z innymi węzłami prowadzi zawsze przez agenta domowego, rezydującego w macierzystej sieci tegoż węzła; ze względu na być może znaczne ich oddalenie

komunikacja ta może przybrać wariant bardzo nieefektywny, w związku z czym opracowano (w wersji MIPv6) opcję jej usprawnienia, zwaną popularnie optymalizacją trasy: rola agenta domowego ogranicza się wówczas do ustanowienia skojarzenia węzła mobilnego z węzłem-korespondentem, w ramach którego dalsza komunikacja odbywa się już w sposób bezpośredni.

Kolejnym niebanalnym zagadnieniem związanym z protokołem IP jest sposób przetwarzania datagramów przez hosty, a raczej stopień rygoryzmu kontroli związanej z tym przetwarzaniem. Rygoryzm ten odzwierciedlony jest przez dwa tzw. modele hosta — silny i słaby; ogólnie rzecz biorąc, w warunkach modelu silnego kwestia legalności adresu IP powiązana jest ściśle z interfejsami, przez które wysyłane są i odbierane datagramy, natomiast model słaby jest w tej kwestii znacznie bardziej liberalny, co jednak czyni host mniej odpornym na ewentualne ataki. Ponadto w sytuacji, gdy z danym hostem związanych jest kilka adresów IP (co w IPv6 jest sytuacją normalną, a IPv4 również jest możliwe), powstaje problem odpowiedniej strategii wyboru adresu źródłowego i adresu docelowego zapisywanych w nagłówku datagramu. Strategia ta była oczywista w czasach, gdy host łączony był z Internetem za pośrednictwem pojedynczego interfejsu i opatrzony pojedynczym adresem IP, dziś w obliczu hostów *multihomed* (czyli hostów połączonych z kilkoma dostawcami za pośrednictwem różnych interfejsów) odpowiednie albo kiepskie zaprojektowanie tej strategii przekłada się w konsekwencji na efektywne albo kiepskie trasowanie. Standardowy ciąg reguł składający się na domyślną postać tej strategii preferuje adresy trwałe, o ograniczonym zakresie, kosztem adresów ogólnych i tymczasowych.

Na koniec zajęliśmy się problematyką ataków internetowych, możliwych do realizacji za pośrednictwem mechanizmów protokołu IP. Większość z tych ataków sprowadza się do podrabiania lub fałszowania adresów IP w nagłówkach i opcjach, z fatalną często konsekwencją dla poprawności trasowania datagramów; intruzi mogą też wykorzystywać („eksploatować") rozmaite błędy i luki tkwiące w konkretnych implementacjach protokołu. Obecnie większość routerów ignoruje opcje specyfikowane w datagramach (a routery brzegowe często usuwają je z datagramów na styku z Internetem). Chociaż podrabianie adresów wciąż jest potencjalnym źródłem zagrożeń, jego konsekwencje są w dużej części niwelowane przez rozmaite mechanizmy filtrowania, m.in. filtrowanie wprowadzające.

5.9. Bibliografia

[AN] *http://www.iana.org/assignments/protocol-numbers*

[AUTOVON] http://en.wikipedia.org/wiki/Autovon

[DC05] J. Doyle, J. Carroll, *Routing TCP/IP, Volume 1, Second Edition* (Cisco Press, 2005).

[DSCPREG] *http://www.iana.org/assignments/dscp-registry/dscp-registry.xml*

[H05] G. Huston, *Just How Big Is IPv6? — or Where Did All Those Addresses Go?0*, „The ISP Column", lipiec 2005, *http://www.potaroo.net/papers/isoc/2005-07/ipv6size.html*

[IP6PARAM] *http://www.iana.org/assignments/ipv6-parameters*

[IPPARAM] *http://www.iana.org/assignments/ip-parameters*

[IV] *http://www.iana.org/assignments/version-numbers*

[LFS07] J. Leguay, T. Friedman, K. Salamatian, *Describing and Simulating Internet Routes*, „Computer Networks", 51(8), czerwiec 2007.

[MB97] L. McKnight, J. Bailey(red.), *Internet Economics* (MIT Press, 1997).

[MP] *http://www.iana.org/assignments/mobility-parameters*

[P90] C. Pinter, *A Book of Abstract Algebra, Second Edition* (Dover, 2010; reprint wydania z 1990 roku).

[PB61] W. Peterson, D. Brown, *Cyclic Codes for Error Detection*, Proc. IRE, 49(228), styczeń 1961.

[RC05] S. Raab, M. Chandra, *Mobile IP Technology and Applications* (Cisco Press, 2005).

[RFC0791] J. Postel, *Internet Protocol*, Internet RFC 0791/STD 0005, wrzesień 1981.

[RFC1108] S. Kent, *U.S. Department of Defense Security Options for the Internet Protocol*, Internet RFC 1108 (historical), listopad 1991.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1385] Z. Wang, *EIP: The Extended Internet Protocol*, Internet RFC 1385 (informational), listopad 1992.

[RFC1393] G. Malkin, *Traceroute Using an IP Option*, Internet RFC 1393 (experimental), styczeń 1993.

[RFC1858] G. Ziemba, D. Reed, P. Traina, *Security Consideration for IP Fragment Filtering*, Internet RFC 1858 (informational), październik 1995.

[RFC2113] D. Katz, *IP Router Alert Option*, Internet RFC 2113, luty 1997.

[RFC2460] S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6)*, Internet RFC 2460, grudzień 1998.

[RFC2473] A. Conta, S. Deering, *Generic Packet Tunneling in IPv6 Specification*, Internet RFC 2473, grudzień 1998.

[RFC2474] K. Nichols, S. Blake, F. Baker, D. Black, *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*, Internet RFC 2474, grudzień 1998.

[RFC2475] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services*, Internet RFC 2475 (informational), grudzień 1998.

[RFC2597] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, *Assured Forwarding PHB Group*, Internet RFC 2597, czerwiec 1999.

[RFC2675] D. Borman, S. Deering, R. Hinden, *IPv6 Jumbograms*, Internet RFC 2675, sierpień 1999.

[RFC2711] C. Partridge, A. Jackson, *IPv6 Router Alert Option*, Internet RFC 2711, październik 1999.

[RFC2827] P. Ferguson, D. Senie, *Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*, Internet RFC 2827/BCP 0038, maj 2000.

[RFC3031] E. Rosen, A. Viswanathan, R. Callon, *Multiprotocol Label Switching Architecture*, Internet RFC 3031, styczeń 2001.

[RFC3128] I. Miller, *Protection Against a Variant of the Tiny Fragment Attack*, Internet RFC 3128 (informational), czerwiec 2001.

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, Internet RFC 3168, wrzesień 2001.

[RFC3246] B. Davie, A. Charny, J. C. R. Bennett, K. Benson, J. Y. Le Boudec, W.Courtney, S. Davari, V. Firoiu, D. Stiliadis, *An Expedited Forwarding PHB (Per-Hop Behavior)*, Internet RFC 3246, marzec 2002.

[RFC3260] D. Grossman, *New Terminology and Clarifications for Diffserv*, Internet RFC 3260 (informational), kwiecień 2002.

[RFC3484] R. Draves, *Default Address Selection for Internet Protocol Version 6 (IPv6)*", Internet RFC 3484, luty 2003.

[RFC3484-revise] A. Matsumoto, J. Kato, T. Fujisaki, T. Chown, *Update to RFC 3484 Default Address Selection for IPv6*, Internet draft-ietf-6man-rfc3484-revise (w przygotowaniu), lipiec 2011. *http://tools.ietf.org/agenda/79/slides/6man-6.pdf*

[RFC3704] F. Baker, P. Savola, *Ingress Filtering for Multihomed Hosts*, Internet RFC 3704/BCP 0084, maj 2004.

[RFC3963] V. Devarapalli, R. Wakikawa, A. Petrescu, P. Thubert, *Network Mobility (NEMO) Basic Support Protocol*, Internet RFC 3963, styczeń 2005.

[RFC4193] R. Hinden, B. Haberman, *Unique Local IPv6 Unicast Addresses*, Internet RFC 4193, październik 2005.

[RFC4213] E. Nordmark, R. Gilligan, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, Internet RFC 4213, październik 2005.

[RFC4225] P. Nikander, J. Arkko, T. Aura, G. Montenegro, E. Nordmark, *Mobile IP Version 6 Route Optimization Security Design Background*, Internet RFC 4225 (informational), grudzień 2005.

[RFC4594] J. Babiarz, K. Chan, F. Baker, *Configuration Guidelines for Diffserv Service Classes*, Internet RFC 4594 (informational), sierpień 2006.

[RFC4782] S. Floyd, M. Allman, A. Jain, P. Sarolahti, *Quick-Start for TCP and IP*, Internet RFC 4782 (experimental), styczeń 2007.

[RFC4866] J. Arkko, C. Vogt, W. Haddad, *Enhanced Route Optimization for Mobile IPv6*, Internet RFC 4866, maj 2007.

[RFC4950] R. Bonica, D. Gan, D. Tappan, C. Pignataro, *ICMP Extensions for Multiprotocol Label Switching*, Internet RFC 4950, sierpień 2007.

[RFC5095] J. Abley, P. Savola, G. Neville-Neil, *Deprecation of Type 0 Routing Headers in IPv6*, Internet RFC 5095, grudzień 2007.

[RFC5096] V. Devarapalli, *Mobile IPv6 Experimental Messages*, Internet RFC 5094, grudzień 2007.

[RFC5142] B. Haley, V. Devarapalli, H. Deng, J. Kempf, *Mobility Header Home Agent Switch Message*, Internet RFC 5142, styczeń 2008.

[RFC5213] S. Gundavelli (red.), K. Leung, V. Devarapalli, K. Chowdhury, B.Patil, *Proxy Mobile IPv6*, Internet RFC 5213, sierpień 2008.

[RFC5220] A. Matsumoto, T. Fujisaki, R. Hiromi, K. Kanayama, *Problem Statement for Default Address Selection in Multi-Prefix Environments: Operational Issues of RFC 3484 Default Rules*, Internet RFC 5220 (informational), lipiec 2008.

[RFC5350] J. Manner, A. McDonald, *IANA Considerations for the IPv4 and IPv6 Router Alert Options*, Internet RFC 5350, wrzesień 2008.

[RFC5380] H. Soliman, C. Castelluccia, K. ElMalki, L. Bellier, *Hierarchical Mobile IPv6 (HMIPv6) Mobility Management*, Internet RFC 5380, październik 2008.

[RFC5568] R. Koodli (red.), *Mobile IPv6 Fast Handovers*, Internet RFC 5568, lipiec 2009.

[RFC5570] M. StJohns, R. Atkinson, G. Thomas, *Common Architecture Label IPv6 Security Option (CALIPSO)*, Internet RFC 5570 (informational), lipiec 2009.

[RFC5865] F. Baker, J. Polk, M. Dolly, *A Differentiated Services Code Point (DSCP) for Capacity-Admitted Traffic*, Internet RFC 5865, maj 2010.

[RFC5944] C. Perkins (red.), *IP Mobility Support for IPv4, Revised*, Internet RFC 5944, listopad 2010.

[RFC6178] D. Smith, J. Mullooly, W. Jaeger, T. Scholl, *Label Edge Router Forwarding of IPv4 Option Packets*, Internet RFC 6178, marzec 2011.

[RFC6275] C. Perkins (red.), D. Johnson, J. Arkko, *Mobility Support in IPv6*, Internet RFC 6275, czerwiec 2011.

[RFC6301] Z. Zhu, R. Rakikawa, L. Zhang, *A Survey of Mobility Support in the Internet*, Internet RFC 6301 (informational), lipiec 2011.

[RTAOPTS] *http://www.iana.org/assignments/ipv6-routeralert-values*

[THL06] N. Thompson, G. He, H. Luo, *Flow Scheduling for End-Host Multihoming*, Proc. IEEE INFOCOM, kwiecień 2006.

[TWEF03] J. Touch, Y. Wang, L. Eggert, G. Flinn, *A Virtual Internet Architecture*, Proc. ACM SIGCOMM Future Directions in Network Architecture Workshop, marzec 2003.

[W03] T. Wu, *Network Neutrality, Broadband Discrimination*, „Journal of Telecommunications and High Technology Law", 2, 2003 (zrewidowane w 2005).

Rozdział 6. Konfigurowanie systemu: DHCP i autokonfiguracja

6.1. Wprowadzenie

Implementacja protokołów TCP/IP w każdym hoście lub routerze wymaga do swego działania pewnego minimum informacji konfiguracyjnej. Informacja ta jest niezbędna do tego, by możliwe było przypisywanie systemom lokalnych nazw, przydzielanie interfejsom adresów IP (lub innych identyfikatorów) czy też korzystanie z innych ważnych usług sieciowych, takich jak usługa nazw mnemonicznych DNS (*Domain Name System*) czy pośrednictwo agenta domowego w mobilnym IP. Przez wiele lat używano różnych sposobów realizacji tego zadania, wszystkie one dają się jednak podzielić na trzy kategorie: „ręczne" dostarczanie niezbędnej informacji, uzyskiwanie jej za pomocą systemu wykorzystującego usługi sieciowe oraz stosowanie odpowiednich algorytmów do zapewnienia jej w sposób automatyczny. Zobaczymy, jak każda z tych kategorii wykorzystywana jest w kontekście IPv4 i IPv6; zrozumienie mechanizmów konfigurowania systemów jest o tyle istotne, że z danymi konfiguracyjnymi ma do czynienia każdy administrator sieci i (do pewnego stopnia) każdy jej użytkownik.

Jak pamiętamy z rozdziału 2., każdy interfejs sieciowy wykorzystywany przez protokoły grupy TCP/IP wymaga trzech elementów: adresu IP, maski podsieci i (dla IPv4) adresu rozgłoszeniowego (*broadcast*) — ten ostatni konstruowany jest zasadniczo na bazie dwóch poprzednich elementów. Jest to absolutne minimum dla komunikacji z innymi systemami w tej samej podsieci; komunikacja przekraczająca granice podsieci, zwana *dostarczaniem pośrednim* (pisaliśmy o nim w rozdziale 5.), wymaga ponadto tablicy trasowania, umożliwiającej wybór odpowiednich routerów dla poszczególnych grup adresu docelowego pakietów. By możliwe było praktyczne funkcjonowanie pewnych usług sieciowych, takich jak WWW czy poczta elektroniczna, mało wygodne dla użytkownika adresy IP zastępowane są przez usługę DNS (patrz rozdział 11.) bardziej przyjaznymi nazwami mnemonicznymi. Jako że DNS jest usługą rozproszoną, każdy wykorzystujący ją system musi posiadać dostęp do przynajmniej jednego jej serwera. Wszystkie wymienione elementy — własny adres IP, maska podsieci, adres IP routera, adres IP serwera DNS — składają się na informację podstawową, niezbędną do funkcjonowania hosta w kontekście Internetu z jego podstawowymi usługami w rodzaju WWW czy e-mail; usługi bardziej zaawansowane wymagają dodatkowych informacji, np. węzeł mobilny (MN)

działający w ramach mobilnego IP wymaga znajomości adresu IP swego agenta domowego. W tym rozdziale skoncentrujemy się na protokołach i procedurach wykorzystywanych do dostarczania wymienionych informacji na potrzeby hosta będącego klientem Internetu, czyli na *protokole dynamicznego konfigurowania hosta* (DHCP — *Dynamic Host Configuration Protocol*) i *bezstanowej autokonfiguracji adresów* (*stateless address autoconfiguration*) w obu wersjach IPv4 i IPv6. Pokażemy także, jak niektórzy dostawcy Internetu przeprowadzają konfigurację systemów klienckich, używając PPP w połączeniu z Ethernetem.

Automatyczne konfigurowanie częściej staje się udziałem hostów niż routerów czy serwerów, które najczęściej konfigurowane są ręcznie lub przy użyciu dedykowanych narzędzi GUI — tak dzieje się z kilku praktycznych przyczyn. Po pierwsze, hosty klienckie zwykle częściej zmieniają swą lokalizację w sieci, a każda taka zmiana staje się mniej uciążliwa właśnie dzięki automatyzacji w zakresie dostosowywania parametrów konfiguracji do nowych warunków. Po drugie, serwery i routery to urządzenia, od których oczekuje się nieprzerwanej dostępności i względnej niezależności, tak więc uniezależnienie ich konfiguracji od innych usług sieciowych sprzyja większej ich niezawodności (i lepszemu do niej zaufaniu). Po trzecie, w typowej sieci korporacyjnej komputery klienckie są zwykle bardziej liczne niż serwery i routery, a typowi użytkownicy są mniej doświadczeni w zakresie infrastruktury sieciowej niż administratorzy; w tej sytuacji zapewnienie scentralizowanej usługi kontroli konfiguracji klienckich sprawia, że zarządzanie tą konfiguracją jest wygodniejsze i mniej podatne na pomyłki.

Zależnie od wykorzystywanych lub świadczonych usług, routery i hosty mogą wymagać informacji wykraczających poza wymieniony zakres podstawowy, m.in. informacji o lokalizacjach agenta domowego, routerów multicast, bram VPN i bram protokołu SIP (*Session Initiation Protocols* — protokół inicjowania sesji). Niektóre z tych usług zawierają standardowe mechanizmy i protokoły ułatwiające pozyskiwanie niezbędnych informacji konfiguracyjnych, inne zdają się wyłącznie na interwencję użytkownika lub administratora.

6.2. Dynamic Host Configuration Protocol (DHCP)

Protokół DHCP, opisywany w dokumencie [RFC2131], jest popularnym protokołem typu klient-serwer, wykorzystywanym do przypisywania hostom (rzadziej routerom) informacji konfiguracyjnych. DHCP jest powszechnie używany zarówno w sieciach korporacyjnych, jak i prostych sieciach domowych — nawet najprostsze routery posiadają wbudowane serwery DHCP. Moduły klienckie DHCP stanowią część wszystkich współczesnych systemów operacyjnych, wbudowywane są także w urządzenia, takie jak drukarki sieciowe i telefony VoIP. DHCP został oryginalnie zaprojektowany do użytku z wersją IPv4, więc w kontekście tej właśnie wersji omawiać będziemy poszczególne jego cechy i związki z IP — chyba że wyraźnie wskażemy, o którą wersję chodzi w danym przypadku. W punkcie 6.2.5 opiszemy wersję DHCPv6 przeznaczoną specjalnie dla IPv6, definiowaną w dokumencie [RFC3315]; niezależnie od niej protokół IPv6 posiada własne procesy automatyzujące pozyskiwanie informacji konfiguracyjnej — w rozwiązaniach hybrydowych są one wykorzystywane równolegle z DHCPv6.

Projekt DHCP wywodzi się w prostej linii z wcześniejszego protokołu o nazwie *Internet Bootstrap Protocol* (BOOTP), opisywanego w [RFC0951] i [RFC1542] i obecnie już nieużywanego. BOOTP dostarczał klientowi elementarne informacje konfiguracyjne jednorazowo, bez związku z faktem, że po pewnym czasie mogą się one stać nieaktualne. DHCP zmienia ten aspekt funkcjonalności BOOTP poprzez koncepcję *dzierżawy* (*lease*) informacji (patrz [GC89]), ponadto zakres dostarczanej przez DHCP informacji jest większy niż w przypadku BOOTP. Dzierżawa informacji to ograniczenie jej okresu ważności do pewnego uzgodnionego z klientem interwału czasowego; w czasie tego interwału (lub po jego upłynięciu) klient może zwracać się do serwera z żądaniem *odnowienia* (*renew*) dzierżawy — okres ważności informacji zaczyna się liczyć na nowo od momentu jej odnowienia. Protokoły DHCP i BOOTP są kompatybilne: serwery DHCP mogą współpracować z klientami przystosowanymi wyłącznie do BOOTP i vice versa — klienty DHCP mogą współpracować z serwerami BOOTP. Dane obu protokołów przenoszone są w otoczkach datagramów UDP, enkapsulowanych w datagramach IP; oba protokoły wykorzystują port 67. po stronie serwerowej i 68. po stronie klienckiej.

Na działanie DHCP składają się dwie zasadnicze funkcje: zarządzanie adresami i dostarczanie danych konfiguracyjnych. Zarządzanie adresami polega na kontrolowanym przydzielaniu (wydzierżawianiu) klientom adresów IP z dostępnej dla serwera puli, zaś dostarczanie klientom informacji odbywa się za pomocą komunikatów w specyficznym formacie, we współpracy z maszyną stanów protokołu. Serwery DHCP, zależnie od sposobu skonfigurowania, mogą wykonywać alokację adresów w trzech trybach: automatycznym, dynamicznym i manualnym. W trybie alokacji *dynamicznej* adres, pobierany z dostępnej *puli* (będącej zwykle zakresem adresów), przydzielany jest klientowi na podstawie jego (klienta) identyfikatora; po wygaśnięciu dzierżawy adres ten może zostać unieważniony. Alokacja *automatyczna* także wykonywana jest na podstawie identyfikatora klienta i adres także pobierany jest z dostępnej puli, lecz nie podlega unieważnieniu. Alokacja *manualna* to przydzielenie klientowi adresu dla niego ustalonego, niepobieranego z puli serwera — jest to zachowanie odziedziczone po protokole BOOTP. Dalej ograniczymy się do alokacji dynamicznej jako najczęściej stosowanej i najbardziej interesującej.

6.2.1. Pule i dzierżawienie adresów

W trybie alokacji dynamicznej klient DHCP żąda od serwera przydzielenia adresu IP, na co serwer odpowiada zwróceniem adresu wybranego z puli dostępnych adresów — pula ta jest najczęściej ciągłym zbiorem (zakresem) adresów przeznaczonych specjalnie na potrzeby DHCP. Przydział ten ma formę *dzierżawy* — adres zostaje przydzielony do użytku jedynie przez pewien okres, zwany okresem *trwania dzierżawy* (*lease duration*). Klient może odnowić dzierżawę, żądając przedłużenia okresu ważności przydzielonego wcześniej adresu (w niektórych konfiguracjach żądanie takiego przedłużenia nie zawsze jest honorowane).

Okres dzierżawy jest ważnym parametrem konfiguracyjnym serwera DHCP. Może on mieć różną wartość — od kilku minut do kilku dni (teoretycznie możliwa jest dzierżawa wieczysta, czyli z nieskończonym okresem, ma ona jednak sens tylko w małych sieciach). Określenie tej wartości jest kwestią kompromisu między wieloma czynnikami: spodziewaną liczbą klientów, pojemnością puli adresów i oczekiwanym stopniem stabilności adresów. Dłuższy okres dzierżawy oznacza większe prawdopodobieństwo wyczerpania

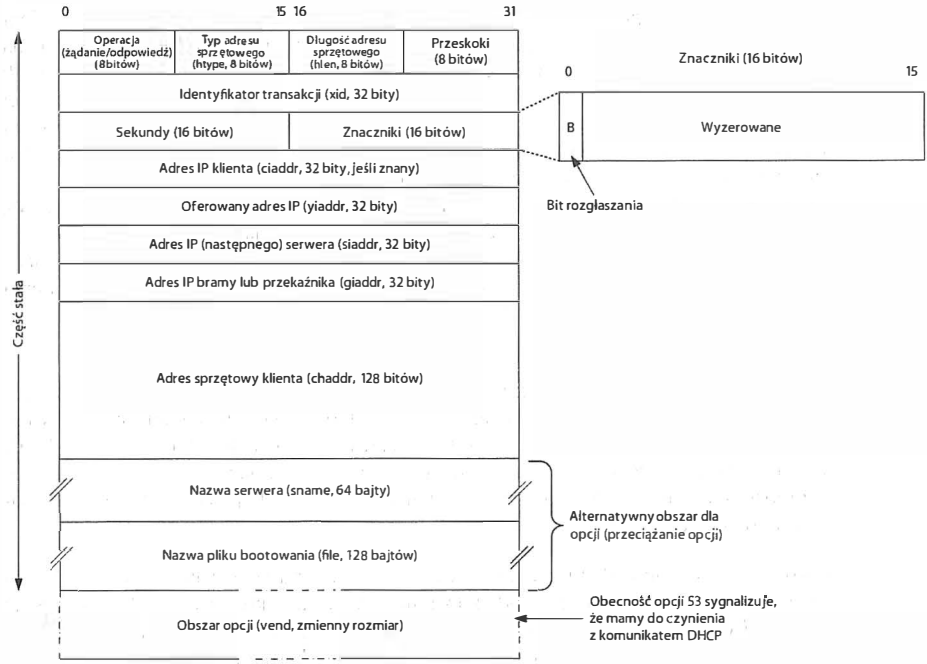
dostępnej puli adresów, lecz jednocześnie większą stabilność adresów i mniejszy ruch sieciowy spowodowany rzadszymi żądaniami odnowy. Krótsze dzierżawy to bardziej efektywne wykorzystywanie dostępnej puli adresów, lecz jednocześnie większa ich rotacja i związane z nią większe obciążenie sieci. Najczęściej wartości domyślne plasują się w granicach 12 – 24 godzin, choć produkty Microsoftu zalecają w tej mierze 8 dni dla małych sieci i 16 – 24 dni dla większych. Oprócz okresu dzierżawy definiuje się także dwa graniczne znaczniki czasowe, oznaczane w dokumentacji *T1* i *T2*, związane z jej odnawianiem: pierwszy z nich wyznacza dolną granicę, powyżej której klient *ma prawo* żądać odnowienia dzierżawy, drugi natomiast jest górną granicą, powyżej której klient *musi* takie żądanie sformułować. Ich domyślne wartości to (odpowiednio) $\frac{1}{2}$ i $\frac{7}{8}$ okresu dzierżawy, administrator może je dowolnie zmieniać. Powrócimy do tej kwestii w punkcie 6.2.4, przy omawianiu operacji protokołu DHCP.

Klient wysyłający do serwera DHCP żądanie ma możliwość dostarczenia w jego ramach rozmaitych informacji, obejmujących m.in. nazwę klienta, żądany okres dzierżawy, kopię adresu aktualnie dzierżawionego lub ostatnio używanego i wiele innych parametrów. Serwer łączy te informacje z innymi uzyskanymi samodzielnie (jak adres MAC klienta, aktualna data i czas, itp.) i na tej podstawie przydziela wybrany adres oraz inne informacje dodatkowe. Udzielenie dzierżawy przez serwer zapisywane jest w jego pamięci trwałej (pliku dyskowym) — awaria lub restart serwera nie mogą przecież powodować utraty informacji o aktualnie obowiązujących dzierżawach.

6.2.2. Format komunikatów DHCP i BOOTP

Wspominaliśmy już, że protokół DHCP stanowi rozszerzenie protokołu BOOTP i jest z nim kompatybilny; kompatybilność ta przejawia się m.in. na poziomie wymiany komunikatów. Klienty przystosowane do korzystania z serwerów BOOTP mogą korzystać także z serwerów DHCP, a serwery BOOTP mogą świadczyć usługi na rzecz klientów DHCP (nawet w sieciach niezawierających serwerów DHCP), za pośrednictwem *agentów przekazywania* (*relay agents*), zwanych często (po prostu) *przekaźnikami*, o których piszemy w punkcie 6.2.6. Komunikat DHCP, pod względem formatu (niemal) identyczny z komunikatem BOOTP, rozpoczyna się od ustalonej części początkowej, po której następuje obszar zmiennej długości (patrz rysunek 6.1). Format ten definiowany jest w kilku dokumentach RFC ([RFC0951], [RFC1542] i [RFC2131]).

Pole *Operacja* służy do rozróżnienia typów komunikatu: wartość 1 oznacza żądanie, wartość 2 — odpowiedź. Wartość pola *Typ adresu sprzętowego*, odzwierciedlającego typ adresu sprzętowego klienta, ustalana jest na podstawie danych udostępnionych przez protokół ARP (któremu poświęciliśmy rozdział 4.); najczęściej pole to zawiera wartość 1 reprezentującą Ethernet (wykaz wszystkich zdefiniowanych wartości znajduje się pod adresem [IARP]). Pole *Długość adresu sprzętowego* zawiera długość adresu sprzętowego w bajtach — dla Ethernetu jest to — oczywiście — 6. Pole *Przeskoki* służy do zliczania przekaźników, przez które przechodzi komunikat na swej drodze: nadawca wpisuje w to pole wartość 0, każdy kolejny przeskok zwiększa tę wartość o 1. Ponieważ w sieci może być w danej chwili realizowanych wiele transakcji DHCP, konieczne jest opatrzenie każdej transakcji unikatowym identyfikatorem warunkującym prawidłowe kojarzenie żądań z odpowiedziami: rolę tę spełnia pole *Identyfikator transakcji*, którego wartość ustalana jest przez nadawcę jako 32-bitowa liczba (pseudo)losowa.



Rysunek 6.1. *Format komunikatu BOOTP, rozszerzony do postaci komunikatu DHCP za pomocą zmiennego pola opcji. Znaczenie pól części stałej zdefiniowane jest w dokumentach [RFC0951], [RFC1542] i [RFC2131]. Dzięki zgodności w części stałej przekaźniki BOOTP mogą przetwarzać komunikaty DHCP, a klienty BOOTP mogą korzystać z serwerów DHCP. Pola Nazwa serwera i Nazwa pliku bootowania mogą być użyte jako alternatywny obszar dla opcji w sytuacji, gdy pożądane jest minimalizowanie rozmiaru komunikatu*

W pole *Sekundy* wpisuje klient liczbę sekund, które upłynęły od momentu wysłania pierwszego żądania przydziału lub odnowienia adresu. W polu *Znaczniki* interpretowany jest obecnie tylko jeden bit, zwany *bitem rozgłaszania*: klient może ustawić ten bit, jeśli nie chce (lub nie potrafi) przetwarzać datagramów opatrzonych adresem unicast (bo np. nie posiada jeszcze przydzielonego adresu unicast), natomiast zdolny jest do przetwarzania datagramów rozgłoszeniowych. Dla serwera i przekaźników jest to sygnał, że odpowiedź na żądanie musi zostać opatrzona adresem broadcast.

W systemach Windows korzystanie z bitu rozgłaszania napotyka na pewne trudności. Klienty w wersjach Windows XP i Windows 7 zerują ten bit, jest on jednak ustawiany przez klienty pracujące pod systemem Windows Vista. W efekcie klienty z Windows Vista sprawiać mogą problemy we współpracy z serwerami nieprzetwarzającymi bitu rozgłaszania w sposób prawidłowy, mimo iż implementacja DHCP w Windows Vista zgodna jest z cytowanymi wcześniej dokumentami RFC. Czytelników zainteresowanych tą tematyką odsyłamy do stosownego artykułu w bazie wiedzy Microsoftu ([MKB928233]).

Kolejne pola zawierają rozmaite adresy IP. I tak pole *Adres IP klienta* zawiera obecny adres klienta (nadawcy żądania) lub wartość 0, gdy adres ten jest nieznany; w pole *Oferowany adres IP* wpisuje serwer adres stanowiący odpowiedź na żądanie klienta, a w pole

Adres IP następnego serwera — adres następnego serwera, którego użyć ma klient w procesie bootowania (np. w sytuacji, gdy obraz systemu operacyjnego znajduje się na serwerze innym niż serwer DHCP, do którego klient kieruje żądanie). W polu *Adres IP bramy lub przekaźnika* serwery DHCP i przekaźniki BOOTP wpisują adres (najczęściej swój własny) routera używanego do forwardowania komunikatów DHCP. W polu *Adres sprzętowy klienta* znajduje się unikatowy, sprzętowy identyfikator kliencki, którego znaczenie i długość wynikają z wymienionych wcześniej pól *Typ adresu sprzętowego* i *Długość adresu sprzętowego*. Serwer może wykorzystywać to pole w różny sposób, np. do zapewnienia identycznych odpowiedzi na kolejne żądania przychodzące od tego samego klienta. Tradycyjnie w polu tym umieszczany jest adres MAC klienta, posiadający wystarczające cechy unikatowości, nowszą tendencją jest jednak identyfikowanie klienta w sposób niezależny od jego cech sprzętowych — o związanej z tym opcji piszemy w punktach 6.2.3 i 6.2.4.

Pozostałe pola części stałej zawierają zasadniczo *Nazwę serwera* i *Nazwę pliku bootowania*. Są to łańcuchy ASCIIZ, czyli ciągi znaków ASCII zakończone ogranicznikiem w postaci zerowego bajta (niezaliczanego do treści łańcucha), charakterystyczne dla języka C. W sytuacji ograniczonej wielkości dostępnego miejsca dla danych wykorzystanie tych pól może być jednak inne — mogą one zawierać niektóre opcje, co szczegółowo wyjaśniamy w punkcie 6.2.3.

Pozostała część komunikatu miała początkowo (czyli w ramach protokołu BOOTP) także ustaloną długość i zawierała dodatkową informację charakterystyczną dla producenta (*Vendor Extensions*). Protokół DHCP nadaje temu polu nowe znaczenie: staje się ono *polem opcji* i może mieć różną długość w poszczególnych komunikatach.

6.2.3. Opcje DHCP i BOOTP

Ze względu na zachowanie zgodności formatu komunikatów BOOTP i DHCP, nowe pola wykorzystywane przez DHCP, a nieobecne w BOOTP, znalazły swe miejsce jako opcje w końcowym obszarze o zmiennej długości (patrz ostatni akapit w poprzednim punkcie). Obszar każdej opcji rozpoczyna się od jej bajta identyfikacyjnego. Dla niektórych opcji rozmiar ich danych jest ustalony i wynika wprost z bajta identyfikacyjnego, więc dane te następują bezpośrednio za bajtem identyfikacyjnym; w szczególnym przypadku opcja składa się z samego bajta identyfikacyjnego. Inne opcje posługują się danymi o zmiennym rozmiarze, który tym samym musi być jawnie wskazany; jest więc zapisywany w bajcie następującym bezpośrednio po bajcie identyfikacyjnym i poprzedzającym dane (rozmiar ten uwzględnia tylko dane, bez bajta identyfikacyjnego i bajta długości).

Protokół DHCP posługuje się względnie dużą liczbą opcji, niektóre z nich rozpoznawane są także przez protokół BOOTP. Ich kompletna lista dostępna jest pod adresem [IBDP], pierwsze 77 — wśród nich najczęściej używane — definiowanych jest w dokumencie [RFC2132]. Należą do nich m.in. opcje reprezentujące: wypełnienie (*Pad* — 0), maskę podsieci (*Subnet Mask* — 1), adres routera (*Router Address* — 3), serwer DNS (*Domain Name Server* — 6), nazwę domeny (*Domain Name* — 15), żądany adres IP (*Requested IP Address* —50), okres dzierżawy adresu (*Address Lease Time* — 51), typ komunikatu DHCP (*DHCP Message Type* — 53), identyfikator serwera (*Server Identifier* — 54), listę parametrów żądania (*Parameter Request List* — 55), komunikat DHCP o błędzie (*DHCP Error Message* — 56), czas odnowienia dzierżawy (*Lease Renewal*

Time — 58), czas ponownego wiązania (*Lease Rebinding Time* — 59), identyfikator klienta (*Client Identifier* — 61), listę przeszukiwania sufiksów domeny (*Domain Search List* — 119) i znacznik końca opcji (*End* — 255).

Opcja *DHCP Message Type* (53) składa się wyłącznie z bajta identyfikacyjnego, którego wartość identyfikuje następująco typ komunikatu:

- 1 — DHCPDISCOVER
- 2 — DHCPOFFER
- 3 — DHCPREQUEST
- 4 — DHCPDECLINE
- 5 — DHCPACK
- 6 — DHCPNAK
- 7 — DHCPRELEASE
- 8 — DHCPINFORM
- 9 — DHCPFORCERENEW (patrz [RFC3203])
- 10 — DHCPLEASEQUERY
- 11 — DHCPLEASEUNASSIGNED
- 12 — DHCPLEASEUNKNOWN
- 13 — DHCPLEASEACTIVE

Znaczenie czterech ostatnich pozycji definiowane jest w dokumencie [RFC4388]. Standardowo opcje zajmują końcowy obszar komunikatu, ale — jak wcześniej wspominaliśmy — można je umieszczać także w polach przeznaczonych oryginalnie na nazwę serwera i nazwę pliku bootowania; nazywa się to *przeciążaniem opcji* (*option overloading*) i sygnalizowane jest przez opcję o identyfikatorze 52 (OVERLOADOPTION). Zauważmy ponadto, że zapisywanie długości danych opcji w pojedynczym bajcie ogranicza tę długość do 255 bajtów; gdy konieczne jest użycie dłuższych danych, odnośną opcję specyfikuje się *wielokrotnie*. Dane poszczególnych wystąpień są wówczas konkatenowane w kolejności wystąpienia, a wynik konkatenacji traktowany jako pojedyncza encja. Jeżeli wiele instancji tej samej opcji znajduje się w różnych lokalizacjach — obszarze opcji, polu *Nazwa serwera* i (lub) polu *Nazwa pliku bootowania* — to w każdym z wymienionych miejsc instancje konkatenowane są niezależnie, po czym trzy wyniki cząstkowych konkatenacji konkatenowane są ze sobą w następującej kolejności: najpierw ten z obszaru opcji, potem ten z pola *Nazwa pliku bootowania*, na końcu ten z pola *Nazwa serwera*.

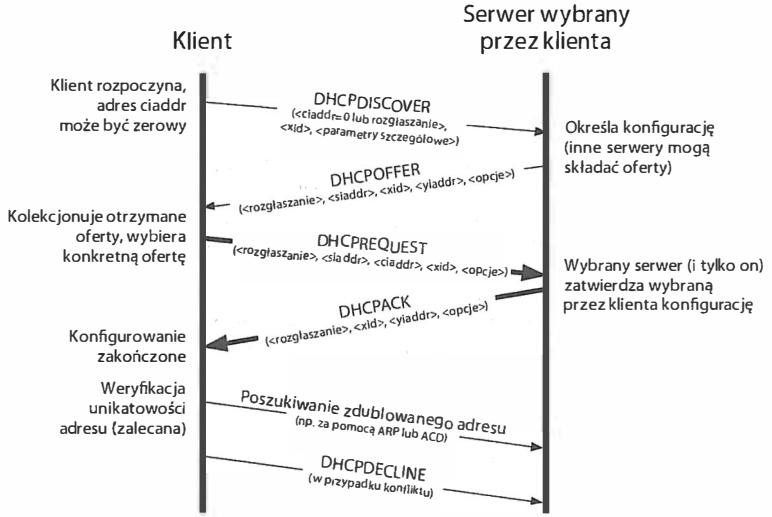
Treścią opcji mogą być elementarne informacje konfiguracyjne, ale także informacje wspierające współpracę z innymi protokołami. Przykładowo w dokumencie [RFC2132] znajduje się wykaz opcji związanych z tradycyjną konfiguracją węzłów TCP/IP, obejmującą m.in. informacje adresowe, lokalizacje serwerów, ustawienia wskaźników i początkową wartość pola *Czas życia* w wysyłanych datagramach IP. Kolejne specyfikacje związane są z konfiguracją NetWare ([RFC2241] i [RFC2242]), klasyfikacją użytkowników ([RFC3004]), FQDN[1] (RFC4702), serwerami ISNS (*Internet Storage Name Service*)

[1] Skrót od *Fully Qualified Domain Name*, oznaczający pełną nazwę domeny — *przyp. tłum.*

[RFC4174], kontrolerami usług broadcast i multicast (BCMS) stosowanymi w telefonii komórkowej 3G ([RFC4280]), strefami czasowymi ([[RFC4833]), autokonfiguracją ([RFC2563]), wyborem podsieci ([RFC3011]), wyborem usługi nazw (patrz rozdział 11. i [RFC2937] oraz protokołem PANA (*Protocol for Carrying Authentication for Network Access* — patrz rozdział 18. i [RFC5192]).Opcje te zaprojektowano z myślą o możliwości obsługi innych protokołów i funkcji (opisywanych w dalszym ciągu tego rozdziału, począwszy od punktu 6.2.7.

6.2.4. Operacje protokołu DHCP

Komunikaty DHCP to w istocie komunikaty BOOTP uzupełnione o zbiór opcji. Gdy nowy klient przyłącza się do sieci, rozpoczyna od wyszukania dostępnych serwerów DHCP i zapoznania się z proponowanymi przez te serwery adresami. Następnie decyduje się na wybór konkretnego serwera i konkretnego adresu z oferowanego przez ten serwer zakresu; jeżeli tylko serwer ten nie zakończył w międzyczasie pracy, a wspomniany adres nadal jest dostępny, serwer akceptuje ów wybór, wysyłając stosowny komunikat potwierdzający. O swej decyzji klient informuje jednocześnie pozostałe serwery DHCP. Chronologię wymiany komunikatów między klientem i serwerem w ramach opisanego procesu przedstawiamy schematycznie na rysunku 6.2.



Rysunek 6.2. *Typowy scenariusz wymiany komunikatów między klientem a serwerem DHCP. Klient wykrywa istniejące w sieci serwery DHCP i zaprasza je do składania ofert (komunikat DHCPDISCOVER). Serwery składają swe oferty (komunikaty DHCPOFFER), klient wybiera jedną z nich i żąda jej potwierdzenia (DHCPREQUEST). Spójność między komunikatami należącymi do tej samej transakcji zapewnia identyfikator transakcji zawarty w polu xid, odnośny serwer identyfikowany jest w polu siaddr. Adres będący przedmiotem żądania znajduje się w polu ciaddr. Jeśli klient posiada już przydzielony adres i chciałby tylko odnowić jego dzierżawę, powyższy scenariusz staje się uboższy o początkowe komunikaty DHCPDISCOVER i DHCPOFFER*

Klient żądający usługi umieszcza w polu *Operacja* wartość BOOTREQUEST, zaś w pierwsze cztery bajty obszaru opcji wpisuje „magiczne ciasteczko", definiowane w [RFC2132], stanowiące ciąg czterech bajtów o wartościach 99, 130, 83 i 99 (dziesiętnie).

Komunikaty przesyłane przez klienta do serwera DHCP enkapsulowane są w datagramach UDP/IP. W polu *Operacja* znajduje się wartość BOOTREQUEST, opcja *DHCP Message Type* zawiera natomiast typ komunikatu (zazwyczaj DHCPDISCOVER lub DHCPREQUEST). W pierwszych czterech bajtach obszaru opcji znajduje się wspomniane wcześniej „ciasteczko". Komunikat wysyłany jest spod adresu źródłowego 0.0.0.0 na porcie 68 do adresu docelowego 255.255.255.255 (czyli adresu lokalnego rozgłaszania) na port 67. Komunikaty wędrujące od serwera do klienta wysyłane są przez serwer z jego adresem IP jako źródłowym, z portu 67; adresem docelowym jest 255.255.255.255, portem docelowym jest — oczywiście — port 68 (szczegóły funkcjonowania protokołu UDP opisujemy w rozdziale 10.).

W typowym scenariuszu wymiany komunikatów klient wysyła do sieci komunikat DHCPDISCOVER. Każdy z serwerów DHCP, który odbierze ten komunikat — bezpośrednio bądź za pośrednictwem przekaźników — może odpowiedzieć komunikatem DHCPOFFER, zawierającym oferowany adres IP w polu *Oferowany adres IP* oraz zwykle kilka informacji towarzyszących, np. adres IP serwera DNS, maskę podsieci itp. Elementem komunikatu DHCPOFFER jest także proponowany *okres dzierżawy* (T), określający maksymalny czas ważności oferowanego adresu IP, o ile ten nie zostanie wcześniej odnowiony. Z okresem tym wiążą się dwie wartości pokrewne: *czas odnowienia* (T1) to okres czasu, po upływie którego klient może zażądać odnowienia dzierżawy (od serwera, z którego ją uzyskał), oraz czas *ponownego wiązania* (*rebinding time*) (T2), po upływie którego klient może żądać ponownego przydziału adresu od dowolnego serwera DHCP. Domyślnie przyjmuje się wartości $T1 = \frac{T}{2}$ oraz $T2 = \frac{7 \cdot T}{8}$.

Klient po odebraniu jednego lub więcej komunikatów DHCPOFFER od jednego lub kilku serwerów decyduje się na wybór konkretnej oferty i sygnalizuje ten fakt wysłaniem — na adres rozgłoszeniowy — komunikatu DHCPREQUEST zawierającego identyfikację serwera i żądany adres IP, pod postacią opcji (odpowiednio) *Server Identifier* i *Requested IP Address*[2]. Spośród serwerów DHCP, które ten komunikat odbiorą, tylko ten określony w polu identyfikacyjnym dokonuje zapisu transakcji w swej pamięci stałej, inne serwery „zapominają" o żądaniu klienta; jednocześnie serwer ten potwierdza klientowi zaakceptowanie jego żądania, wysyłając komunikat DHCPACK — odtąd klient staje się pełnoprawnym użytkownikiem przydzielonego adresu na okres jego dzierżawy; gdyby jednak z pewnych względów ostateczne żądanie klienta nie mogło zostać zrealizowane (bo żądany adres został w międzyczasie przydzielony albo stał się z innych powodów nieaktualny), serwer wysłałby do klienta komunikat DHCPNAK, oznaczający odmowę.

Odpowiedź DHCPACK z serwera DHCP nie musi jednak wcale oznaczać końca zadania, ponieważ przydzielony klientowi adres IP może już istnieć w sieci, bo np. pojawił się w niej z pominięciem mechanizmów DHCP. Dla wykluczenia takiego niebezpieczeństwa

[2] W dalszym ciągu, dla uproszczenia, określenie „klient wysyła komunikat do serwera *X*" oznaczać będzie to samo, co „klient wysyła komunikat rozgłoszeniowy zawierający identyfikator serwera *X* w polu *Server Identifier* — *przyp. tłum.*

klient powinien więc przeprowadzić procedurę ACD, którą opisywaliśmy w rozdziale 4., a w przypadku stwierdzenia zdublowania adresu powinien zrezygnować z otrzymanej dzierżawy, wysyłając komunikat DHCPDECLINE do serwera, z którego ją uzyskał. Po odczekaniu (standardowo zalecanych) 10 sekund klient może ponowić opisaną procedurę uzyskiwania dzierżawy adresu.

Klient, który zdecyduje się na rezygnację z dzierżawy przed upływem okresu jej ważności, może ów zamiar wyrazić, wysyłając do odnośnego serwera komunikat DHCPRELEASE.

W sytuacji gdy klient posiada już przydzielony adres IP i chciałby tylko odnowić jego dzierżawę, w powyższym scenariuszu pominąć można dwa początkowe komunikaty DHCPDISCOVER i DHCPOFFER, poczynając od komunikatu DHCPREQUEST zawierającego wspomniany adres. Tak jak poprzednio, serwer może żądanie klienta zaakceptować (DHCPACK) lub odrzucić (DHCPNAK).

Podobna do powyższej jest sytuacja, gdy klient posiada już przydzielony adres IP, nie żąda jego przedłużenia, lecz chciałby uzyskać pewne informacje uzupełniające (niewiążące się z przydziałem adresu). Klient sygnalizuje ten zamiar, wysyłając do serwera komunikat DHCPINFORM (zamiast DHCPREQUEST), i otrzymuje żądane informacje w treści komunikatu DHCPACK.

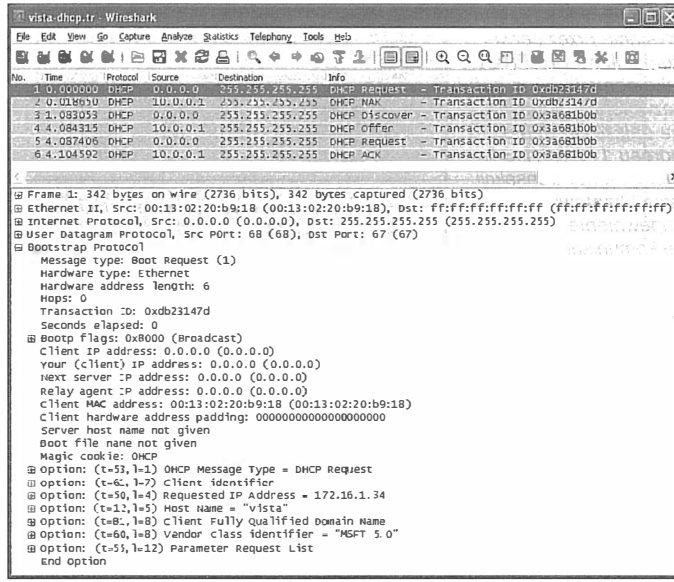
6.2.4.1. Przykład

Zobaczmy teraz, co kryje się pod podszewką DHCP, czyli prześledźmy szczegółowo opisane jego funkcjonowanie na przykładzie komunikacji laptopa z systemem Windows Vista (jako klienta) z linuksowym serwerem DHCP (w przypadku klienta sterowanego Windows 7 sytuacja tylko nieznacznie różni się od prezentowanej). Klient był poprzednio skojarzony z siecią bezprzewodową i wykorzystywał adres IP o prefiksie niedostępnym w nowej sieci, z którą właśnie został skojarzony. Pamiętając poprzedni adres, próbuje go zachować do dalszego użytku, wysyłając komunikat DHCPREQUEST, widoczny w oknie programu Wireshark na rysunku 6.3.

W dokumentach [RFC4436] i [RFC6059] opisywana jest procedura *wykrywania skojarzenia z siecią* (*Detecting Network Attachment*, w skrócie DNA), odpowiednio dla wersji IPv4 i IPv6. Dokumenty te nie definiują nowych protokołów, lecz sugerują wykorzystywanie protokołu ARP (dla IPv4) lub kombinacji komunikatów *Neighbor Solicitation* i *Router Discovery* (dla IPv6 — patrz rozdział 8.) w sposób redukujący opóźnienie związane z uzyskiwaniem informacji konfiguracyjnych w związku z przełączaniem hosta między sieciami. Ponieważ są to mechanizmy stosunkowo nowe (szczególnie w wersji dla IPv6), nie we wszystkich systemach są implementowane.

Na rysunku 6.3 widzimy żądanie DHCP przesyłane za pomocą ramki rozgłoszeniowej warstwy łącza danych (z docelowym adresem sprzętowym ff:ff:ff:ff:ff:ff); źródłowym adresem IP jest adres nieokreślony (0.0.0.0), adresem docelowym — adres lokalnego rozgłaszania 255.255.255.255. Klient nie ma w tym momencie wielkiego wyboru — nie znając prefiksu nowej sieci ani nie wiedząc, czy będzie mógł korzystać z dotychczasowego adresu, nie ma praktycznie alternatywy dla podanych adresów. Na poziomie warstwy transportowej komunikat ma postać datagramu UDP wysyłanego z portu klienckiego 68 (bootpc) do portu serwerowego 67 (bootps); ponieważ protokół DHCP jest rozszerzeniem protokołu BOOTP, w programie Wireshark widzimy odwołania do tego

Rysunek 6.3. *Klient przemieszcza się do innej sieci i próbując zachować dotychczasowy adres IP — 172.16.1.34 — wysyła komunikat DHCPREQUEST do serwera DHCP*



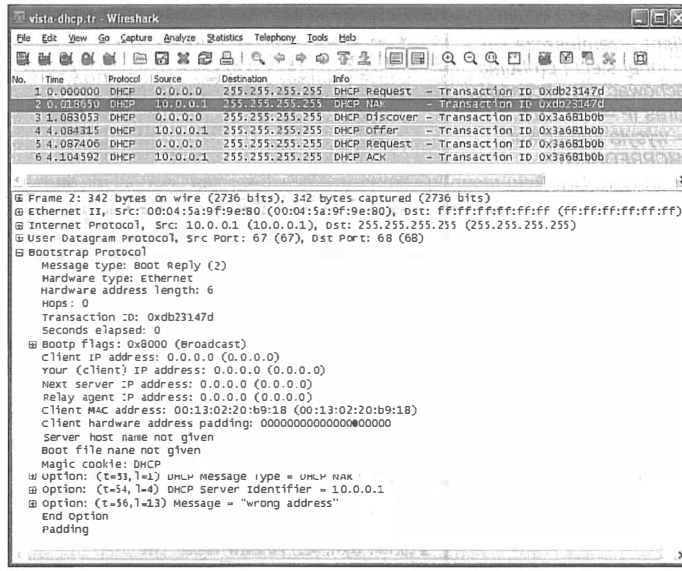
ostatniego w postaci nazwy protokołu (`Bootstrap Protocol`) oraz identyfikatora typu komunikatu (`BOOTREQUEST` (1)). Raportowany adres sprzętowy jest adresem ethernetowym (typ 1 i długość 6). Losowo wybrany unikatowy identyfikator transakcji równy jest `0xdb23147d`. Ustawiony jest bit rozgłoszeniowy w bajcie znaczników, wskutek czego odpowiedzi serwera będą wysyłane na adres rozgłoszeniowy. W jednej z opcji widzimy adres dotychczas używany przez klienta `172.16.1.34` — dokładniejszą analizę opcji DHCP rozpoczniemy w punkcie 6.2.9.

Jeden z pobliskich serwerów DHCP odbiera komunikat `DHCPREQUEST` zawierający adres `172.16.1.34`, adres ten jednak nie może zostać przydzielony w ramach bieżącej sieci — serwer odrzuca żądanie, odpowiadając komunikatem `DHCPNAK` (patrz rysunek 6.4).

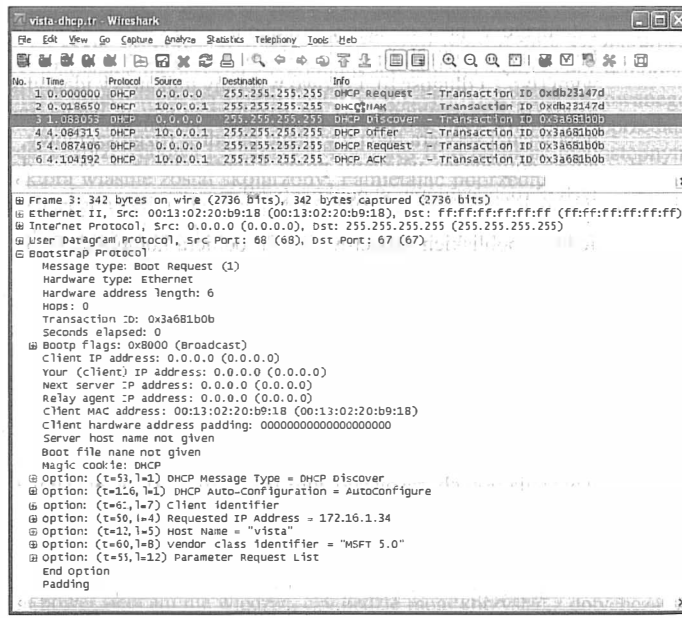
Jak widać na rysunku, komunikat `DHCPNAK` wysyłany jest przez serwer na adres rozgłoszeniowy (`Broadcast`). Typ komunikatu wyczytać możemy z opcji *Message Type* (`t=53`), następna opcja (`t=54`) wskazuje `10.0.0.1` jako identyfikator serwera, zaś przyczynę odmowy (`wrong address`) odczytać możemy z kolejnej opcji (`t=56`). Komunikat zawiera także kopię adresu sprzętowego klienta (tu: adresu MAC). Dla klienta oznacza to niemożność używania dotychczasowego adresu `172.16.1.34`, w związku z czym rozpoczyna on opisaną wcześniej procedurę, wysyłając komunikat `DHCPDISCOVER` (patrz rysunek 6.5).

Komunikat `DHCPDISCOVER`, podobnie jak komunikat `DHCPREQUEST`, zawiera adres dotychczas używany przez klienta (na typ etapie klient nie żąda jeszcze nowego adresu), bogatsza jest jednak lista opcji, pojawił się też nowy identyfikator transakcji `0x3a681b0b`. Większość z pozostałych oryginalnych pól protokołu BOOTP ma wartość 0, wyjątkiem jest pole *Adres sprzętowy klienta* zawierające adres MAC klienta. Zgodnie z oczekiwaniami,

Rysunek 6.4. *Wysłanie przez serwer komunikatu DHCPNAK jako odmowy zezwolenia klientowi na dalsze używanie adresu 172.16.1.34; identyfikator transakcji daje klientowi zapewnienie, że komunikat ten istotnie jest odpowiedzią na jego żądanie*

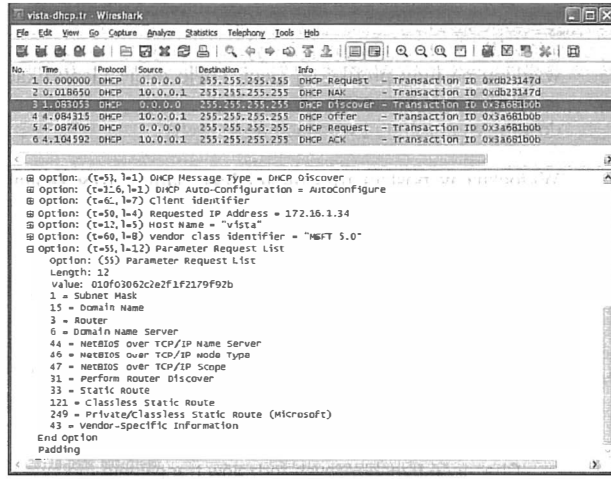


Rysunek 6.5. *Wysłanie przez klienta komunikatu DHCPDISCOVER jako pierwszy etap procedury pozyskiwania nowego adresu IP*



adres ten figuruje jako źródłowy w ramce ethernetowej, ponieważ pakiet nie jest forwardowany przez przekaźniki. Reszta komunikatu DHCPDISCOVER obejmuje 7 opcji, z których jedna widoczna jest w postaci rozwiniętej na rysunku 6.6 — pokazane są jej podopcje.

Rysunek 6.6. *Opcja parametrów żądania (Parameter Request List) określa szczegółowo elementy informacji konfiguracyjnej wymaganej przez klienta*



Na rysunku 6.6 widoczne są poszczególne opcje komunikatu `DHCPDISCOVER`. Druga z opcji (`t=116`) sygnalizuje żądanie przez klienta informacji na temat możliwości autokonfiguracji adresów (temat ten omawiamy dokładniej w podrozdziale 6.3): jeśli niemożliwe będzie przydzielenie klientowi adresu przez serwer DHCP, dostępność autokonfiguracji oznacza zezwolenie ze strony serwera na uzyskanie przez klienta adresu IP we własnym zakresie.

Opcja *Client Identifier* (`t=61`) zawiera identyfikator, jakim klient przedstawia się serwerom DHCP (identyfikator ten ma postać `01001302208918`, niewidoczną na rysunku 6.6, ponieważ opcja nie jest rozwinięta). Większość współczesnych systemów zezwala klientom na używanie identyfikatorów wybranych samodzielnie, co jednak stwarza ryzyko kolizji, czyli wyboru tego samego identyfikatora przez różne klienty; z tego względu generalnie zalecane jest pozostanie przy tradycyjnej konwencji, czyli używanie identyfikatorów klienckich generowanych automatycznie przez system. Taka automatyczna generacja opiera się przeważnie na adresie sprzętowym klienta, przykładowo w systemach linii Windows identyfikator klienta powstaje przez poprzedzenie jego adresu MAC jednoznakowym wyróżnikiem typu adresu — dla Ethernetu wyróżnik ten jest bajtem o wartości `x01`.

Zapotrzebowanie na stabilne identyfikatory klientów doprowadziło do opracowania koncepcji identyfikatorów niezależnych od fizycznych własności klienta — np. od adresu MAC, który choć zmienia się raczej rzadko, to jednak generalnie zmieniać się może. Poza tym dla klienta posiadającego kilka interfejsów (kart) sieciowych serwer DHCP generować może różne identyfikatory, zależnie od interfejsu, którego adres MAC obierze sobie za podstawę tego generowania. W dokumencie [RFC4361] opisana jest metoda generowania unikalnych identyfikatorów dla klientów IPv4, oparta na analogicznym schemacie oryginalnie zaprojektowanym dla IPv6. Metoda ta zakłada użycie (w ramach DHCPv4) identyfikatorów DUID (*DHCP Unique Identifiers*) w połączeniu z identyfikatorami IAID (*Identity Association Identifier*), które definiowane są (dla DHCPv6) w dokumencie [RFC3315] i o których piszemy w podpunktach 6.2.5.3 oraz 6.2.5.4. Metoda ta sprawia jednocześnie, że pole *Adres sprzętowy klienta* w komunikatach DHCP staje się polem nieistotnym (przestarzałym — *deprecetad*). Obecnie jednak opisywany schemat nie jest jeszcze szeroko rozpowszechniony.

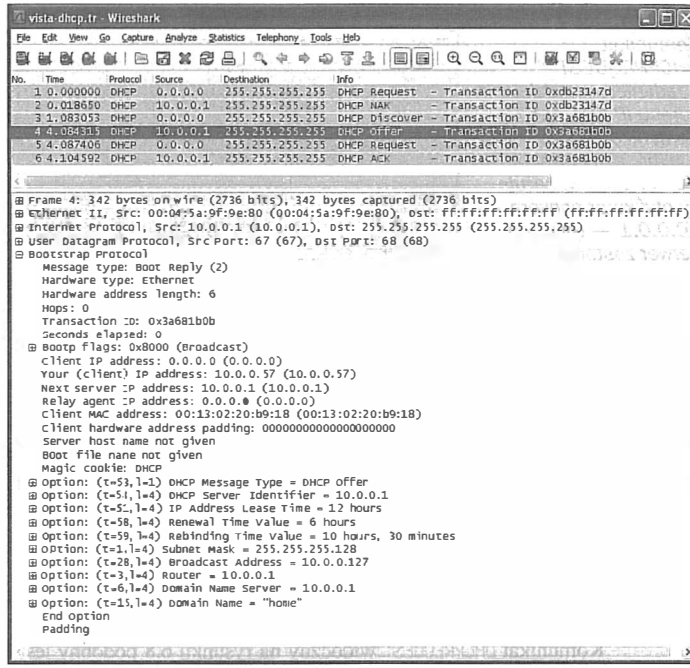
W ramach opcji *Requested IP Address* (t=50) widzimy używany dotychczas przez klienta adres IP 172.16.1.34; jak już wyjaśnialiśmy, w nowej sieci adres ten jest niedostępny ze względu na inny prefiks tej sieci. Dwie opcje identyfikacyjne (t=12 i t=60) zawierają nazwę hosta (vista) i tzw. identyfikator klasy producenta (MSFT 5.0 — oznacza on Windows w wersji 2000 i wersjach nowszych).

Widoczna w postaci rozwiniętej lista parametrów żądania (opcja *Parameter Request List*, t=55) precyzuje zestaw informacji konfiguracyjnych żądanych przez klienta. Lista ta stanowi w istocie ciąg bajtów, z których każdy zawiera numer wymaganej opcji; w żądaniu z rysunku 6.6 są to konwencjonalne parametry internetowe (maska podsieci, nazwa domeny, adres IP serwera DNS, adres IP domyślnego routera) oraz specyficzne dla produktów Microsoftu parametry związane z NetBIOS-em. Parametr 31 oznacza zainteresowanie klienta funkcją ICMP wykrywania routera (szczegóły w rozdziale 8.), trzy następne parametry dotyczą statycznych pozycji w tablicy forwardowania (o tablicach forwardowania pisaliśmy w rozdziale 5.), a dokładniej — pozycji reprezentujących poszczególne rodzaje trasowania.

Obecność trzech różnych rodzajów trasowania ma swe historyczne źródło w ewolucji adresów IP. W epoce adresowania klasowego, przed pełnym rozpowszechnieniem masek podsieci (patrz rozdział 2.), porcja adresu określająca miejsce sieciowe zdeterminowana była bezpośrednio przez początkowe bity adresu określające jego klasę — parametr 33 reprezentuje trasowanie oparte na tej właśnie konwencji. Gdy pojawiło się adresowanie bezklasowe, każdemu adresowi IPv4 towarzyszyła nieodłącznie maska podsieci wyznaczająca podział na adres podsieci i numer hosta w tej podsieci (ponownie patrz rozdział 2.) — z tym rodzajem adresowania, definiowanym w [RFC3442], związany jest parametr 121. Wariant adresowania specyficzny dla Microsoftu (parametr 249) podobny jest do adresowania bezklasowego.

Ostatni parametr — 43 — reprezentuje elementy informacji konfiguracyjnej, charakterystyczne dla konkretnego producenta. Generalnie informacja ta interpretowana jest w kontekście identyfikatora klasy producenta (t=60) — parametr 43 sam z siebie nie obejmuje identyfikacji producenta. Utrudnia to włączanie do pojedynczego komunikatu DHCP informacji pochodzących od kilku producentów — mimo iż opcja t=60 może wystąpić w komunikacie wielokrotnie, trudno byłoby kojarzyć poszczególne jej wystąpienia z konkretnymi instancjami parametru 43. W dokumencie [RFC3925] opisano mechanizm wprowadzający jednoznaczność tego kojarzenia, w oparciu o opcję t=124, wykorzystującą tzw. numery przedsiębiorstw (*enterprise numbers*) przydzielane centralnie przez IANA. Informacja specyficzna dla Microsoftu obejmuje — oprócz wspomnianych już opcji NetBIOS-u — sposób przetwarzania metryki (preferencji) domyślnej trasy w tablicach forwardowania oraz traktowanie (zachowywanie albo zwalnianie) dzierżawy adresu przy zamykaniu systemu. Jest ona także wykorzystywana przez system *ochrony dostępu do sieci* (NAP — *Network Address Protection*, patrz [MS-DHCPN]). Systemy Mac OS wykorzystują informację specyficzną dla firmy Apple w usłudze bootowania z sieci (*NetBoot service*) oraz protokole BSDP (*Boot Server Discovery Protocol* — patrz [F07]).

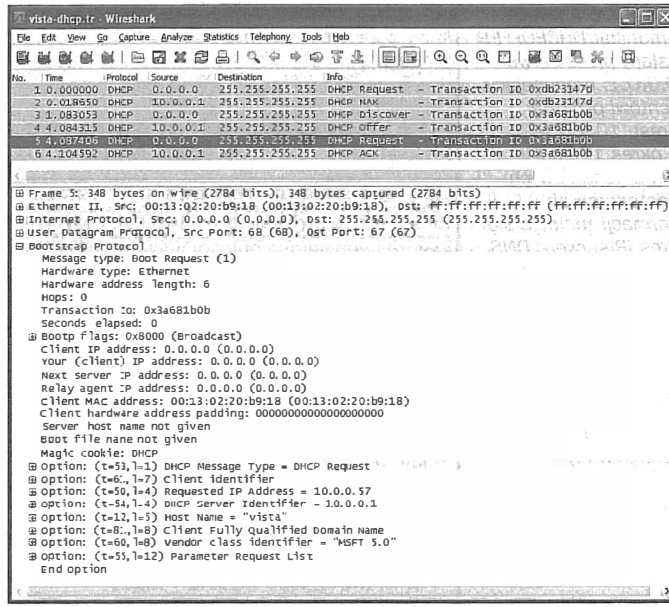
W odpowiedzi na odebrany komunikat DHCPDISCOVER serwer DHCP odpowiada komunikatem DHCPOFFER, zawierającym proponowany adres IP, okres jego dzierżawy i dodatkowe informacje konfiguracyjne. Widok z rysunku 6.7 uzyskano w instalacji z jednym serwerem DHCP, rezydującym na jednym komputerze pełniącym także rolę routera i serwera DNS.



Rysunek 6.7. *Komunikat DHCPOFFER wysłany przez serwer 10.0.0.1, proponujący klientowi przydział adresu 10.0.0.57 na okres 12 godzin. Wśród dodatkowych udostępnionych informacji znajduje się adres IP serwera DNS, nazwa domeny, adres IP domyślnego routera, maska podsieci i adres rozgłaszania. W opisywanym przykładzie serwer DHCP, serwer DNS i domyślny router rezydują na tym samym komputerze o adresie 10.0.0.1*

Na rysunku 6.7 ponownie widzimy znajome pola protokołu BOOTP i zestaw opcji DHCP związanych z zarządzaniem adresami. W polu *Operacja* znajduje się wartość 2 oznaczająca odpowiedź (BOOTREPLY), pole *Oferowany adres IP* zawiera proponowany przez serwer adres 10.0.0.57 — zwróćmy uwagę, że jest to adres *różny* od adresu 172.16.1.34 wnioskowanego przez klienta w komunikacie DHCPDISCOVER: prefiks 172.16/12 nie pasuje do nowej sieci. Na informację dodatkową składają się opcje obejmujące: adres IP serwera DHCP (10.0.0.1), oferowany okres dzierżawy (12 godzin), minimalny czas odnowienia dzierżawy T1 (6 godzin), czas ponownego wiązania T2 (10,5 godziny), maskę podsieci (255.255.255.128), lokalny adres rozgłoszeniowy (10.0.0.127), adresy IP serwera DNS i domyślnego routera (oba równe adresowi serwera DHCP — 10.0.0.1) oraz nazwę domeny (home — nazwa ta nie jest objęta żadnymi standardami, nie powinna być więc używana jako nazwa domeny poza sieciami prywatnymi). Sieć wykorzystywana w tym przykładzie jest siecią domową, której węzły opatrywane są nazwami według schematu <*nazwa maszyny*>.home.

Po otrzymaniu (jedynej możliwej) oferty DHCPOFFER klient ponownie formułuje pod adresem serwera żądanie przydziału adresu IP — tym razem proponowanego adresu 10.0.0.57. Szczegóły wysyłanego do serwera komunikatu DHCPREQUEST widoczne są na rysunku 6.8.



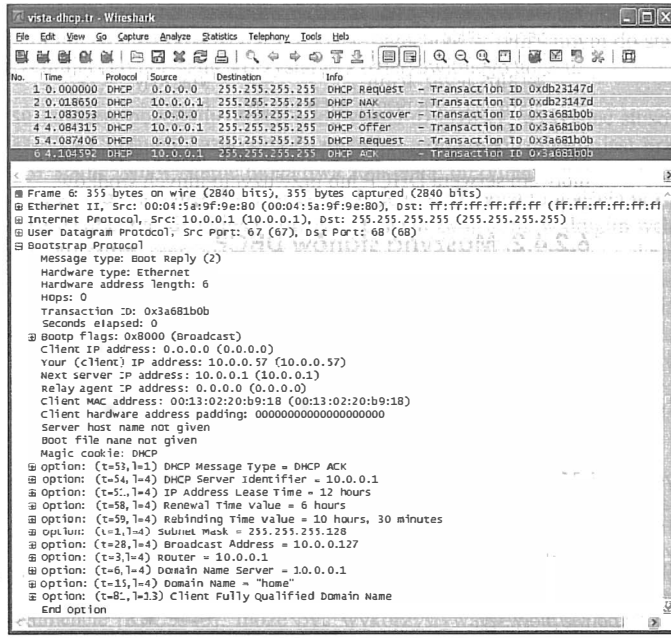
Rysunek 6.8. *Komunikat DHCPREQUEST z żądaniem przydzielenia adresu IP 10.0.0.57. Komunikat wysyłany jest w trybie rozgłaszania, jedna z opcji wskazuje identyfikator serwera 10.0.0.1 — tylko ten serwer zostanie zaangażowany w przydział adresu dla klienta, inne serwery zignorują otrzymany komunikat*

Komunikat DHCPREQUEST widoczny na rysunku 6.8 podobny jest do poprzednio wysłanego komunikatu DHCPDISCOVER, jednak z kilkoma istotnymi różnicami. Poza oczywistą różnicą w opcji *Message Type*, najbardziej widoczna jest różnica we wnioskowanym adresie, wskazany też jest konkretny identyfikator serwera (10.0.0.1). Ponadto, wobec zaproponowania przez serwer konkretnego adresu, klient zrezygnował z opcji autokonfigurowania. Zauważmy, że ten komunikat — podobnie jak wspomniany komunikat DHCPDISCOVER — jest komunikatem rozgłoszeniowym, dotrze więc do wszystkich węzłów (serwerów i klientów) w sieci; obecność konkretnego identyfikatora sprawia, że tylko jeden konkretny serwer DHCP zaangażowany zostanie w udzielenie dzierżawy adresu. Jak łatwo się domyślić, kolejnym wysyłanym komunikatem będzie odpowiedź serwera DHCPACK, szczegółowo pokazana na rysunku 6.9.

Komunikat DHCPACK podobny jest do poprzedniego komunikatu DHCPOFFER, zawiera jednak opcję FQDN (t=81), której treścią jest pełna kwalifikowana nazwa domeny vista.home (widoczna po rozwinięciu opcji). Teoretycznie komunikat ten jest przyzwoleniem dla klienta na używanie adresu IP 10.0.0.57 przez okres maksymalnie 12 godzin (tyle wynosi okres dzierżawy — patrz opcja t=51), klient jednak — w dobrze pojętym własnym interesie — powinien się jeszcze upewnić o unikatowości tego adresu, wykonując procedurę ACD (lub podobną) opisywaną w rozdziale 4.

Prezentowana w tym przykładzie wymiana komunikatów jest typowa dla bootowania systemu lub przyłączania hosta do nowej sieci. Możliwe jest także jej jawne zainicjowanie za pomocą odpowiednich poleceń (po ewentualnym zwolnieniu już wykorzystywanej dzierżawy). W systemie Windows zwolnienie dzierżawy osiąga się za pomocą polecenia:

```
C:\> ipconfig /release
```



Rysunek 6.9. *Komunikat DHCPACK jako przyzwolenie dla klienta na używanie przydzielonego adresu IP 10.0.0.57 przez okres maksymalnie 12 godzin*

zaś opisywaną procedurę pozyskiwania adresu (lub jego odnawiania) uruchamia się poleceniem:

```
C:\> ipconfig /renew
```

W Linuksie analogiczny efekt uzyskać można za pomocą poleceń (odpowiednio):

```
Linux# dhclient -r
```

oraz

```
Linux# dhclient
```

Zestaw pozyskanych z serwera DHCP informacji konfiguracyjnych dla lokalnego systemu obejrzeć można w systemie Windows za pomocą odmiany polecenia `ipconfig`:

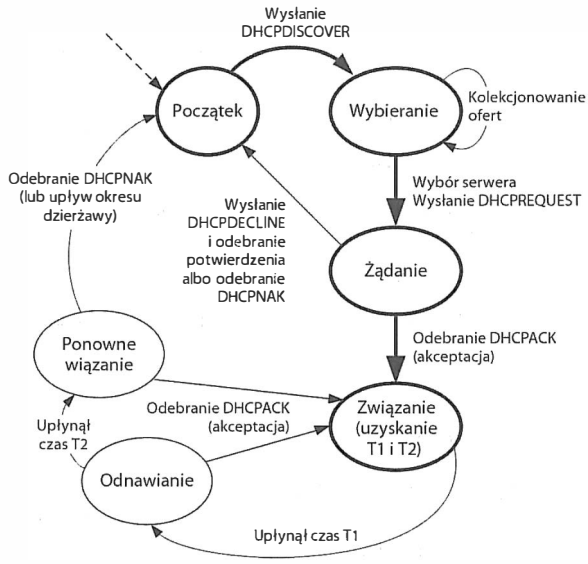
```
C:\> ipconfig /all
...
Karta bezprzewodowej sieci LAN Połączenie sieci bezprzewodowej:
   Sufiks DNS konkretnego połączenia .: home
   Opis. . . . . . . . . . . . . . . : Intel(R) PRO/Wireless 3945ABG
                                       Network Connection
   Adres fizyczny. . . . . . . . . . : 00-13-02-20-B9-18
   DHCP włączone . . . . . . . . . . : Tak
   Autokonfiguracja włączona . . . . : Tak
   Adres IPv4. . . . . . . . . . . . : 10.0.0.57(Preferowane)
   Maska podsieci. . . . . . . . . . : 255.255.255.128
   Dzierżawa uzyskana. . . . . . . . : 21 grudnia 2008 23:31:48
   Dzierżawa wygasa. . . . . . . . . : 22 grudnia 2008 11:31:40
```

```
Brama domyślna. . . . . . . . . . . : 10.0.0.1
Serwer DHCP . . . . . . . . . . . . : 10.0.0.1
Serwery DNS . . . . . . . . . . . . : 10.0.0.1
NetBIOS przez Tcpip . . . . . . . . : Włączony
Lista przeszukiwania sufiksów DNS : home
```

W taki właśnie sposób obejrzeć można informacje o konfiguracji uzyskane za pośrednictwem serwera DHCP lub przy użyciu innych środków.

6.2.4.2. Maszyna stanów DHCP

Funkcjonowanie protokołu DHCP można postrzegać w kategoriach maszyny stanów (automatu skończonego). Na rysunku 6.10 widoczna jest maszyna stanu protokołu DHCP z perspektywy klienta. Każdy stan charakteryzuje się specyficznym zbiorem komunikatów, które protokół może wysłać i które spodziewa się otrzymać, przebywając w tym stanie. Zmiana stanu może być powodowana wysłaniem lub odebraniem komunikatu albo upłynięciem zadanego odcinka czasu.



Rysunek 6.10. *Maszyna stanu klienta DHCP. Stany i przejścia wyróżnione pogrubieniem są typowe dla klienta wnioskującego o dzierżawę adresu po raz pierwszy. Linią przerywaną wyróżniono wejście do stanu początkowego*

Maszyna stanu klienta DHCP rozpoczyna swój cykl życiowy od stanu `Początek`. Stan ten reprezentuje kompletną niewiedzę klienta w kwestii konfiguracji; klient wysyła komunikat rozgłoszeniowy `DHCPDISCOVER` i przechodzi tym samym do stanu `Wybieranie`. Pozostając w tym stanie, kolekcjonuje oferty serwerów DHCP (w postaci komunikatów `DHCPOFFER`), które odpowiedziały na jego komunikat `DHCPDISCOVER`. Gdy zdecyduje się na wybór konkretnej oferty, wysyła komunikat `DHCPREQUEST` (z identyfikatorem wybranego serwera) i przechodzi do stanu `Żądanie` reprezentującego oczekiwanie na odpowiedź wspomnianego serwera. Może on także otrzymywać potwierdzenia (`DHCPACK`) dotyczące adresów, na wybór których nie zdecydował się; jeśli nie zdecyduje się na wybór żadnego adresu, powraca do stanu `Początek`, rozpoczynając cykl od nowa. Bardziej

prawdopodobne jest jednak, że otrzyma potwierdzenie dla wybranego przez siebie adresu wraz z wartościami interwałów czasowych T1 i T2 — przechodząc tym samym do stanu Związanie, reprezentującego normalne korzystanie z dzierżawionego adresu.

Po upływie czasu T1 klient podejmuje próbę odnowienia dzierżawy, wysyłając stosowny komunikat DHCPREQUEST (niepokazany na rysunku) — co symbolizowane jest przez stan Odnawianie. W stanie tym wystąpić mogą dwa zdarzenia: otrzymanie potwierdzenia odnowienia dzierżawy albo upłynięcie czasu T2 (liczonego od momentu wejścia w stan Związanie); w pierwszym przypadku klient powraca do stanu Związanie, w drugim wchodzi w stan Ponowne wiązanie, reprezentujący żądanie nowego adresu, jeszcze w okresie obowiązywania aktualnej dzierżawy. Tu również sytuacja może się rozwinąć dwojako: otrzymanie potwierdzenia (DHCPACK) oznacza powrót do stanu Związanie, natomiast otrzymanie odpowiedzi przeczącej (DHCPNAK) lub nieotrzymanie jej przed upłynięciem okresu bieżącej dzierżawy oznacza klęskę w kwestii utrzymania ciągłości dzierżawy i przejście do stanu Początek; jeśli klient nie dysponuje alternatywnym połączeniem sieciowym lub innymi metodami uzyskania nowego adresu, musi się po prostu poddać, kończąc awaryjnie ewentualne trwające połączenie.

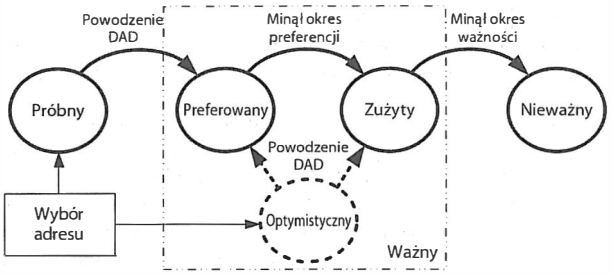
6.2.5. DHCPv6

Mimo iż obie wersje protokołu DHCP — dla IPv4 i dla IPv6 — koncepcyjnie służą podobnym celom, to już ich projekt i opcje wdrażania są wyraźnie różne. Protokół DHCPv6, opisywany w dokumencie [RFC3315], może być wykorzystywany w dwóch trybach: *stanowym* (*stateful*), w którym funkcjonuje podobnie do DHCPv4, oraz *bezstanowym* (*stateless*) wykorzystującym autokonfigurację adresów (o której piszemy w podrozdziale 6.3). Ogólnie rzecz biorąc, w trybie bezstanowym od klienta oczekuje się zdolności do samodzielnego skonfigurowania zestawu swych adresów IPv6, natomiast protokół DHCPv6 dostarcza mu jedynie dodatkowych informacji, np. adresu IP serwera DNS. Informację o lokalizacji serwera DNS można także uzyskiwać w inny sposób — za pomocą komunikatów ICMPv6 *Router Advertisement* (patrz rozdziały 8. i 11. oraz [RFC6106]).

6.2.5.1. Cykl życiowy adresu IPv6

Typowy host implementujący IPv6 posługuje się kilkoma adresami na każdym ze swych interfejsów. Każdy z tych adresów może być wykorzystywany jedynie w swym okresie ważności, który dzieli się na dwie fazy: preferencji i zużycia. Adres znajdujący się w stanie preferencyjnym dostępny jest do ogólnego użytku zarówno w roli adresu źródłowego, jak i docelowego. Gdy mija okres preferencji, adres staje się adresem *zużytym* (*deprecated*) i może być nadal używany na potrzeby trwających połączeń (np. połączeń TCP), lecz już nie do nawiązywania nowych. Kompletny cykl życiowy adresu IPv6 przedstawiono (w formie maszyny stanów) na rysunku 6.11.

Adres rozpoczyna swój cykl życiowy od stanu *próbnego* albo *optymistycznego*. W stanie próbnym trwa weryfikacja unikatowości adresu — procedura określana akronimem DAD (od *Duplicate Address Detection*), opisywana dokładniej w podpunkcie 6.3.2.1, sprawdza, czy któryś z węzłów tej samej sieci nie używa już tego adresu; sam adres może być na tym etapie stosowany jedynie na potrzeby protokołu komunikatów *Neighbor Discovery* (patrz rozdział 8.), nie może pełnić roli adresu źródłowego ani docelowego.

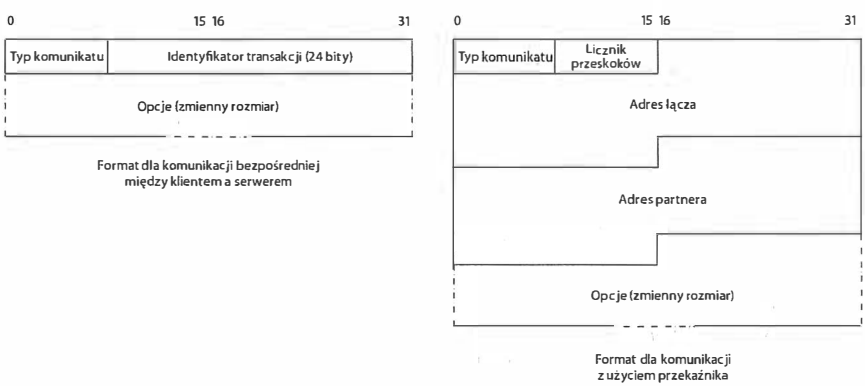


Rysunek 6.11. *Cykl życiowy adresu IPv6. Adres próbny poddawany jest weryfikacji przez procedurę DAD na unikatowość, po pomyślnej weryfikacji staje się adresem preferencyjnym i może być używany bez ograniczeń przez okres preferencji, po upływie którego staje się adresem zużytym. Adresy zużyte mogą być wykorzystywane do końca okresu ważności jedynie na potrzeby trwających już połączeń, lecz nie do nawiązywania nowych*

Alternatywą dla stanu próbnego jest stan *optymistyczny* (patrz [RFC4429]), w którym adres może być używany w ograniczonym zakresie, choć z mniejszymi rygorami (czyli z większym optymizmem) niż w stanie próbnym; faktycznie jest to więc nie tyle stan de facto, co po prostu użycie specjalnych („optymistycznych") reguł DAD. Optymistyczne używanie adresu jest pod wieloma względami podobne do traktowania go jak adresu zużytego, w szczególnym przypadku adres optymistyczny może być jednocześnie adresem zużytym, zależnie od jego okresów ważności i preferencji.

6.2.5.2. Format komunikatów DHCPv6

Komunikaty DHCPv6 enkapsulowane są w datagramach UDP/IPv6, z portami 546 (klienckim) i 547 (serwerowym) (o protokole UDP piszemy obszernie w rozdziale 10.). Adresem źródłowym zarówno dla serwerów, jak i przekaźników jest adres lokalny dla łącza. Komunikat DHCPv6 może występować w dwóch formatach: podstawowym, przeznaczonym dla komunikacji bezpośredniej między klientem a serwerem, i rozszerzonym, wykorzystywanym w komunikacji z użyciem przekaźnika. Oba formaty przedstawiono na rysunku 6.12.



Rysunek 6.12. *Dwa formaty komunikatu DHCPv6: podstawowy (po lewej) i rozszerzony (po prawej). W obu formatach najistotniejsze informacje zawarte są w obszarze opcji*

Zawartość pola *Adres łącza* (w formacie rozszerzonym) używana jest przez serwer do zidentyfikowania łącza, na którym zlokalizowany jest klient. Pole *Adres partnera* zawiera adres poprzedniego przeskoku na drodze komunikatu — czyli klienta lub przekaźnika, bo przekaźniki mogą być łączone w łańcuch. Szczegóły funkcjonowania przekaźników w obu wersjach protokołu — DHCPv4 i DHCPv6 — opisujemy w punkcie 6.2.6.

W obu formatach początkowy bajt identyfikuje typ komunikatu; w formacie podstawowym są to przeważnie tradycyjne żądania i odpowiedzi, podczas gdy format rozszerzony mają przede wszystkim komunikaty przekazywane do przekaźnika i z niego. W formacie rozszerzonym w obszarze opcji zawsze znajduje się opcja *Relay Messsage,* zawierająca kompletny komunikat przekazywany przez przekaźnik; oczywiście, w obu formatach mogą występować także inne opcje.

Jedna z zasadniczych różnic między DHCPv6 a DHCPv4 związana jest z adresowaniem multicastingu. Komunikaty rozgłoszeniowe klienta, przeznaczone dla serwerów i przekaźników DHCP, wysyłane są przez niego na adres `ff02::1:2` (zobacz znaczenie prefiksu `ff00::/8` w tabeli 2.8); adresem źródłowym jest adres lokalny łącza. Mimo iż formaty komunikatów DHCPv6 są całkowicie różne od formatu komunikatów BOOTP i DHCPv4, to jednak semantyka komunikatów jest w obu tych obszarach podobna. W tabeli 6.1 zestawiono listę komunikatów DHCPv6 wraz ze wskazaniem dokumentu definiującego każdy komunikat oraz (przybliżonego) odpowiednika z DHCPv4.

Tabela 6.1. *Typy i identyfikatory komunikatów DHCPv6 oraz równoważne im komunikaty DHCPv4*

Komunikat DHCPv6	Identyfikator w DHCPv6	Definiowany w dokumencie	Równoważny komunikat DHCPv4	Definiowany w dokumencie
SOLICIT	1	[RFC3315]	DHCPDISCOVER	[RFC2132]
ADVERTISE	2	[RFC3315]	DHCPOFFER	[RFC2132]
REQUEST	3	[RFC3315]	DHCPREQUEST	[RFC2132]
CONFIRM	4	[RFC3315]	DHCPREQUEST	[RFC2132]
RENEW	5	[RFC3315]	DHCPREQUEST	[RFC2132]
REBIND	6	[RFC3315]	DHCPDISCOVER	[RFC2132]
REPLY	7	[RFC3315]	DHCPACK/DHCPNAK	[RFC2132]
RELEASE	8	[RFC3315]	DHCPRELEASE	[RFC2132]
DECLINE	9	[RFC3315]	DHCPDECLINE	[RFC2132]
RECONFIGURE	10	[RFC3315]	DHCPFORCERENEW	[RFC3203]
INFORMATIONREQUEST	11	[RFC3315]	DHCPINFORM	[RFC2132]
RELAY-FORW	12	[RFC3315]	Nie istnieje	
RELAY-REPL	13	[RFC3315]	Nie istnieje	
LEASEQUERY	14	[RFC5007]	DHCPLEASEQUERY	[RFC4388]
LEASEQUERY-REPLY	15	[RFC5007]	DHCPLEASE{UNASSIGNED, UNKNOWN,ACTIVE}	[RFC4388]
LEASEQUERY-DONE	16	[RFC5460]	DHCPLEASEQUERYDONE	[ID4LQ]
LEASEQUERY-DATA	17	[RFC5460]	Nie istnieje	
			OHCPBULKLEASEQUERY	[ID4LQ]

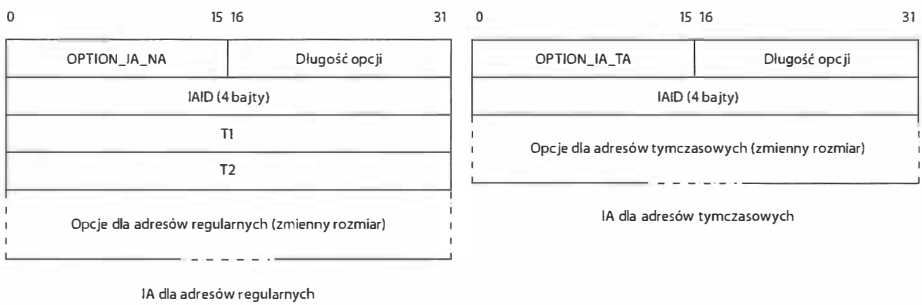
W komunikatach DHCPv6 najistotniejsze informacje — adresy, okresy dzierżawy, lokalizacje usług oraz identyfikatory klientów i serwerów — przenoszone są w formie opcji. Dwie ważne koncepcje związane z opcjami DHCPv6 to skojarzenie tożsamości i unikatowy identyfikator DHCP.

6.2.5.3. Skojarzenie tożsamości

Skojarzenie tożsamości (*Identity Association*, w skrócie IA) to encja reprezentująca informację o konfiguracji (m.in. — kolekcję adresów) opatrzona unikatowym *identyfikatorem* (*IA Identifier*, w skrócie IAID). Do każdego interfejsu, dla którego klient żąda informacji konfiguracyjnej, musi być przypisane co najmniej jedno IA, każde IA przypisane jest natomiast do dokładnie jednego interfejsu. Dla każdego IA przypisanego do interfejsu klienta klient wybiera unikatowy identyfikator IAID, który będzie reprezentował owo IA w komunikacji z serwerem.

Składająca się na IA informacja konfiguracyjna obejmuje m.in. adresy (jeden lub więcej) wraz z informacją o dzierżawie (wartości T, T1 i T2) i długością okresu ważności i okresu preferencji dla każdego adresu (patrz [RFC4862]). Ponadto kolekcja adresów zaliczana jest do jednego z dwóch typów — tworzące ją adresy mogą być adresami *regularnymi* (*nontemporary*) albo *tymczasowymi* (*temporary*) (patrz [RFC4941]). Adresy tymczasowe generowane są przy znaczącym udziale liczb pseudolosowych, jako takie sprzyjają więc ochronie prywatności poprzez utrudnianie szpiegowania hostów przy użyciu ich adresów IPv6. Zasadniczo adresy tymczasowe przydzielane są wraz z adresami regularnymi, lecz — dzięki używaniu liczb pseudolosowych — cechują się znacznie większą zmiennością niż adresy regularne.

Serwer odpowiadający na żądanie klienta przydziela w ramach określonego IA jeden lub więcej adresów, bazując na założeniach przydzielania adresów (*address assignment policies*) ustalonych przez administratora. Założenia te generalnie zależne są od łącza, z którego przychodzi żądanie, standardowej informacji o kliencie (DUID — o tym w następnym podpunkcie) oraz innych informacji dostarczanych przez opcje DHCP. Na rysunku 6.13 pokazany jest format rekordu reprezentującego IA, odpowiednio dla adresów regularnych i adresów tymczasowych.



Rysunek 6.13. *Formaty IA dla adresów regularnych (po lewej) i adresów tymczasowych (po prawej). Każda opcja może dostarczać dodatkowych informacji na temat poszczególnych adresów i ich dzierżawy*

Najważniejszą, widoczną na pierwszy rzut oka różnicą między obydwoma formatami IA jest brak pól T1 i T2 dla adresów tymczasowych. Dla adresów regularnych wartości te pełnią tę samą rolę, co w DHCPv4, ich brak w adresach tymczasowych wynika z faktu, że często przyjmowane są one w postaci obowiązującej dla uprzednio przydzielonych adresów regularnych. Szczegóły dotyczące wykorzystywania adresów tymczasowych znajdują się w dokumencie [RFC4941].

6.2.5.4. Unikatowy identyfikator DHCP (DUID)

Unikalny identyfikator DHCP — DUID (od *DHCP Unique Identifier*) — identyfikuje klienta lub serwer w sposób unikatowy i pozostaje niezmienny w czasie. Serwery wykorzystują DUID klientów w procesie przydziału adresów i udostępniania innych informacji konfiguracyjnych (w ramach IA). DUID mają zróżnicowaną długość i w większości przypadków traktowane są przez klienty i serwery jako całość, bez interpretowania ich struktury.

Od DUID wymaga się unikatowości, jednocześnie algorytm ich generowania nie może być zbyt skomplikowany. W zamiarze pogodzenia tych sprzecznych wymagań dokument [RFC3315] definiuje trzy różne typy DUID, przewidując jednocześnie możliwość rozszerzania tego zbioru o kolejne typy[3]. Wspomniane trzy typy to:

- DUID-LLT — tworzone na bazie adresu warstwy łącza danych oraz bieżącego wskazania czasu,
- DUID-EN — tworzone na bazie numeru przedsiębiorstwa (PEN) i identyfikatora producenta,
- DUID-LL — tworzone wyłącznie w oparciu o adres warstwy łącza danych.

Typ DUID kodowany jest na jego początkowych dwóch bajtach — wymienione powyżej typy identyfikowane są wartościami (kolejno) 1, 2 i 3 (aktualna lista zdefiniowanych identyfikatorów dostępna jest pod adresem [ID6PARAM]). W identyfikatorach DUID-LLT i DUID-LL kolejne dwa bajty określają tzw. typ sprzętowy (*hardware type*) w oparciu o [RFC0826], w identyfikatorze DUID-EN po dwóch początkowych bajtach następuje 32-bitowy prywatny numer przedsiębiorstwa (PEN)[4].

Prywatny numer przedsiębiorstwa (*Private Enterprise Number*, w skrócie PEN) to 32-bitowa wartość nadawana przedsiębiorstwu przez IANA, wykorzystywana zwykle w kontekście protokołu SNMP na potrzeby zarządzania siecią. Kompletna lista przydzielonych numerów PEN, dostępna pod adresem [IEPARAM], liczy obecnie (24 grudnia 2012 roku) 41 068 pozycji.

Spośród trzech wymienionych typów DUID zalecany do użytku jest DUID-LLT. Po dwóch początkowych bajtach zawierających wartość (kolejno) 0 i 1 i kolejnych dwóch określających typ sprzętowy następuje 32-bitowy znacznik (licznik) czasowy, w postaci liczby sekund, jakie upłynęły od północy rozpoczynającej rok 2000. Pojemność tego

[3] Dokument RFC6355 z sierpnia 2011 roku definiuje kolejny typ DUID — DUID-UUID, oparty na unikatowych identyfikatorach UUID/GUID generowanych natywnie w systemie operacyjnym, identyfikowany wartością 4 w dwóch pierwszych bajtach — *przyp. tłum.*

[4] W identyfikatorze DUID-UUID po dwóch początkowych bajtach następuje 128-bitowy UUID — *przyp. tłum.*

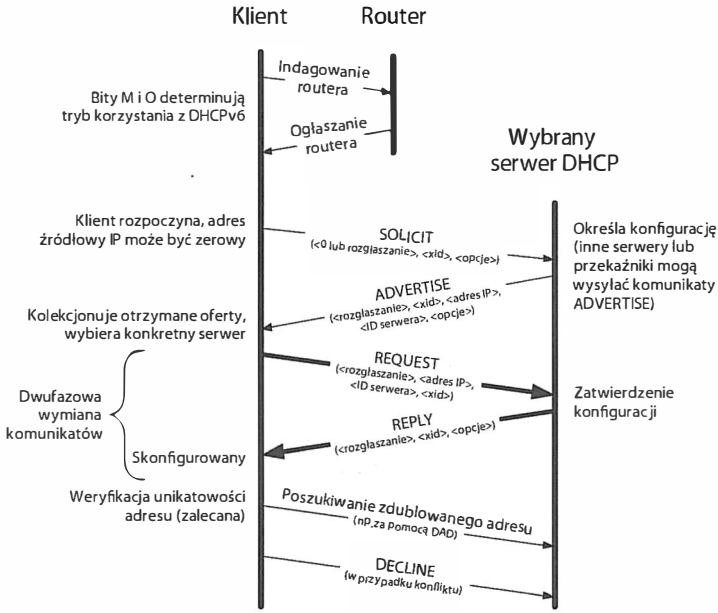
licznika wystarcza więc na 2^{32} sekund, czyli ok. 136 lat. Kolejne bajty zawierają adres warstwy łącza danych w tzw. postaci kanonicznej, zgodnie z dokumentem RFC2464; może to być adres dowolnego interfejsu klienta, byle stosowany konsekwentnie w odniesieniu do ruchu na wszystkich interfejsach i to nawet wtedy, gdy wspomniany wybrany interfejs zostanie z hosta usunięty, host musi więc dysponować stabilną (nieulotną) pamięcią do jego zapamiętania. Dla hostów pozbawionych takiej pamięci, lecz *gwarantujących nieusuwalność* wybranego adresu warstwy łącza danych (czyli wybranego interfejsu), zalecany jest identyfikator `DUID-LL` o strukturze podobnej do `DUID-LLT` (nie zawiera on znacznika czasowego, adres warstwy łącza danych rozpoczyna się od piątego bajta).

6.2.5.5. Operacje protokołu DHCPv6

Elementy funkcjonalne protokołu DHCPv6 podobne są do tych z DHCPv4. To, w jakim zakresie (jeśli w ogóle) klient w ogóle korzystać będzie z DHCP, zależy od opcji konfiguracyjnych dostarczanych w komunikacie ICMPv6 *Router Advertisement* (patrz [RFC5175]) wysyłanym do klienta przez router (patrz rozdział 8.). Częścią tego komunikatu jest bajt znaczników, zawierający m.in. bity oznaczone *M* i *O*. Ustawienie bitu *M* (to skrót od *Managed Address Configuration* — [realizowane] zarządzenie konfiguracją adresów) oznacza, że potrzebne adresy IPv6 może klient uzyskać za pomocą protokołu DHCPv6; ustawienie bitu *O* (skrót od *Other Configuration* — inna konfiguracyjna) oznacza możliwość uzyskania za pomocą DHCPv6 dodatkowej informacji konfiguracyjnej niezwiązanej z adresami. Kombinacja *M*=0, *O*=0 jest bodaj najmniej przydatna, bo oznacza niemożność konfigurowania adresów za pośrednictwem DHCPv6, zatem klient musi wykonać to zadanie za pomocą mechanizmów autokonfiguracji, opisywanych w podrozdziale 6.3. Kombinacja *M*=0, *O*=1 także oznacza konieczność autokonfigurowania, lecz serwer DHCPv6 może dostarczać dodatkowych informacji konfiguracyjnych.

W tabeli 6.1 prezentowaliśmy już komunikaty protokołu DHCPv6, natomiast na rysunku 6.14 przedstawiamy scenariusz ich wymiany między klientem a routerem i wybranym serwerem DHCP.

Przedstawiony na rysunku scenariusz zapoczątkowywany jest przez klienta, który za pomocą komunikatu ICMPv6 *Router Discovery* (patrz rozdział 8.) wysłanego do routera próbuje uzyskać adres IP lokalny dla łącza. Odpowiedź routera zawiera (oprócz wielu innych składników) opisywane wcześniej bity `M` i `O`; gdy dostępne są usługi serwera DHCPv6, ustawiony jest przynajmniej bit `M`. Klient po stwierdzeniu, że `M` = 1, wysyła w trybie rozgłaszania (patrz rozdział 9.) komunikat `SOLICIT` w celu odnalezienia dostępnych serwerów DHCPv6, na co reakcją jest nadejście komunikatu `ADVERTISE` (lub wielu takich komunikatów) świadczące o dostępności serwera DHCPv6 (lub wielu takich serwerów). Wymienione komunikaty stanowią pierwsze dwa stadia tzw. czterofazowej wymiany komunikatów (*four-message exchange*) charakterystycznej dla DHCPv6 (dwa następne stadia to komunikaty `REQUEST` i `REPLY`); jednak w sytuacji, gdy lokalizacja DHCPv6 jest już znana klientowi bądź klient nie zabiega o przydzielenie adresu (np. w przypadku bezstanowego DHCPv6 lub użycia opcji *Rapid Commit* — patrz punkt 6.2.9), te dwa stadia są nieobecne i wymiana czterofazowa redukuje się do dwufazowej. Potwierdzenie ze strony serwera ma formę komunikatu `REPLY`, na który składają się m.in. DUID, IA w jednym z trzech typów — dla adresów tymczasowych lub regularnych



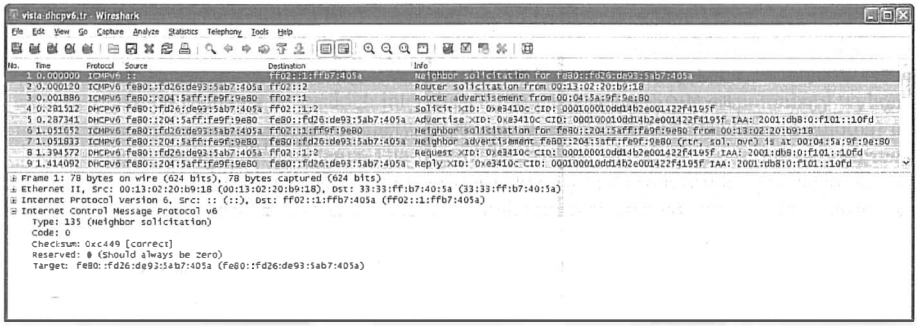
Rysunek 6.14. *Podstawowy schemat funkcjonowania protokołu DHCPv6. Klient na podstawie otrzymanego komunikatu ICMPv6 Router Advertisement sprawdza, czy dostępne są usługi DHCPv6 — jeśli tak, to klient z nich korzysta, podobnie jak w DHCPv4, lecz szczegóły ich realizacji są w porównaniu z DHCPv4 znacząco różne*

(patrz podpunkt 6.2.5.3) albo dla prefiksu (patrz podpunkt 6.2.5.7), oraz identyfikator IAID, będący 32-bitową liczbą wybraną przez klienta. Każde takie potwierdzenie (wiązanie) może obejmować jedną lub kilka dzierżaw, a w ramach jednej transakcji DHCPv6 możliwa jest obsługa wielu wiązań.

6.2.5.6. Szczegółowy przykład

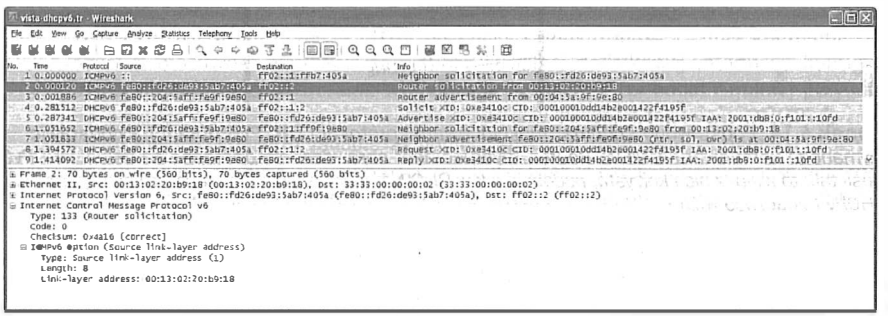
Na rysunku 6.15 widoczne jest okno programu Wireshark uruchomionego w komputerze z systemem Windows Vista +SP1, podłączającym się właśnie do sieci bezprzewodowej. Stos protokołów IPv4 jest w tym komputerze wyłączony. Klient rozpoczyna interakcję od uzyskania adresu lokalnego dla łącza, a następnie sprawdza, czy adres ten nie jest już używany.

Widzimy w związku z tym komunikat ICMPv6 *Neighbor Solicitation* (DAD) z „optymistycznym" adresem `fe80::fd26:de93:5ab7:405a` (procedurą DAD zajmiemy się dokładniej w podpunkcie 6.3.2.1, przy okazji omawiania bezstanowego konfigurowania adresów). Klient optymistycznie zakłada, że adres nie jest używany nigdzie indziej na łączu; w celu zweryfikowania tego założenia wysyła wspomniany komunikat na powiązany adres *solicited node* `ff02::1:ffb7:405a`, po czym natychmiast rozpoczyna procedurę „indagowania" routerów (*Router Solicitation* — RS), wysyłając komunikat *Router Solicit* na adres multicast `ff02::2` reprezentujący wszystkie routery (patrz rysunek 6.16).



Rysunek 6.15. *Procedura DAD weryfikacji adresu lokalnego dla łącza, realizowana przez klienta za pomocą komunikatów Neighbor Solicitation*

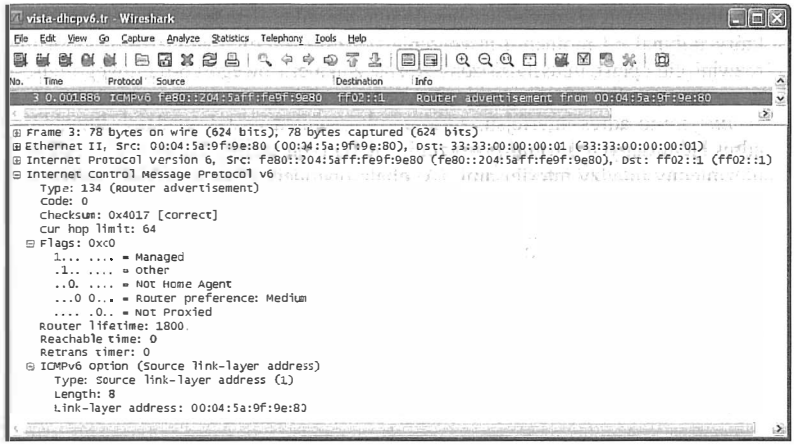
Każdy router, który odbierze ten komunikat, odpowie wysłaniem komunikatu-ogłoszenia *Router Advertisement* (RA), zawierającego rozmaite informacje, z których najważniejszą jest w tym momencie stan bitów *M* i *O*, decydujący o dalszych poczynaniach klienta.



Rysunek 6.16. *Komunikat Router Solicitation jako żądanie wysłania komunikatu Router Advertisement przez najbliższy router; komunikat wysyłany jest na adres multicast All Routers (ff02::2)*

W prezentowanym przykładzie widzimy indagowanie routerów w trybie optymistycznym, czyli przy użyciu adresu źródłowego łącza danych (*Source Link-Layer Address Option* — SLLAO), którego unikatowość nie jest gwarantowana. Adres ten, w połączeniu z adresem sprzętowym nadawcy, zostanie odzwierciedlony w tabelach trasowania przyległych routerów, jeśli więc optymizm nadawcy okaże się być bez pokrycia — bo procedura DAD wykryje zdublowanie adresu — nadawca posłuży się zapewne innym adresem, lecz *we wspomnianych tablicach trasowania pozostanie niepoprawna pozycja*, fałszująca rzeczywistą zależność pomiędzy adresem sprzętowym nadawcy a faktycznie używanym przez niego adresem IP. Problem ten, sygnalizowany w dokumencie [RFC4429], nie wydaje się jednak poważny, bo samo prawdopodobieństwo zdublowania „optymistycznie" wybranego adresu jest niewielkie. W publikacji [IDDN] opisywane są prace mające na celu zoptymalizowanie opcji SLLAO pod kątem eliminacji wspomnianego zagrożenia.

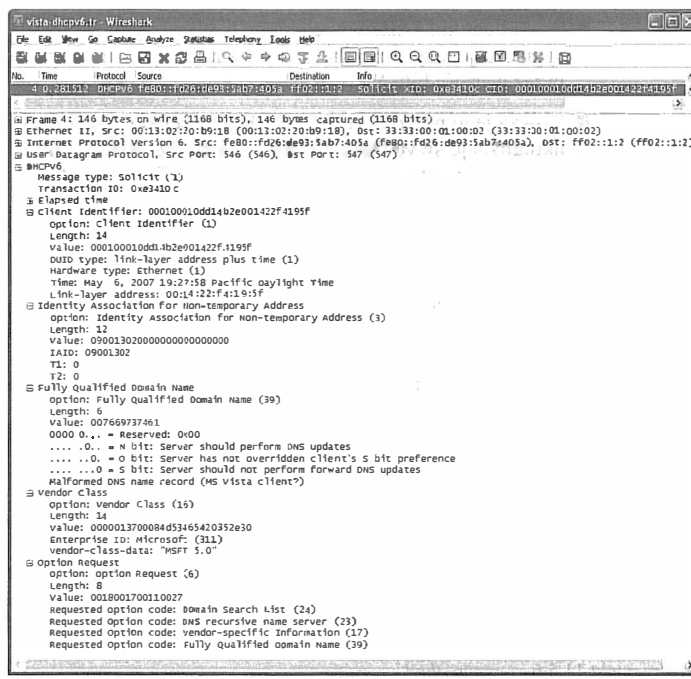
Widoczny na rysunku 6.17 komunikat RA wskazuje obecność routera i jego adres łącza danych `00:04:5a:9f:9e:80`, który wykorzystywany będzie przez klienta jako docelowy w ramkach przyszłej jego komunikacji ze wspomnianym routerem. W bajcie znaczników widzimy ustawienie obu bitów `M` i `O`, klient ma więc możliwość uzyskania od serwera



Rysunek 6.17. *Komunikat ICMPv6 Router Advertisement (patrz rozdział 8.) wskazuje na dostępność usługi DHCPv6 w zakresie przydzielania adresów oraz udzielania dodatkowych informacji konfiguracyjnych (m.in. adresu serwera DNS) — serwer DHCPv6 wykorzystywany jest w trybie stanowym*

DHCPv6 zarówno przydziału adresu (adresów), jak i innych informacji pomocniczych. Klient wykorzystuje tę możliwość, wysyłając do wspomnianego serwera komunikat SOLICIT, co uwidocznione zostało na rysunku 6.18.

Rysunek 6.18. *Komunikat DHCPv6 SOLICIT z żądaniem lokalizacji jednego lub kilku serwerów DHCPv6; komunikat zawiera identyfikator klienta oraz sprecyzowanie szczegółów wymaganej przez niego informacji*



Komunikat SOLICIT zawiera identyfikator transakcji (podobnie jak w DHCPv4), czas, który upłynął od momentu utworzenia (*elapsed time*) równy 0 (co widoczne jest po rozwinięciu opcji), i DUID złożony ze znacznika czasowego i 6-bajtowego adresu MAC, w tym przypadku adresu przewodowego interfejsu ethernetowego 00:14:22:f4:19:5f — *nie jest* to adres interfejsu, z którego komunikat został wysłany. Przypomnijmy, że adres łącza danych użyty do tworzenia DUID typu DUID-LLT i DUID-LL musi pozostawać niezmienny między interfejsami. IA, opatrzone identyfikatorem IAID równym 09001302 (wybranym przez klienta), zawiera adresy regularne. Wartości T1 i T2 pozostały w żądaniu z wartością zerową, co oznacza brak szczególnych wymagań klienta względem nich; ich wartości zostaną ustalone przez serwer.

Następna opcja — FQDN — opisywana w [RFC4704] służy do przekazania klientowi pełnej kwalifikowanej nazwy jego domeny, lecz ma także wpływ na sposób współdziałania DHCPv6 i DNS (piszemy o tym w podrozdziale 6.4). Opcja ta wykorzystywana jest do dynamicznego uaktualniania tablic odwzorowujących nazwy domen na adresy IPv6 przez klienty i serwery (odwzorowanie odwrotne wykonywane jest zwykle tylko przez serwery).

Pierwsza część obszaru opcji to bajt znaczników określających szczegóły wspomnianego odwzorowywania —aktualnie interpretowane są trzy następujące bity tego bajta. Oto one.

- S — za jego pomocą klient określa, czy serwer powinien wykonywać aktualizację rekordu źródłowego AAAA (1) albo jej nie wykonywać (0)[5]. W przesyłanej klientowi odpowiedzi serwer sygnalizuje swą decyzję w tym względzie — wartość 1 tego bitu oznacza, że serwer przyjmuje odpowiedzialność za wspomnianą aktualizację.
- O — klient musi ustawić ten bit na 0; w przesyłanej klientowi odpowiedzi serwer sygnalizuje, czy zmienił (1), czy też zaakceptował (0) preferencję klienta wyrażoną przez wartość bitu *S*.
- N — za jego pomocą klient określa, czy serwer powinien wykonywać jakąkolwiek aktualizację swych tablic DNS (0), czy też nie (1). Gdy ustawiony jest na 1, bit *S* musi mieć wartość 0. Wartość zwrócona klientowi informuje go, czy serwer będzie wykonywał wspomnianą aktualizację (0), czy nie (1).

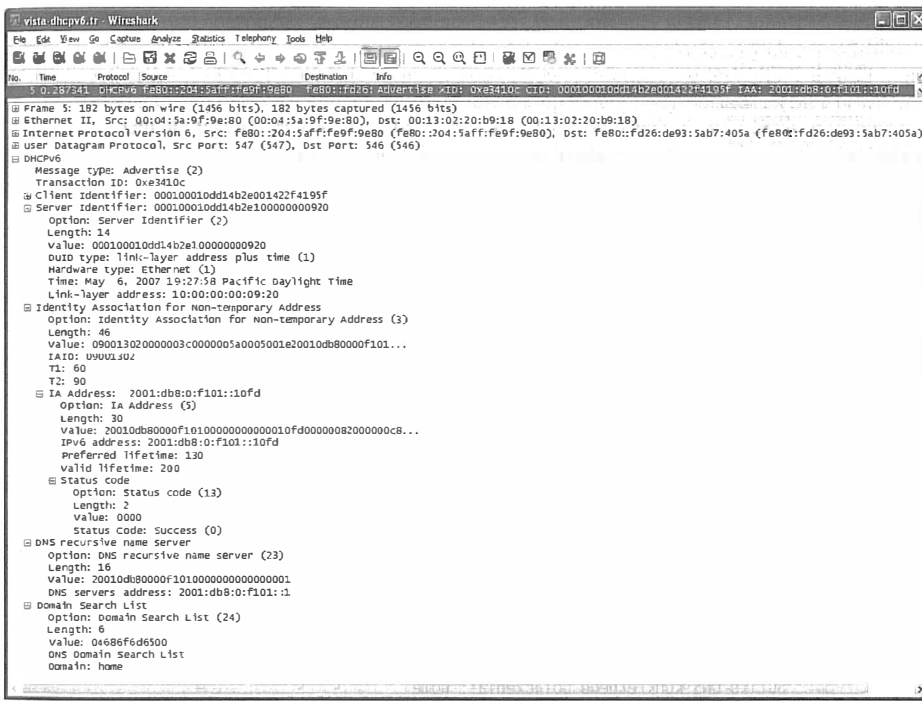
Pozostałe bity bajta znaczników powinny być zerowane przez klienta oraz serwer i nie-interpretowane. Druga część opcji przeznaczona jest na nazwę domeny. Klient umieszcza w tym polu częściową nazwę domeny, serwer natomiast uzupełnia ją do pełnej kwalifikowanej postaci (FQDN). Klient może także umieścić w tym obszarze pełną nazwę FQDN, w takim przypadku nazwa ta musi zostać zakończona zerowym bajtem. W obu przypadkach koniec nazwy rozpoznawany jest na podstawie pola długości opcji.

Na rysunku 6.18, we fragmencie dotyczącym opcji FQDN, zobaczyć można ostrzeżenie o niepoprawnym formacie nazwy domeny i sugestię, że nazwa ta mogła zostać wygenerowana w systemie Windows Vista — *Malformed DNS name record (MS Vista Client?)* — co istotnie miało miejsce. W systemie Windows Vista wykorzystano mianowicie przestarzały sposób kodowania nazw domen — w postaci zwykłych łańcuchów ASCIIZ. Sposób ten został zarzucony przez specyfikację RFC4704 na rzecz kodowania opartego na hierarchicznym układzie etykiet. Microsoft udostępnił użytkownikom Visty doraźną poprawkę (*hotfix*) eliminującą ten błąd; w Windows 7 kodowanie nazw domen wykonywane jest już zgodnie z RFC4704.

[5] Patrz punkt 11.5.6 — *przyp. tłum.*

Pozostałe opcje komunikatu SOLICIT obejmują klasę producenta i listę parametrów żądania. Klasa producenta — w tym przypadku MSFT 5.0 — wykorzystywana jest przez serwery DHCPv6 do określenia, do jakiego rodzaju przetwarzania zdolny jest klient.

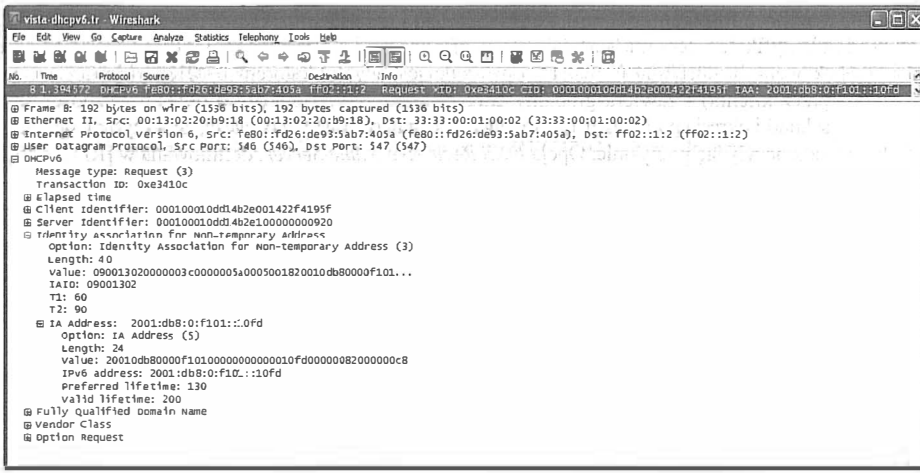
Odpowiedzią serwera na otrzymany komunikat SOLICIT jest wysłanie do klienta komunikatu ADVERTISE, co uwidocznione zostało na rysunku 6.19. Komunikat ten to istne bogactwo informacji: identyfikator klienta jest kopią identyfikatora zawartego w żądaniu, identyfikator serwera jest konkatenacją znacznika czasowego i adresu sprzętowego serwera 10:00:00:00:09:20. IA opatrzone jest identyfikatorem 09001302 (dostarczonym przez klienta) i zawiera globalny adres 2001:db8:0:f101::10fd o okresie ważności 200 sekund i okresie preferencji 130 sekund. Kod statusu równy 0 oznacza, że wszystkie operacje zakończyły się pomyślnie. Opcja *DNS Recursive Name Server*, definiowana w [RFC3646], zawiera adres serwera DNS —2001:db8:0:f101::1 — natomiast w polu opcji przeszukiwania domeny (*Domain Search List*) widnieje łańcuch home. Zauważmy, że w komunikacie brakuje opcji FQDN, ponieważ serwer DHCP jej nie implementuje.



Rysunek 6.19. *Komunikat DHCPv6 ADVERTISE zawiera proponowany adres i okres jego dzierżawy, a także adres IPv6 serwera DNS i listę przeszukiwania domeny*

Dwa kolejne pakiety to konwencjonalne komunikaty indagowania sąsiadów (*Neighbor Solicitation*) i ogłoszenie sąsiada (*Neighbor Advertisement*) — nie będziemy analizować ich szczegółów. Następny komunikat to żądanie klienta REQUEST dotyczące zatwierdzenia globalnego regularnego adresu 2001:db8:0:f101::10fd (patrz rysunek 6.20). Komunikat

REQUEST (patrz rysunek 6.20) podobny jest do komunikatu SOLICIT, dodatkowo zawiera jednak elementy dostarczone przez serwer w komunikacie ADVERTISE z rysunku 6.19 — adres IP oraz wartości T1 i T2. Identyfikator transakcji (XID) pozostaje niezmienny we wszystkich komunikatach DHCPv6, aż do zakończenia transakcji przez komunikat REPLY, różniący się od komunikatu ADVERTISE jedynie wartością w polu *Typ komunikatu*.



Rysunek 6.20. *Komunikat DHCPv6 REQUEST jest podobny do komunikatu SOLICIT, zawiera jednak informację otrzymaną przez klienta w komunikacie ADVERTISE*

Zaprezentowana wymiana komunikatów DHCPv6 jest typowa dla bootowania systemu lub przyłączania hosta do nowej sieci. Podobnie jak w przypadku DHCPv4, możliwe jest jawne zmuszenie systemu do uzyskania niezbędnych danych lub ich zwolnienia. W systemach Windows zwolnienie to realizowane jest za pomocą polecenia:

```
C:\> ipconfig /release6
```

Natomiast opisany proces jej pozyskiwania inicjuje się poleceniem:

```
C:\> ipconfig /renew6
```

Zestaw informacji konfiguracyjnych związanych z poszczególnymi interfejsami uzyskać można za pomocą prezentowanego już polecenia:

```
C:\> ipconfig /all
...
Karta bezprzewodowej sieci LAN Połączenie sieci bezprzewodowej
   Sufiks DNS konkretnego połączenia : home
   Opis. . . . . . . . . . . . . . . : Intel(R) PRO/Wireless 3945ABG Network
                                       Connection
   Adres fizyczny. . . . . . . . . . : 00-13-02-20-89-18
   DHCP włączone . . . . . . . . . . : Tak
   Autokonfiguracja włączona . . . . : Tak
   Adres IPv6. . . . . . . . . . . . : 2001:db8:0:f101::12cd(Preferowane)
   Dzierżawa uzyskana. . . . . . . . : 21 grudnia 2008 23:30:45
   Dzierżawa wygasa. . . . . . . . . : 22 grudnia 2008 23:37:04
   Adres IPv6 połączenia lokalnego fe80::fd26:de93:5ab7:405a%9(Preferowane)
```

```
Brama domyślna: fe80::204:5aff:fe9f:9e80%9
Identyfikator IAID DHCPv6: 150999810
Identyfikator DUID klienta DHCPv6: 00-01-00-01-0D-D1-4B-2E-00-14-22-F4-19-5F
Serwery DNS : 2001:db8:0:f101::1
NetBIOS przez Tcpip : Wyłączony
Lista przeszukiwania sufiksów DNS : home
```

Zwróćmy uwagę na ważny fakt, iż widoczny w raporcie adres łącza danych (00:13:02:20:b9:18) nie został nigdy wykorzystany w prezentowanym przykładzie na potrzeby generowania adresów IPv6.

6.2.5.7. Delegowanie prefiksów (DHCPv6-PD i 6rd)

Choć nasza dyskusja na temat protokołu DHCP koncentrowała się wokół konfigurowania hostów, to DHCPv6 może być wykorzystywany także do konfigurowania routerów. Konfigurowanie to realizuje się za pomocą *delegowania* przez router zakresu przestrzeni adresowej do innego routera — zakres ten specyfikowany jest w postaci prefiksu IPv6, przenoszonego w ramach opcji *Prefix*, definiowanej w [RFC3633]. Delegowanie takie jest używane w sytuacji, gdy router delegujący — spełniający jednocześnie funkcje serwera DHCPv6 — nie wymaga szczegółowej informacji o topologii sieci, do której delegowany jest wspomniany prefiks. Jest tak np. wtedy, gdy dostawca (ISP) przydziela klientowi pewien zakres adresów IP, pozostawiając mu swobodę w zakresie posegmentowania tego zakresu i przydzielenia konkretnych adresów dla poszczególnych hostów: dostawca może wówczas zdecydować o delegowaniu wspomnianego zakresu do sprzętu w siedzibie klienta, przy użyciu opcji DHCPv6-PD.

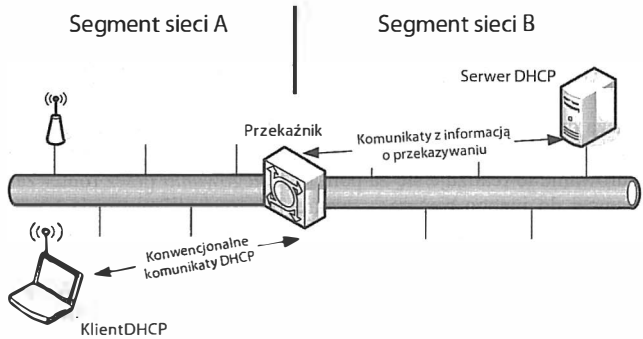
W związku z delegowaniem prefiksu definiuje się trzecią odmianę IA (IA_PD), oprócz dwóch istniejących, a przeznaczonych dla adresów regularnych (IA_NA) i adresów tymczasowych (IA_TA). Podobnie jak dwie pozostałe, IA_PD opatrzona jest unikatowym identyfikatorem IAID i zawiera informacje w formacie podobnym do adresów. DHCPv6-PD zalecane jest nie tylko dla routerów stacjonarnych, lecz również (patrz [RFC6276]) dla mobilnych routerów (i — oczywiście — podsieci przemieszczających się wraz z tymi routerami).

Specjalna odmiana DHCPv6-PD, zwana *6rd* i opisywana w [RFC5569], zaprojektowana została z myślą o obsłudze błyskawicznego wdrażania IPv6 przez dostawców Internetu („rd" to skrót od *rapid development* — błyskawiczne wdrażanie — zaś cyfra 6 symbolizuje, oczywiście, IPv6). Jak można się domyślać, „błyskawiczne" oznaczy to samo, co „zautomatyzowane" — faktycznie, przydzielony aktualnie klientowi adres unicast IPv4 staje się bazą dla algorytmicznego generowania informacji związanej z adresami IPv6. A konkretnie: 32-bitowy adres IPv4 (lub jego początkową część wyodrębnioną na podstawie maski — patrz poniżej) zostaje poprzedzony dostarczonym przez ISP *prefiksem 6rd* o długości mniejszej niż 32 bity; prefiks 6rd jest treścią opcji OPTION_6RD (numer 212), definiowanej w [RFC5969]. Wynikowa konkatenacja, o długości mniejszej niż 64 bity, tworzy prefiks delegowany w ramach DHCPv6-PD, wykorzystywany później w procesie autokonfiguracji (o której piszemy w podrozdziale 6.4).

Wspomniana opcja OPTION_6RD ma zmienną długość i zawiera następujące elementy: długość maski adresu IPv4, długość prefiksu 6rd, sam prefiks 6rd i listę przekaźników IPv4 (czyli listę adresów IPv4 przekaźników implementujących funkcję 6rd). Maska adresu IPv4 określa jego początkową część konkatenowaną z prefiksem 6rd.

6.2.6. Przekaźniki DHCP

W większości prostych sieci pojedynczy serwer DHCP dostępny jest bezpośrednio dla wszystkich węzłów; w bardziej złożonych sieciach — np. sieciach korporacyjnych — może być niezbędne (lub wygodne) przekazywanie ruchu przez jeden lub więcej *przekaźników* (*relay agents*), co w najprostszym wariancie przedstawiono schematycznie na rysunku 6.21.



Rysunek 6.21. *Agent przekazywania (przekaźnik) DHCP rozszerza dostępność operacji DHCP poza pojedynczą sieć. Przenoszenie informacji specyficznych dla komunikacji przekaźnika z serwerem DHCPv4 odbywa się przy użyciu opcji Relay Agent Information; w DHCPv6 jest podobnie, inny jest jednak zestaw wykorzystywanych opcji*

Przekaźnik umożliwia rozszerzenie operacji DHCP na wiele segmentów sieci: w konfiguracji przedstawionej na rysunku 6.21 przekaźnik rezydujący na styku dwóch segmentów zajmuje się forwardowaniem komunikatów DHCP, które wzbogacać może o dodatkowe informacje, np. za pomocą opcji lub przez wypełnienie standardowo pustych pól. Zauważmy przy tym, że przekaźnik zwykle nie obsługuje całego ruchu DHCP wymienianego między klientem a serwerem przez granice segmentu: zajmuje się on jedynie forwardowaniem komunikatów rozgłoszeniowych, wymienianych w procesie negocjowania adresów: gdy tylko klient uzyska przydział adresu IP i otrzyma adres IP serwera (za pośrednictwem opcji *Server Identification*), przekaźnik staje się zbędny, bo dalsza komunikacja między klientem a serwerem odbywa się w sposób normalny, tak jak między dwoma adresami unicast. Tradycyjnie przekaźniki były i są urządzeniami warstwy 3., w związku z czym często wyposażane bywają w funkcje trasowania, jak jednak zobaczymy dalej w tym rozdziale, alternatywą są dla nich przekaźniki operujące (prawie) wyłącznie w warstwie 2.

6.2.6.1. Opcje przekazywania

Według oryginalnej koncepcji dokumentu [RFC2131] jedynym zadaniem przekaźnika BOOTP lub DHCP jest umożliwienie komunikacji między dwoma systemami (segmentami) niezdolnymi komunikować się za pośrednictwem routera. Dzięki temu system niezdolny do wykonywania funkcji dostarczania pośredniego, bo nieposiadający jeszcze adresu IP, może taki adres uzyskać z centralnej lokalizacji. Rozwiązanie to spisuje się znakomicie w sieciach zarządzanych centralnie w sposób administracyjny, lecz jest

niewystarczające chociażby wtedy, gdy infrastruktura DHCP znajduje się z dala od miejsca, w którym wykorzystywane są usługi DHCP. W takiej sytuacji znajduje się np. dostawca Internetu, mający ograniczone (z definicji) zaufanie do swych klientów i potrzebujący dodatkowych informacji o charakterze statystycznym, dla celów rozliczeń czy optymalizacji wykorzystywania sieci. Informacji tych, rzecz jasna, nie dostarcza standardowo protokół DHCP. W sieciach IPv4 zadanie to spełnia opcja *Relay Agent Information*, w skrócie RAIO, definiowana w dokumencie [RFC3046]; jej odpowiednikiem w IPv6 zajmiemy się w następnym podpunkcie.

RAIO to nie pojedyncza opcja, lecz w istocie framework, w ramach którego możliwe jest definiowanie różnych podopcji, m.in. identyfikujących źródło żądania DHCP — użytkownika, sieć, obwód itp. Wiele z tych podopcji ma swe odpowiedniki w postaci opcji w wersji IPv6.

Ponieważ sporo informacji wymienianych w komunikacji między przekaźnikiem i serwerem to informacje istotne z perspektywy bezpieczeństwa systemu, powstaje problem uwierzytelniania tej komunikacji; jest to zadanie podopcji *Authentication* definiowanej w dokumencie [RFC4030], zapewniającej integralność wspomnianej informacji. Jej funkcjonowanie podobne jest do opóźnionego uwierzytelniania DHCP, opisywanego w punkcie 6.2.7, z tą różnicą, że zamiast haszowania MD5 używany jest silniejszy algorytm SHA-1 (o algorytmach kryptograficznych piszemy w rozdziale 18.).

6.2.6.2. Podopcja Remote-ID (DHCPv4) i opcja Remote-ID (DHCPv6)

Jednym z podstawowych wymagań stawianych przekaźnikowi jest zdolność do identyfikowania klienta, formułującego żądanie, za pomocą informacji wykraczających poza te, które sam klient standardowo dostarcza. Opcja *Relay Agent Information*, zwana także *podopcją zdalnej identyfikacji* (*Remote-ID suboption*), umożliwia taką identyfikację przy użyciu wielkości interpretowalnych w sposób lokalny (identyfikatora użytkownika, jego nazwy, identyfikatora modemu, zdalnego adresu IP na łączu punkt-punkt itp.). W DHCPv6 identyczną możliwość daje opcja *Relay Agent Remote-ID*, zdefiniowana w dokumencie [RFC4649]; w stosunku do swej odpowiedniczki z DHCPv4 zawiera ona dodatkowe pole — numer przedsiębiorstwa (PEN), o którym wspominaliśmy w jednej z wcześniejszych uwag, określający znaczenie dalszych informacji. Powszechnie stosowaną praktyką jest obsadzanie DUID w roli zdalnego identyfikatora.

6.2.6.3. Nadpisywanie identyfikacji serwera

W niektórych przypadkach przekaźnik z własnej inicjatywy włącza się w przetwarzanie informacji przesyłanej między klientem a serwerem DHCP; efekt ten osiąga się za pomocą specjalnej podopcji *Server Identifier Override*, definiowanej w dokumencie [RFC5107] i stanowiącej wariant opcji RAIO.

Jak wcześniej wyjaśnialiśmy, rola przekaźnika kończy się generalnie w momencie uzyskania przez klienta adresu IP oraz identyfikatora serwera, ponieważ od tego momentu klient i serwer mogą komunikować się bezpośrednio. Okazuje się jednak, że w wielu przypadkach pożyteczna byłaby możliwość dołączania przez przekaźnik pewnych opcji (przykładowo identyfikatora obwodu) w komunikatach innych niż `SOLICIT`, np. w komunikacie `REQUEST` wysyłanym przez klienta znajdującego się w stanie `Odnawianie` (patrz

rysunek 6.10) — standardowo jednak przekaźnik nie uczestniczy w przetwarzaniu tego komunikatu, z powodu wyżej wymienionego. Można jednak zmienić ten stan rzeczy w następujący sposób. Gdy przekaźnik otrzymuje od klienta komunikat SOLICIT i przesyła go do serwera, może poinstruować ów serwer, by w zwrotnym komunikacie ADVERTISE (DHCPOFFER) zamiast faktycznego identyfikatora serwera znalazł się identyfikator przekaźnika. W rezultacie następne komunikaty klienta kierowane będą do przekaźnika, choć klient przekonany będzie, że komunikuje się bezpośrednio z serwerem. Efekt ten wywołany zostaje z inicjatywy przekaźnika, który w komunikacie SOLICIT umieszcza opcję *Server Identifier Override*, definiowaną w dokumencie [RFC5107], stanowiącą wariant opcji RAIO; właśnie ta opcja jest narzędziem wspomnianego „poinstruowania" serwera, wpisującego w rezultacie (zamiast własnej identyfikacji) 4-bajtowy identyfikator (zwykle adres IPv4) stanowiący dane opcji.

Opcja *Server Identifier Override* używana jest zwykle w połączeniu z inną, definiowaną w dokumencie [RFC5010] i nazwaną *Relay Agents Flag*. Opcja ta jest de facto zbiorem znaczników przenoszących informacje z przekaźnika do serwera; obecnie interpretowany jest tylko jeden znacznik, decydujący o tym, czy wysyłany przez klienta komunikat inicjujący transakcję używać ma adresu docelowego broadcast, czy unicast; zależnie od stanu tego znacznika serwer DHCP może oferować przydział różnych adresów.

6.2.6.4. Zapytanie o dzierżawę — proste lub hurtowe

W niektórych środowiskach wskazane jest kolekcjonowanie przez przekaźnik (lub inny niezależny system) informacji związanej z udzielaniem dzierżawy adresów konkretnemu klientowi, a w przypadku DHCPv6 także informacji związanej z delegowaniem prefiksów. W przykładowej konfiguracji — jak ta z rysunku 6.21 — przekaźnik może — oczywiście — rejestrować przechodzący przez niego ruch od klienta do serwera; problem jednak w tym, że w razie awarii przekaźnika zebrana przez niego informacja może zostać utracona, byłoby więc wskazane zapewnić mu możliwość odtworzenia jej na żądanie. Zadanie to spełnia mechanizm *zapytania o dzierżawę*, w oryginale nazwany *leasequery*, definiowany w dokumentach [RFC4388] i [RFC6148] (dla DHCPv4) i [RFC5007] (dla DHCPv6). Wysyłane przez przekaźnik do serwera żądanie udostępnienia wspomnianej informacji ma postać komunikatu DHCPLEASEQUERY (w DHCPv4) lub LEASEQUERY-REQUEST (w DHCPv6). Klient, którego żądanie dotyczy, może być w komunikacie identyfikowany na kilka sposobów: w DHCPv4 mamy do dyspozycji jego adres IPv4, adres MAC, identyfikator i zdalny identyfikator (*Remote ID*), DHCPv6 umożliwia natomiast użycie adresu IPv6 albo identyfikatora DUID.

Odpowiedzią serwera DHCPv4 może być jeden z trzech komunikatów: DHCPLEASEUNASSIGNED, DHCPLEASEACTIVE lub DHCPLEASEUNKNOWN — znaczenie każdego z nich zależne jest od kryterium zapytania użytego w komunikacie DHCPLEASEQUERY[6]. W przypadku zapytania na podstawie adresu IP klienta pierwszy komunikat oznacza, że przydzielanie wskazanego adresu IP leży w gestii tegoż serwera, lecz adres ten nie jest aktualnie dzierżawiony. Drugi komunikat oznacza trwającą dzierżawę wskazanego adresu; przekaźnikowi udostępniane są wszystkie parametry tej dzierżawy, łącznie z wartościami T1 i T2 (bez jakiejkolwiek sugestii, do czego ewentualnie mogłyby mu się przydać). Trzeci komunikat oznacza, że serwer nie dysponuje wiarygodną informacją na temat wskazanego adresu.

[6] Znaczenie to jest szczegółowo wyjaśnione w cytowanym dokumencie RFC4388 — *przyp. tłum.*

Serwer DHCPv6 odpowiada komunikatem LEASEQUERY-REPLY zawierającym opcję danych klienta (OPTION_CLIENT_DATA, numer 45), która de facto jest kolekcją opcji reprezentujących identyfikator klienta, jego adres IPv6, prefiks IPv6 i czas (w sekundach), jaki upłynął od ostatniej komunikacji klienta z serwerem (*Client Last Transaction Time*). W komunikacie LEASEQUERY-REPLY mogą ponadto występować dwie opcje, o nazwach *Relay Data* i *Client Link*: pierwsza obejmuje dane, jakie przekaźnik wysłał do serwera w odnośnym zapytaniu LEASEQUERY, druga natomiast zawiera adres łącza klienta, na którym dokonywane jest wiązanie adresów. Ponownie sposób wykorzystania tych informacji spoczywa całkowicie w gestii otrzymującego je przekaźnika.

Rozszerzeniem prostego żądania informacji o dzierżawie jest *hurtowe żądanie informacji o dzierżawie* (*Bulk Lease query*, w skrócie BL), opisywane w dokumencie [RFC5460] i artykule [ID4LQ]. Umożliwia ono pobieranie informacji o wielu wiązaniach równocześnie i realizowane jest przy użyciu protokołu TCP, nie UDP, jak w przypadku prostego żądania. Prawdę mówiąc, BL jest usługą dość specjalizowaną i nie jest traktowane jako element konwencjonalnego protokołu DHCP; klient żądający konwencjonalnej informacji konfiguracyjnej nie będzie więc raczej z BL korzystał.

BL może być natomiast przydatne w kontekście delegowania prefiksów. Router zazwyczaj funkcjonuje wtedy jako klient DHCP-PD: uzyskuje żądany prefiks i następnie wykorzystuje go do ustanowienia zakresu adresów przydzielanych klientom DHCP w zwykły sposób. Jednak w przypadku awarii (lub restartu) informacja o wspomnianym prefiksie może zostać utracona i trudno ją szybko odzyskać za pomocą prostego zapytania o dzierżawę, bo ta wymaga jakiegoś sposobu *identyfikacji konkretnego wiązania*. BL wychodzi temu problemowi naprzeciw poprzez uogólnienie zbioru możliwych typów zapytań.

W stosunku do prostego zapytania o dzierżawę, BL charakteryzuje się kilkoma rozszerzeniami. Po pierwsze, użycie protokołu TCP (z portem 547. dla IPv6 i portem 67. dla IPv4) zamiast UDP pozwoliło na zwiększenie rozmiaru informacji przenoszonej w ramach jednego zapytania, konieczne w związku z przeszukiwaniem dużej liczby delegowanych prefiksów. BL dostarcza także opcji *Relay Identifier*, umożliwiającej łatwiejsze identyfikowanie przekaźnika będącego źródłem zapytania. Zapytanie BL może więc bazować na identyfikatorze przekaźnika, adresie łącza (segmentu sieci) lub identyfikatorze przekaźnika.

Opcja *Relay ID* (w DHCPv6) i podopcja DHCPv4 o tej samej nazwie (patrz [ID4RI]) mogą zawierać DUID identyfikujący przekaźnik. Przekaźnik może dołączyć tę opcję do komunikatu forwardowanego do serwera, serwer natomiast może ją wykorzystać do skojarzenia wykonywanego wiązania z konkretnym przekaźnikiem. BL obsługuje zapytania oparte na DUID ([RFC5007] i [RFC4388]), lecz także te oparte na identyfikatorze przekaźnika, adresie łącza i zdalnym identyfikatorze.

Ponieważ, jak wcześniej wspominaliśmy, BL nie jest uważane za część DHCP, obsługiwane jest wyłącznie przez odpowiednie serwery bazujące na protokołach TCP i IP. Serwery te nie implementują kompletnego protokołu DHCP, lecz jedynie obsługę komunikatów LEASEQUERY.

Oprócz wymienionych już komunikatów LEASEQUERY-REQUEST i LEASEQUERY-REPLY, BL używa jeszcze dwóch. Gdy z odpowiedzią na żądanie związana jest dodatkowa informacja, jest ona przesyłana w serii komunikatów LEASEQUERY-DATA, następujących bezpośrednio po komunikacie LEASEQUERY-REPLY; każdy komunikat tej serii reprezentuje jedno wiązanie.

Koniec serii sygnalizowany jest przez komunikat LEASEQUERY-DONE. Wszystkie komunikaty LEASEQUERY-DATA oraz „graniczne" komunikaty LEASEQUERY-REPLY i LEASEQUERY-DONE używają wspólnego identyfikatora transakcji — tego samego, który przesłany został w komunikacie LEASEQUERY-REQUEST.

6.2.6.5. Przekaźniki warstwy 2.

W niektórych sieciach urządzenia warstwy 2. — m.in. przełączniki i mostki — zlokalizowane w pobliżu punktów końcowych zajmują się przekazywaniem i przetwarzaniem żądań DHCP. Jako urządzenia warstwy 2. nie implementują one w pełni stosu protokołów TCP/IP, nie są więc adresowalne za pomocą adresów IP i nie mogą funkcjonować jako klasyczne przekaźniki. Możliwe jest jednak ich działanie w trybie tzw. „odchudzonych" przekaźników DHCP (*Lightweight DHCP Relay Agents*, w skrócie LDRA), opisywane w publikacji [IDL2RA] dla IPv4 i dokumencie [RFC6221] dla IPv6. Zgodnie z tym opisem, interfejsy urządzenia podzielone zostają na dwie kategorie — prowadzące do urządzeń sieciowych, bliższe topologicznie serwerom (*network-facing*) i prowadzące do urządzeń klienckich (*client-facing*). Niezależnie od tego, interfejsy dzielone są na zaufane (*trusted*) i niezaufane (*untrusted*) — zaufane są te, na których nie występuje ryzyko „podrabiania" (*spoofing*) pakietów przychodzących.

Konsekwencją nieidentyfikowania LDRA przez adres IP jest problem z obsługa pola *giaddr* i opcji RAIO w komunikatach DHCP. Dokument [IDL2RA] zaleca w związku z tym, by LDRA dołączał opcję RAIO (o ile nie została dołączona wcześniej) do komunikatów przychodzących od klienta, lecz pozostawiał pole *giaddr* niewypełnione. W rezultacie przekazywany komunikat docierał będzie na zasadzie rozgłaszania do jednego lub wielu serwerów DHCP, jak również do każdego innego LDRA — przekaźnik jako mostek lub przełącznik wykonywać będzie tzw. *floodowanie*[7], czyli wysyłanie komunikatu przez wszystkie interfejsy różne od tego, przez które komunikat ten przybył. Floodowanie to dotyczyć będzie jednak tylko komunikatów odebranych przez interfejsy zaufane.

Odpowiedzi przesyłane przez serwery w trybie rozgłaszania (np. w postaci komunikatów DHCPOFFER) mogą być przechwytywane przez LDRA i — po usunięciu opcji RAIO — przekazywane klientom-nadawcom żądań. Wiele LDRA przechwytuje również ruch DHCP z adresami unicast, opcja RAIO jest wówczas także tworzona i usuwana w analogiczny sposób.

Z powyższego opisu wynika wymaganie pod adresem kompatybilnych serwerów DHCP, by te zdolne były do przetwarzania komunikatów DHCP z pustym (lub niepoprawnie wypełnionym) polem *giaddr* niezależnie od tego, czy komunikaty te rozsyłane są w trybie rozgłoszeniowym, czy wysyłane w trybie unicast.

Komunikaty DHCPv6 przetwarzane są przez kompatybilne LDRA przy użyciu komunikatów RELAY-FORW i RELAY-REPL. Komunikaty ADVERTISE, REPLY, RECONFIGURE i RELAY-REPL przychodzące na interfejsy od strony klientów są ignorowane, podobnie jak (ze względów bezpieczeństwa) komunikaty RELAY-FORW przychodzące na niezaufane interfejsy klienckie.

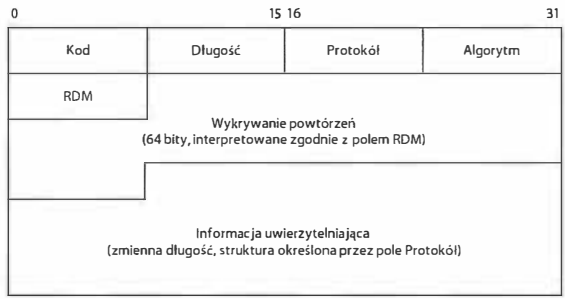
[7] Od ang. *flooding* — wylew, powódź — *przyp. tłum.*

Komunikaty RELAY-FORW zawierają wartości identyfikujące interfejsy od strony klientów (w polach *Adres łącza* i *Adres partnera* oraz w opcji *Interface-ID*): adres łącza jest zerowy, adres partnera jest adresem IP klienta, opcja *Interface-ID* konstruowana jest zgodnie z wartością ustaloną przez LDRA. Gdy LDRA odbierze komunikat RELAY-REPL z zerowym adresem łącza, przesyła go do klienta przez interfejs o identyfikatorze określonym w opcji *Interface-ID*, ustawianym przez serwer. W komunikatach RELAY-FORW otrzymywanych od klientów inkrementowany jest licznik przeskoków. Przychodzące przez interfejsy od strony sieci komunikaty inne niż RELAY-REPL są przez LDRA ignorowane.

6.2.7. Uwierzytelnianie DHCP

Niemal każdy z aspektów komunikacji sieciowej rozpatrywać można (i trzeba) także pod kątem bezpieczeństwa — w tej książce robimy to na zakończenie każdego rozdziału i nie inaczej będzie w tym rozdziale. Uznaliśmy jednak, że w kontekście wielu różnych mechanizmów protokołu DHCP warto już teraz poświęcić uwagę ich odporności na ataki i inne okoliczności zakłócające normalną pracę sieci; wadliwe funkcjonowanie tych mechanizmów może spowodować, że wiele hostów, karmionych fałszywą informacją już od momentu uruchomienia, funkcjonować będzie w sposób daleki od oczekiwań, a być może także niebezpieczny dla innych hostów.

Niestety, oryginalna specyfikacja DHCP nie zawiera żadnych elementów dedykowanych bezpieczeństwu, co stwarza furtkę do różnych działań destruktywnych (mimowolnych lub celowo sprowokowanych) ze strony klientów i serwerów. Niedostatek ten uzupełnia dokument [RFC3118] definiujący komunikaty realizujące uwierzytelnianie w ramach DHCP, wśród nich opcję *Authentication*, o formacie przedstawionym na rysunku 6.22.



Rysunek 6.22. *Opcja Authentication protokołu DHCP zawiera informację uwierzytelniającą oraz dane umożliwiające wykrywanie ataków powtarzania komunikatu. Opcja ta, zdefiniowana w roku 2001, wciąż nie jest powszechnie używana*

Jak łatwo się domyślić, celem opcji *Authentication* jest weryfikacja, czy komunikaty DHCP generowane są przez autoryzowanych nadawców. Pole *Kod* zawiera wartość 90, w polu *Długość* zapisany jest rozmiar opcji w bajtach, liczony od pierwszego bajta pola *Protokół*. Opcja ta dołączana jest do komunikatów DHCPDISCOVER i DHCPINFORM wysyłanych przez klienta oraz do komunikatów DHCPOFFER lub DHCPACK stanowiących odpowiedzi serwera. Postać *Informacji uwierzytelniającej* zależna jest od zawartości pól *Protokół*

i *Algorytm*: jeśli pola te są zerowe, informacja ta ma postać prostego *tokenu konfiguracyjnego* — jego zgodność, sprawdzana przez serwery i stacje klienckie, jest warunkiem zaakceptowania otrzymanego komunikatu. Token ten może mieć np. postać hasła czy innego łańcucha znaków o podobnym charakterze; ponieważ może zostać przechwycony przez intruza, zapewnia jedynie ograniczone bezpieczeństwo, lecz dobrze nadaje się do zapobiegania przypadkowym problemom z DHCP.

Gdy pola *Protokół* i *Algorytm* mają oba wartość 1, stosowana jest bardziej zaawansowana i bezpieczniejsza metoda, zwana *opóźnionym uwierzytelnianiem* (*deferred authentication*). Serwery i klienty weryfikują wówczas integralność przesyłanej informacji za pomocą kryptograficznego *kodu uwierzytelniania komunikatu* (*Message Authentication Code*, w skrócie MAC — patrz rozdział 18.)[8] w oparciu o sekretny klucz współdzielony przez klienta i serwer, i nieznany komukolwiek innemu. Klient i serwer zyskują tym samym pewność co do autentyczności partnera oraz gwarancję, że przesyłana informacje nie została zniekształcona w trakcie transmisji.

Opcja *Authentication* zawiera także zabezpieczenia przed atakami powtarzania komunikatów: legalny komunikat może mianowicie zostać przez intruza przechwycony, zapamiętany, a po pewnym czasie wysłany po raz drugi do oryginalnego adresata. Zniwelowanie szans powodzenia takiego przedsięwzięcia jest zadaniem dwóch kolejnych pól. Gdy pole *RDM* ma wartość 0, pole *Wykrywanie powtórzeń* zawiera licznik, od którego oczekuje się konsekwentnej inkrementacji — jego wartość w kolejnych komunikatach powinna być systematycznie coraz większa, naruszenie tej zasady świadczy o obecności powtórzonego komunikatu. Owszem, metoda ta załamuje się w sytuacji, gdy pakiety docierają do adresata w kolejności innej niż zostały wysłane przez nadawcę, sytuacja taka jest jednak wysoce nieprawdopodobna w sieciach LAN, w których między klientem a serwerem DHCP istnieje tylko jedna trasa (a właśnie w sieciach LAN funkcjonuje większość implementacji protokołów DHCP).

Uwierzytelnianie DHCP nie jest obecnie techniką szeroko rozpowszechnioną z dwóch głównych przyczyn. Po pierwsze, wymaga ona rozdysponowania sekretnych kluczy między serwery i wszystkie klienty wymagające uwierzytelniania, co może być czynnością wysoce pracochłonną. Po drugie, uwierzytelnianie to jest de facto dodatkiem do protokołu DHCP, opracowanym w czasie, gdy mnóstwo jego (protokołu) implementacji z powodzeniem funkcjonowało w dotychczasowej postaci, a większość adminów hołdowała starej sprawdzonej zasadzie „jeśli coś działa, nie waż się poprawiać". Mimo to, warto rozważyć zastosowanie uwierzytelniania do komunikacji z użyciem przekaźników DHCP (o których pisaliśmy w punkcie 6.2.6), w sposób opisany w specyfikacji [RFC4030].

6.2.8. Rozszerzenie rekonfiguracji

Zasadniczo odnowienie dzierżawy DHCP następuje z inicjatywy klienta, jednak dokument [RFC3203] definiuje alternatywę w postaci *rozszerzenia rekonfiguracji* (*reconfigure extension*) i związanego z nim komunikatu `DHCPFORCERENEW`. Rozszerzenie to pozwala serwerowi na *zmuszenie* konkretnego klienta do natychmiastowego przejścia w stan `Odnawianie` (vide maszyna stanu klienta DHCP na rysunku 6.10) i wysłanie (w innej

[8] Patrz także mój przypis dotyczący akronimu MAC w rozdziale 3. — *przyp. tłum.*

sytuacji naturalnego) komunikatu DHCPREQUEST. I nic nie stoi na przeszkodzie, by ewentualną odpowiedzią serwera na tak wymuszone żądanie klienta był komunikat DHCPNAK, zmuszający klienta do zresetowania swego statusu, czyli przejścia w stan Początek. I rozpoczęcia wszystkiego od nowa, czyli wysłania komunikatu DHCPDISCOVER.

Jak łatwo wywnioskować z powyższego, opisane rozszerzenie daje serwerowi DHCP możliwość anulowania udzielonych dzierżaw w sytuacji, gdy wymagają tego znaczące zmiany w konstrukcji sieci, spowodowane np. awarią węzła lub przeprowadzane z inicjatywy administratora. Komunikat DHCPFORCERENEW, jako wrażliwy na atak przeciążeniowy (DoS), musi być obowiązkowo uwierzytelniany, a ponieważ — jak wspominaliśmy — uwierzytelnianie DHCP nie cieszy się szczególną popularnością, więc równie mało popularne jest samo rozszerzenie rekonfiguracji.

6.2.9. Opcja Rapid Commit

Równie kategoryczny może być serwer DHCP wobec klienta inicjującego dzierżawę, czyli wysyłającego komunikat DHCPDISCOVER; zamiast wyboru, jaki daje klientowi zestaw komunikatów DHCPOFFER, serwer narzuca klientowi swoją ofertę, przesyłając proponowany adres od razu w komunikacie DHCPACK, co efektywnie oznacza pominięcie wymiany komunikatów DHCPOFFER oraz DHCPREQUEST i zredukowanie czterofazowej wymiany komunikatów do dwufazowej (patrz rysunek 6.14). Scenariusz taki jest istotą opcji *Rapid Commit*, definiowanej w dokumencie [RFC4039], a motywacją jej wynalezienia jest chęć szybkiego konfigurowania hostów często zmieniających swą sieć (czyli większości urządzeń mobilnych). Gdy dostępny jest tylko jeden serwer DHCP, dysponujący obfitym zasobem adresów do przydziału, opcja ta nie daje większych korzyści.

Klient zamierzający skorzystać z opcji *Rapid Commit* musi umieścić ją w komunikacie DHCPDISCOVER (nie może ona wystąpić w żadnym innym), serwer zaś dołącza ją do wysyłanej odpowiedzi DHCPACK (ponownie, nie ma on prawa wykorzystać jej w żadnym innym komunikacie). Klient po otrzymaniu kategorycznej odpowiedzi ma prawo się jeszcze rozmyślić, np. wskutek wykrycia zdublowania adresu za pomocą procedury ACD — powinien w tym celu wysłać komunikat DHCPDECLINE powodujący anulowanie uzyskanej dzierżawy bądź komunikat DHCPRELEASE oznaczający (w każdych okolicznościach) rezygnację z trwającej właśnie dzierżawy.

6.2.10. Informacja o lokalizacji

Wśród rozmaitych informacji konfiguracyjnych hosta często użyteczna okazuje się informacja o jego geolokalizacji, wyrażonej np. w postaci szerokości geograficznej, długości geograficznej i wysokości nad poziomem morza. Realizowany przez IETF projekt o nazwie Geoconf, zmierzający do standardowego udostępniania tej informacji, określanej popularnie skrótem LCI (od *Location Configuration Information*) znalazł swe ukoronowanie w postaci specyfikacji [RFC5225], definiującej dwie opcje DHCP: *GeoConf* (numer 123) i *GeoLoc* (numer 144). Oprócz wspomnianych współrzędnych (szerokość, długość i wysokość), na informację LCI składa się także wskaźnik dokładności (rozdzielczości), z jaką zostały one wyznaczone. LCI może być wykorzystywana w wielu zastosowaniach, np. przez ekipy ratunkowe lokalizujące telefon komórkowy, z którego została wezwana pomoc.

Oprócz lokalizacji konkretnego systemu we współrzędnych przestrzennych, nie mniej użyteczna jest informacja o jego umiejscowieniu w kategoriach podziału politycznego lub administracyjnego (*civic location*) — umiejscowieniu wyrażonym w formie kraju, miasta, dzielnicy, ulicy itp. parametrów. Związana z tym opcja `GEOCONV_CIVIC` (numer 99), definiowana w [RFC4776], wykorzystuje ten sam format informacji, co LCI, choć w związku z nią pojawia się dodatkowy problem: otóż w przeciwieństwie do współrzędnych przestrzennych, wyrażanych w postaci liczb, informacja geopolityczna charakteryzuje się znacznie bardziej zróżnicowaną formą ekspresji, nie tylko obejmującą rozmaite konwencje nazewnicze, lecz często wymagającą użycia niestandardowych alfabetów regionalnych, dalece wykraczających poza zestaw znaków języka angielskiego i symboli zwyczajowo używanych przez DHCP. Dodatkowo pojawia się problem poszanowania prywatności — zagadnienie całkowicie wykraczające poza koncepcję protokołu DHCP; IETF poświęciła temu problemowi framework o nazwie *Geopriv* — zainteresowanych nim czytelników odsyłamy do dokumentu [RFC3693].

Alternatywą dla opisanego rozwiązania może być wykorzystywanie innego protokołu warstwy wyższej dedykowanego geolokalizacji — *HTTP-Enabled Location Delivery*, w skrócie HELD. Zamiast bezpośredniego kodowania LCI w komunikatach DHCP, opcje `OPTION_V4_ACCESS_DOMAIN` (numer 213) i `OPTION_V6_ACCESS_DOMAIN` (numer 57), definiowane w [RFC5986], dostarczają kwalifikowaną nazwę domeny (FQDN) serwerów HELD dla danej lokalizacji, w wersji (odpowiednio) IPv4 i IPv6.

Gdy host rozpozna swą lokalizację, może żądać związanych z nią usług, np. udostępnienia adresu najbliższego szpitala. Umożliwia mu to opracowany przez IETF framework o nazwie *Location-to-Service Translation*, w skrócie LoST ([RFC5222], za pośrednictwem protokołu warstwy aplikacji wykorzystującego specyficzny lokalizator URI. Opcje DHCP `OPTION_V4_LOST` (numer 137) i `OPTION_V6_LOST` (numer 51) (patrz [RFC5223]) dostarczają FQDN (zakodowanych standardowo — patrz rozdział 11.) dla serwerów (odpowiednio) DHCPv4 i DHCPv6 implementujących LoST.

6.2.11. Informacje dla urządzeń mobilnych (MoS i ANDSF)

Coraz większa popularność komputerów przenośnych i smartfonów uzyskujących dostęp do Internetu za pośrednictwem technologii komórkowej nie mogła nie znaleźć odzwierciedlenia w postaci stosownych frameworków i odpowiednich opcji DHCP. Przesyłaniu informacji między różnymi sieciami bezprzewodowymi na potrzeby urządzeń mobilnych dedykowane są obecnie dwa zestawy tych opcji: IEEE 802.11 *Mobility Services* (MoS) *Discovery* oraz *Access Network Discovery and Selection Function* (ANDSF). Drugi z wymienionych frameworków znajduje się w fazie zabiegów standaryzacyjnych, prowadzonych przez organizację o nazwie 3GPP (*3rd Generation Partnership Project* — „projekt partnerstwa 3. generacji") zajmującą się standardami dotyczącymi przesyłania danych w sieciach komórkowych. Z kolei standard IEEE 802.21 (patrz [802.21-2008]) jest specyfikacją niezależnego od nośnika frameworku dla urządzeń przenośnych (*Media Independent Handoff*, w skrócie MIH). Framework ten dedykowany jest różnym usługom, świadczonym między sieciami różnych typów, definiowanym przez IEEE (802.3, 802.11. 802.16), przez 3GPP i przez 3GPP2.

IETF opisuje projektowanie frameworku tego typu w dokumencie [RFC5677]. MoS udostępnia trzy rodzaje usług: informacyjne (*information services*), poleceniowe (*command services*) i zdarzeniowe (*event services*). Mówiąc ogólnie, usługi te dostarczają informacji o dostępnych sieciach, zapewniają sterowanie parametrami łącza i realizują powiadamianie o zmianach w statusie łącza. Opcje DHCP *MoS Discovery* ([RFC5678]) umożliwiają węzłowi mobilnemu uzyskiwanie adresów IP (lub nazw domen) serwerów świadczących wymienione usługi, w obu wersjach protokołu. Dla DHCPv4 opcje `OPTION-IPv4_Address-MoS` (numer 139) i `OPTION-IPv4_FQDN-MoS` (numer 140) zawierają wektory podopcji reprezentujących (odpowiednio) adresy IP i nazwy FQDN wspomnianych serwerów, w DHCPv6 ich odpowiednikami są opcje `OPTIONIPv6_Address-MoS` (numer 54) i `OPTION-IPv6_FQDN` (numer 55).

Na bazie specyfikacji ANDSF autorstwa 3GPP dokument [RFC6153] definiuje opcje DHCPv4 i DHCPv6 związane z przenoszeniem informacji związanej z ANDSF, a konkretnie — opcje umożliwiające urządzeniom mobilnym uzyskiwanie adresów serwerów ANDSF. Serwery te konfigurowane są przez operatorów komórkowych i oferuję informacje dotyczące m.in. dostępności i polityki dostępu do sieci z wieloma technologiami transportowymi, np. sieciami równolegle wykorzystującymi 3G i Wi-Fi.

Opcja `ANDSF IPv4 Address` (numer 142) zawiera listę adresów IPv4 serwerów ANDSF uporządkowaną w kolejności malejących preferencji ich wykorzystywania; analogiczna lista adresów IPv6 jest przedmiotem opcji `ANDSF IPv6 Address` (numer 143). W DHCPv4 węzeł żądający informacji ANDSF umieszcza opcję `ANDSF IPv4 Address` na liście parametrów żądania (*Parameter Request List*), w protokole DHCPv6 opcja `ANDSF IPv6 Address` musi zostać włączona do opcji *Option Request Option* (ORO) (patrz sekcja 22.7 dokumentu [RFC3315]).

6.2.12. Podsłuchiwanie DHCP

„Podsłuchiwanie" ruchu DHCP to opcja oferowana przez niektórych producentów przełączników, polegająca na inspekcji zawartości komunikatów DHCP i eliminowaniu tych, które zawierają adresy niefigurujące na liście kontroli dostępu. Technika ta pozwala uporać się z dwoma problemami. Po pierwsze, poważnie ogranicza potencjalne szkody, jakie mogłyby wyrządzić hosty złośliwie podszywające się pod serwery DHCP — podstępne oferty fałszywych adresów po prostu nie docierają do klientów. Po drugie, wybrane adresy MAC można w ten sposób pozbawić możliwości kojarzenia z adresami IP; ponieważ jednak adresy MAC można łatwo zmieniać za pomocą poleceń systemowych, jest to tylko ochrona połowiczna.

6.3. Bezstanowe konfigurowanie adresów (SLAAC)

Podczas gdy większości routerów adresy przydzielane są manualnie, hostom przyporządkowuje się zwykle adresy na kilka sposobów: manualnie, za pomocą specjalizowanego protokołu (DHCP) lub automatycznie według pewnego algorytmu. Ta ostatnia możliwość ma dwie odmiany, zależnie od tego, jakiego typu adres jest przydzielany: jeśli jest to adres używany tylko na jednym łączu (czyli lokalny dla łącza), dla hosta wystarczające będzie znalezienie jakiegokolwiek adresu nieużywanego aktualnie na tym łączu; jeśli

jednak jest to adres o znaczeniu globalnym, wykorzystywany do łączności ze światem zewnętrznym, przynamniej jego część musi być ustalona według pewnych ograniczeń czy zasad. Taki mechanizm zautomatyzowanego ustalania swych adresów przez host nazywa się *bezstanowym autokonfigurowaniem adresów* (*Stateless Address Autoconfiguration*, w skrócie SLAAC) i występuje w dwóch wersjach, dla adresów IPv4 i dla adresów IPv6.

6.3.1. Dynamiczne konfigurowanie adresów IPv4 lokalnych dla łącza

Gdy host przyłącza się do sieci pozbawionej serwerów DHCP, a jego adresy IP nie zostały skonfigurowane manualnie, musi adresy te wygenerować we własnym zakresie, by możliwa była jego komunikacja z innymi węzłami na bazie protokołu IP. W dokumencie [RFC3927] opisywana jest metoda automatycznego wyboru przez host adresu IPv4 z przedziału od `169.254.1.1` do `169.254.254.254` przy masce podsieci `255.255.0.0` (patrz [RFC5735] oraz tabela 2.7 w rozdziale 2.). Metoda ta nazywana jest dynamicznym konfigurowaniem adresów lokalnych dla łącza lub *automatycznym prywatnym adresowaniem IP* (*Automatic Private IP Addressing*, w skrócie APIPA). Host wybiera po prostu losowo adres z podanego przedziału, po czym sprawdza (za pomocą procedury ACD opisywanej w rozdziale 4.), czy adres ten nie jest już używany przez inny węzeł w podsieci.

6.3.2. Procedura SLAAC dla adresów IPv6 lokalnych dla łącza

Technika SLAAC dla adresów IPv6 opisana jest w dokumencie [RFC4862]. Jej istotą jest automatyczne (lub autonomiczne) uzyskiwanie przez węzły adresów IPv6, przebiegające dla każdego węzła w trzech następujących fazach: uzyskanie adresu lokalnego dla łącza, uzyskanie globalnego adresu za pomocą konfiguracji bezstanowej i zweryfikowanie unikatowości adresu lokalnego dla łącza. Formowanie adresu globalnego odbywa się przy udziale routera — adres ten generowany jest jako kombinacja prefiksu ogłaszanego przez router oraz pewnej informacji generowanej w sposób lokalny; możliwe jest pominięcie tej fazy i generowanie jedynie adresu lokalnego (np. w sytuacji, gdy routery nie są dostępne).

SLAAC może być także używane w połączeniu z DHCPv6 dostarczającym dodatkowych informacji nieadresowych (jest to tzw. bezstanowy DHCPv6 — *stateless DHCPv6*) lub z manualnym konfigurowaniem. Hosty wykorzystujące SLAAC mogą współistnieć w tej samej sieci z hostami korzystającymi z DHCPv6. Generalnie DHCPv6 w trybie „stanowym" (*stateful*) zapewnia lepszą kontrolę nad przydziałem adresów, jednak kombinacja SLAAC z bezstanowym DHCPv6 wydaje się opcją wygodniejszą dla użytkowników i administratorów.

Próbne (*tentative*) i optymistyczne (*optimistic*) adresy IPv6 (patrz podpunkt 6.2.5.1) wybierane są za pomocą procedur opisywanych w dokumentach [RFC4291] i [RFC4941]. Procedury te przydatne są jedynie w sieciach obsługujących multicasting, a generowane przez nie adresy po przejściu do stanu preferencyjnego pozostają w nim bezterminowo (okres preferencji, a w konsekwencji okres ważności ustalane są jako nieskończone). Konstruowanie adresu opiera się na „dobrze znanym" prefiksie `fe80::0`: pierwsze 10 bitów adresu otrzymuje postać `1111111010` (czyli `fe80::/10` — patrz tabela 2.8 w rozdziale 2.),

ostatnie N bitów ustawiane jest przez host[1], pozostałe bity zostają wyzerowane. Tak skonstruowany adres otrzymuje status „próbny" lub „optymistyczny" i poddawany jest procedurze DAD weryfikującej jego unikatowość.

6.3.2.1. Wykrywanie zdublowania adresów IPv6 (DAD)

Celem procedury DAD (to skrót od *Duplicate Address Detection — wykrywanie zdublowania adresu*) jest zweryfikowanie, czy dany adres IPv6 o statusie „próbny" lub „optymistyczny" jest na danym łączu unikatowy, czyli nie jest używany przez żaden inny węzeł. Dla uproszczenia ograniczymy się tu do adresów próbnych, w odniesieniu do adresów optymistycznych procedura DAD ma podobny przebieg. Dokument [RFC4862] definiujący tę procedurę zaleca jej konsekwentne stosowanie dla każdego nowo przydzielanego adresu — czy to przy konfiguracji manualnej, czy przy użyciu autokonfiguracji, czy też w ramach autokonfiguracji.

Test na unikatowość adresu jest w swym pomyśle dość prosty: węzeł badający unikatowość wysyła do swej sieci komunikat ICMPv6 *Neighbor Solicitation* w nadziei otrzymania potwierdzenia w postaci komunikatu *Neighbor Advertisement* (oba komunikaty opisujemy w rozdziale 8.) — otrzymanie takiego potwierdzenia oznacza *istnienie* przedmiotowego adresu, konieczność skonstruowania innego i powtórzenia całej procedury; jego nieotrzymanie uprawnia do nadania testowanemu adresowi statusu preferencyjnego.

Fizycznie procedura DAD realizowana jest następująco: węzeł dołącza się do grupy multicast wszystkich węzłów (*All Nodes*) oraz grupy multicast *Solicited-Node* dla testowanego adresu (patrz rozdział 9.). Następnie wysyła jeden lub kilka komunikatów ICMPv6 *Neighbor Solicitation*; adresem źródłowym w tych komunikatach jest adres nieokreślony ::/128, czyli w całości zerowy (ponownie patrz tabela 2.8), zaś adresem docelowym — wspomniany adres multicast *Solicited-Node* dla testowanego adresu; sam testowany adres zapisany jest w polu *Adres docelowy*. Jeśli w odpowiedzi nadejdzie komunikat *Neighbor Advertisement*, procedura kończy się odrzuceniem testowanego adresu.

W rezultacie dołączania węzłów do grup multicast generowane są komunikaty MLD (patrz rozdział 9.), lecz (zgodnie z wymogami [RFC4862]) transmisja każdego z nich zostaje opóźniona o losowy interwał czasu. Celem tego zabiegu jest zapobieżenie nagłemu przeciążeniu sieci spowodowanemu zmasowanym atakiem takich komunikatów podczas równoczesnego dołączania wielu węzłów do grupy multicast *All Hosts*, co może mieć miejsce w przypadku wznowienia zasilania sprzętu po awarii. Z perspektywy DAD komunikaty MDL przygotowują „podsłuchujące" przełączniki (patrz punkt 6.2.12) na nadejście ruchu multicast.

Drugim istotnym elementem procedury DAD jest wyczulenie węzła na odbierane od innych węzłów komunikaty *Neighbor Solicitation*, dotyczące *tego samego adresu*: jeśli mianowicie w trakcie testowania danego adresu węzeł stwierdzi, że identyczny adres jest przedmiotem procedury DAD w innym węźle, po prostu poddaje się *pro publico bono*, dając większą szansę konkurentowi (konkurentom).

Gdy — wskutek którejś z opisanych przyczyn — procedura DAD załamie się, dyskwalifikując testowany adres, węzeł staje przed koniecznością skonstruowania innego adresu

[1] Zgodnie z dokumentem RFC4291, $N = 64$, a końcowe bity adresu są tożsame z identyfikatorem interfejsu; w podpunkcie 6.3.2.3 przedstawiona jest jednak alternatywa dla tej reguły — *przyp. tłum.*

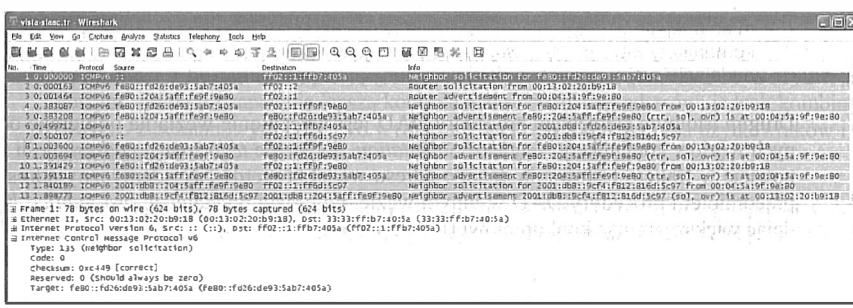
i ponownego uruchomienia procedury DAD. Sęk w tym, że jeżeli generowanie adresów odbywa się w oparciu o adres MAC węzła, to ma charakter deterministyczny i prawie na pewno ponowne generowanie adresu da ten sam wynik, co poprzednio; w takiej sytuacji do akcji musi wkroczyć administrator, proponując explicite nowy adres próbny. Wspomniany determinizm będzie mniej prawdopodobny, jeśli generowanie adresu odbywać się będzie w oparciu o identyfikator interfejsu tworzony na bardziej elastycznych zasadach.

6.3.2.2. Procedura SLAAC dla globalnych adresów IPv6

Po uzyskaniu adresu lokalnego dla łącza węzeł prawdopodobnie rozpocznie starania o uzyskanie adresu globalnego. Procedura pozyskiwania adresu globalnego przebiega podobnie jak w przypadku adresu lokalnego dla łącza, jednak zamiast predefiniowanego prefiksu `feB0::/10` używany jest prefiks dostarczany przez router, zawarty w opcji *Prefix* komunikatu *Router Advertisement* (patrz rozdział 8.) — ustawienie odpowiedniego znacznika opcji sygnalizuje takie właśnie wykorzystanie prefiksu. Dalsza część procedury przebiega podobnie — na końcowe bity tworzonego adresu wpisywany jest identyfikator interfejsu, pozostałe bity zostają wyzerowane. W ramach wspomnianej opcji dostarczana jest także informacja umożliwiająca ustalenie okresu preferencji i okresu ważności generowanego adresu.

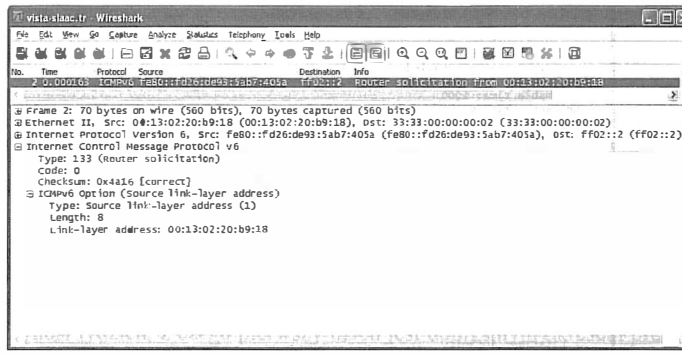
6.3.2.3. Przykład

Okno programu Wireshark widoczne na rysunku 6.23 przedstawia serię zdarzeń zachodzących w ramach hosta (z systemem Windows Vista + SP1) próbującego uzyskać adres za pomocą SLAAC. System wybiera najpierw adres lokalny dla łącza w oparciu o prefiks `feB0::/64` i liczbę pseudolosową. Oryginalnie do wspomnianego prefiksu dołączany był adres MAC hosta, zgodnie ze specyfikacją [RFC4291]; wykorzystanie w zamian wartości losowej powoduje, że adres hosta zmienia się przy kolejnych przydziałach, co pozwala lepiej chronić prywatność jego użytkownika. W rezultacie procedura DAD wykonywana jest w stosunku do adresu `feB0::fd26:de93:5ab7:405a` — widzimy wysłanie przez host komunikatu *Neighbor Solicitation* (NS) i komunikatu *Router Solicitation* (RS) — indagowanie routerów ma na celu uzyskanie prefiksu bazowego dla globalnego adresu.



Rysunek 6.23. *W ramach protokołu SLAAC host rozpoczyna procedurę DAD weryfikującą unikatowość adresu lokalnego dla łącza, wysyłając komunikat ICMPv6 Neighbor Solicitation; jako adres źródłowy komunikatu wykazywany jest adres nieokreślony (zerowy), bo nadawca nie posiada jeszcze adresu IP, którego mógłby użyć w tej roli*

Komunikat RS, szczegółowo uwidoczniony w oknie na rysunku 6.24, wysyłany jest na adres multicast `ff02::2` reprezentujący wszystkie routery (*All Routers*); w charakterze adresu źródłowego wykorzystywany jest testowany (próbny) adres `fe80::fd26:de93:5ab7:405a`. Przychodzący komunikat RA, jako odpowiedź na RS, szczegółowo przedstawiony na rysunku 6.25, wysłany został na adres multicast `ff02::1` reprezentujący wszystkie systemy (*All Systems*), z routera o lokalnym adresie `fe80::204:5aff:fe9f:9e80`. Obecne w nim znaczniki, definiowane w dokumencie [RFC5175], są wszystkie wyzerowano, co oznacza brak na tym łączu usług DHCPv6 w zakresie konfigurowania adresów IP.



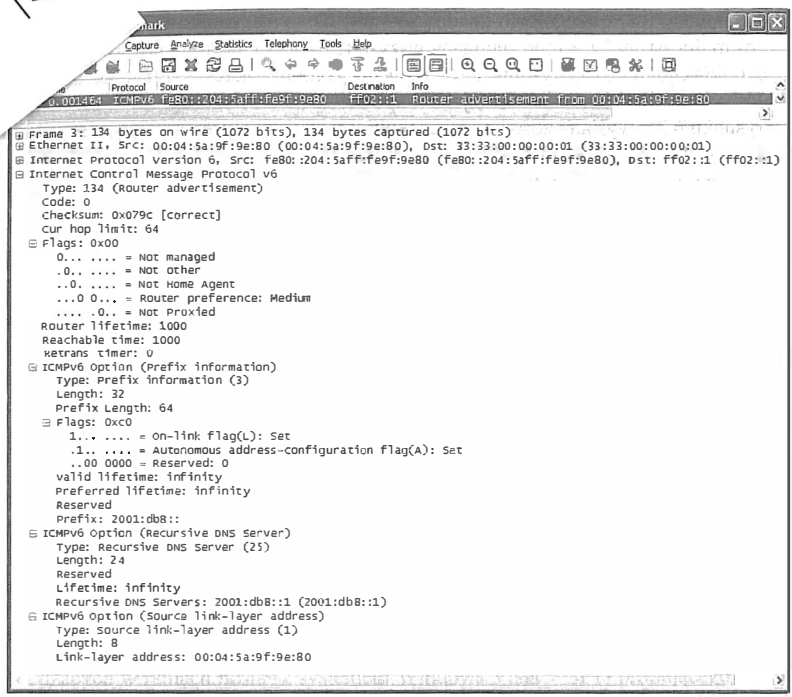
Rysunek 6.24. *Komunikat ICMPv6 Neighbor Solicitation jako żądanie udostępnienia informacji konfiguracyjnej, m.in. prefiksu bazowego dla globalnego adresu IP*

W opcji *Prefix* widzimy udostępniony przez router prefiks `2001:db8::/64` służący do konstruowania adresów globalnych dla węzłów na tym łączu. Długość prefiksu (64) nie jest jawnie zapisywana, bo wynika bezpośrednio ze specyfikacji [RFC4291]. Wartość *Znaczników* (`0xc0`) oznacza ustawienie dwóch bitów świadczących o tym, że prefiks jest „na łączu" (określenie to oznacza, że może być wykorzystywany przy połączeniach z routerem) oraz że może być używany przez host jako baza do automatycznego tworzenia adresów.

Opcja *Recursive DNS Server* (RDNSS), definiowana w [RFC6106], oznajmia, że serwer DNS dostępny jest pod adresem `2001::db8::1`.

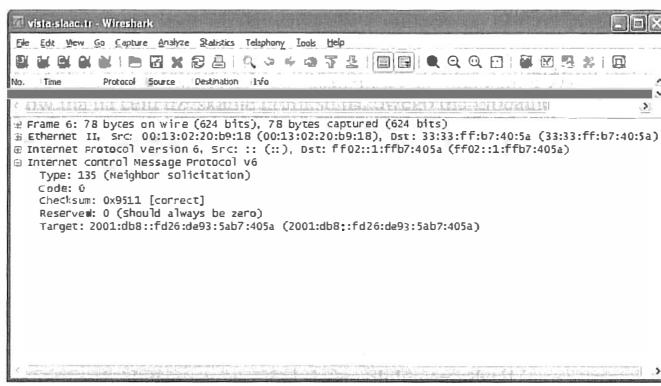
Cytowana już wcześniej opcja *Source Link-Layer Address Option* (SLLAO) wskazuje `00:04:5a:9f:9e:80` jako adres MAC routera. Informacja ta wykorzystywana jest przez węzły do uaktualniania swych tablic *Neighbor Cache* — w IPv6 są to odpowiedniki tablic skojarzeniowych protokołu ARP, opisujemy je szczegółowo w rozdziale 8.

Po wymianie komunikatów *Neighbor Solicitation* i *Neighbor Advertisement* między klientem a routerem klient rozpoczyna kolejną procedurę DAD w celu zweryfikowania adresu globalnego (patrz rysunek 6.26). Adres ten — `2001:db8::fd26:de93:5ab7:405a` — skonstruowany został na bazie prefiksu udostępnionego przez router (`2001::db8`) i tej samej liczby pseudolosowej, która użyta została do utworzenia adresu lokalnego dla łącza. Dzięki temu adres multicast *Solicited-Node* wykorzystywany przez procedurę DAD jest taki sam jak poprzednio (`ff02::1:ffb7:405a`).

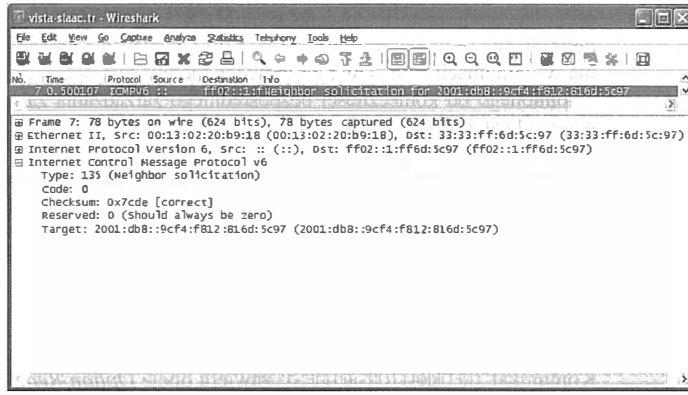


Rysunek 6.25. *Komunikat ICMPv6 Router Advertisement zawierający informację na temat domyślnego routera oraz prefiks dla globalnego adresu IP. Dodatkowo dostępny jest adres serwera DNS oraz wskazanie, czy router może pełnić rolę agenta domowego w mobilnym IPv6 (nie w tym przypadku)*

Rysunek 6.26. *Procedura DAD weryfikująca unikatowość globalnego adresu opartego na prefiksie 2001:db8::/64; komunikat wysyłany jest na ten sam adres multicast Solicted-Node, co pierwszy pakiet*



To jeszcze nie koniec: klient dysponujący już jednym globalnym adresem przystępuje do wygenerowania drugiego, tymczasowego, na bazie tego samego prefiksu, lecz — oczywiście — przy użyciu innej wartości pseudolosowej. Skonstruowany w ten sposób adres 2001:db8::9cf4:f812:816d:5c97 poddawany jest weryfikacji przez procedurę DAD. Związany z tym komunikat *Neighbor Solicitation* uwidoczniony został na rysunku 6.27.



Rysunek 6.27. *Procedura DAD dla adresu 2001:db8::9cf4:f812:816d:5c97*

Sens używania tymczasowych adresów globalnych IPv6 zasadza się w polepszeniu ochrony prywatności ich użytkowników. Adresy tymczasowe mają wyraźnie krótszy czas życia od swych trwałych odpowiedników; czas preferencji i czas ważności przyjmowane są jako mniejsza z dwóch wartości: tej przekazanej w opcji *Prefix* i tej przyjętej lokalnie jako domyślna. W systemie Windows Vista domyślny okres ważności adresu tymczasowego wynosi tydzień, z czego pierwszy dzień jest domyślnym okresem preferencyjnym.

Wygenerowanie i zweryfikowanie trzech adresów — lokalnego i dwóch globalnych — to już całość procesu SLAAC. Wspomniane adresy są wystarczające do komunikacji zarówno lokalnej, jak i globalnej; tymczasowy adres globalny zmienia się periodycznie, co utrudnia śledzenie hostów na podstawie ich numerów IPv6. Jednak w sytuacji, gdy tymczasowe adresy globalne są dla danego hosta zbędne, można wyłączyć ich generowanie. W systemie Windows robi się to za pomocą polecenia:

```
C:\> netsh interface ipv6 set privacy state=disabled
```

W Linuksie generowanie adresów tymczasowych jest domyślnie wyłączone; można je włączyć za pomocą następującego ciągu poleceń:

```
Linux# sysctl -w net.ipv6.conf.all.use_tempaddr=2
Linux# sysctl -w net.ipv6.conf.default.use_tempaddr=2
```

Wyłączenie adresów tymczasowych następuje w wyniku użycia ciągu poleceń:

```
Linux# sysctl -w net.ipv6.conf.all.use_tempaddr=0
Linux# sysctl -w net.ipv6.conf.default.use_tempaddr=0
```

6.3.2.4. Bezstanowy DHCP

Przydzielanie klientom adresów IP przez serwer DHCP wiąże się z koniecznością utrzymywania informacji o stanie przydziału dla każdego klienta z osobna, dlatego też serwer funkcjonujący w tym trybie określany bywa przymiotnikiem „stanowy" (*stateful*). Serwery DHCPv6 mogą funkcjonować również w trybie *bezstanowym* (*stateless*) — nie zarządzają wówczas przydziałami adresów, lecz jedynie udostępniają pomocnicze informacje konfiguracyjne; zgodnie z nazwą, nie wiąże się to z utrzymywaniem informacji

o stanie, czyli historii dotychczasowych udostępnień. Bezstanowe serwery DHCPv6 definiowane są w dokumencie [RFC3736] i z założenia przeznaczone są do funkcjonowania w kombinacji z SLAAC, co prawdopodobnie jest opcją atrakcyjną dla administratora, uwolnionego od konieczności bezpośredniego zarządzania zasobami adresów (nieodzownego w DHCPv4).

Bezstanowy tryb pracy serwera DHCPv6 oznacza jego uwolnienie od funkcji zarządzania adresami, czyli od implementacji wielu komunikatów wymienionych w tabeli 6.1 i opcji niezbędnych do ustanawiania wiązań IA. Upraszcza to oprogramowanie zarządzające serwerem i ułatwia jego konfigurowanie. Funkcjonowanie przekaźników DHCP pozostaje jednak bez zmian.

Żądanie klienta zwracającego się do bezstanowego serwera DHCPv6 ma postać komunikatu `INFORMATION-REQUEST`, odpowiedź serwera ma tradycyjną postać komunikatu `REPLY`. Komunikat `INFORMATION-REQUEST` zawiera opcję *Option Request Option*, wymieniającą poszczególne kategorie informacji żądanej przez klienta; komunikat ten może zawierać także identyfikator klienta (w postaci opcji *Client Identifier*), co pozwala na dostosowywanie odpowiedzi do specyfiki konkretnych klientów.

Od bezstanowego serwera DHCPv6 wymaga się implementowani komunikatów `INFORMATION-REQUEST`, `REPLY`, `RELAY-FORW` i `RELAYREPL` oraz opcji *Option Request*, *Status Code*, *Server Identifier*, *Client Message*, *Server Message* i *Interface-ID*; trzy ostatnie są niezbędne do współpracy z przekaźnikami DHCP. Aby serwer prezentował choć minimalną użyteczność, wskazane jest także implementowanie opcji *DNS Server*, *DNS Search List* i (ewentualnie) *SIP Servers*. Wśród opcji polecanych, choć niewymaganych, wymienić można także *Preference*, *Elapsed Time*, *User Class*, *Vendor Class*, *Vendor-Specific Information*, *Client Identifier* i *Authentication*.

6.3.2.5. Przydatność autokonfiguracji

Praktyczna użyteczność autokonfiguracji adresów IP jest poważnie ograniczona przez fakt, że router znajdujący się w tej samej sieci, co klient, skonfigurowany jest na używanie określonego zakresu adresów IP, nieobejmującego zwykle adresów, jakie klient przydziela sobie w ramach autokonfiguracji. Dotyczy to szczególnie APIPA — jest mało prawdopodobne, by router wykorzystywał adresy o prefiksie `169.254/16`. W konsekwencji autokonfigurowany klient będzie bezproblemowo współpracował z innymi węzłami w swej podsieci, natomiast prawdopodobne problemy pojawią się przy próbie skorzystania z usługi DNS. Ponieważ Internet bez DNS-u to mało komfortowe doświadczenie dla użytkownika, z dwojga złego lepiej więc uniemożliwić klientowi nabycie adresu IP w drodze autokonfiguracji (czego efekt może być względnie łatwo wykryty), niż stwarzać mu iluzję rzeczywistej użyteczności.

Obok najpopularniejszego DNS istnieją inne usługi nazw przydatne do adresowania lokalnego dla łącza, m.in. *Bonjour/Zeroconf* firmy Apple (również w wersji dla Windows) oraz LLMNR (*Link-Local Multicast Name Resolution*) i NetBIOS firmy Microsoft. Jako że są to produkty niezależnych firm, nieobjęte zabiegami standaryzacyjnymi IETF, różnią się wyraźnie między sobą szczegółami funkcjonowania. Do rozwiązań alternatywnych dla DNS powrócimy jeszcze w rozdziale 11.

Możliwe jest generalnie wyłączenie APIPA, brakuje jednak standardowych poleceń realizujących to zadanie i konieczna jest bezpośrednia interwencja użytkownika (administratora) w ustawienia systemu. W systemie Windows należy utworzyć klucz rejestru:

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\IPAutoconfigurationEnabled
```

i przypisać mu wartość domyślną typu REG_DWORD o wartości 0. Spowoduje to wyłączenie autokonfiguracji APIPA na wszystkich interfejsach.

W Linuksie identyczny efekt osiąga się przez wpisanie (lub zmodyfikowanie) dyrektywy:

```
NOZEROCONF=yes
```

w pliku */etc/sysconfig/network*. Można także wyłączyć APIPA selektywnie na wskazanym interfejsie, wpisując wspomnianą dyrektywę do pliku konfiguracyjnego tegoż interfejsu, np. do pliku */etc/sysconfig/network-scripts/ifcfg-eth0* zawierającego ustawienia dla pierwszego interfejsu ethernetowego.

W przypadku IPv6 SLAAC łatwo uzyskać globalny adres IPv6, lecz jego powiązanie z nazwą nie jest zabezpieczone, co prowadzić może do różnych nieprzyjemnych konsekwencji (o których piszemy w rozdziałach 11. i 18.). Stanowi to przesłankę na rzecz unikania (mimo wszystko) wdrażania SLAAC. Zablokowanie SLAAC dla globalnych adresów IPv6 osiągnąć można dwojako. Pierwszy sposób polega na takim zaaranżowaniu wysyłanych przez lokalny router komunikatów *Router Advertisement*, by wyłączony był znacznik zezwalający na użycie udostępnianego prefiksu do celów autokonfiguracji; bardziej radykalnym posunięciem jest całkowita rezygnacja z oferowania opcji *Prefix*. Drugi sposób to wyłączenie autokonfiguracji globalnych adresów IP w lokalnych ustawieniach klienta.

W linuksowym hoście wyłączenie SLAAC następuje w wyniku użycia polecenia:

```
Linux# sysctl -w net.ipv6.conf.all.autoconf=0
```

W systemach Mac OS i FreeBSD podobny efekt (przynajmniej w odniesieniu do lokalnych adresów łącza) uzyskać można za pomocą polecenia:

```
FreeBSD# sysctl -w net.inet6.ip6.auto_linklocal=0
```

W systemie Windows wyłączenie autokonfiguracji jest jedną z funkcji programu netsh:

```
C:\> netsh
netsh> interface ipv6
netsh interface ipv6> set interface <nazwa interfejsu> managedaddress=disabled
```

Ponieważ szczegóły funkcjonowania przedstawionych poleceń mogą zmieniać się w kolejnych wersjach (czy nawet kolejnych aktualizacjach) odnośnych systemów operacyjnych, przed użyciem konkretnego polecenia wskazane jest dokładne zapoznanie się z opisem jego składni i semantyki.

6.4. Współdziałanie DHCP i DNS

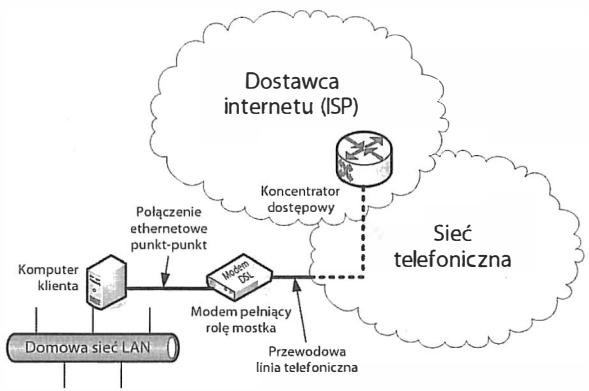
Jedną z najistotniejszych dla klienta informacji konfiguracyjnych jest niewątpliwie adres IP serwera DNS. Dzięki usłudze DNS klient uwolniony zostaje od bezpośredniego żonglowania adresami IP na rzecz wygodnych nazw mnemonicznych; bez tej — lub analogicznej — usługi klientowi znacznie trudniej byłoby się poruszać nie tylko po Internecie, ale być może także po własnej sieci (vide domena .home w prezentowanych wcześniej przykładach).

Ponieważ manualne mapowanie nazw mnemonicznych na adresy IP nie jest czynnością szczególnie twórczą, wygodnie jest łączyć przydzielenie adresów przez DHCP z metodami automatycznego aktualizowania informacji DNS związanej z tymi adresami. Cel ten realizuje się bądź to za pomocą usługi *dynamicznego DNS* (o której pisać będziemy w rozdziale 11.), bądź za pomocą „kombinowanego" serwera DHCP/DNS.

Taki kombinowany serwer — którego przykładem jest linuksowy pakiet `dnsmasq` — to program, który można skonfigurować w ten sposób, by na podstawie identyfikatora klienta i (lub) nazwy domeny zawartych w żądaniu `DHCPREQUEST` (dotyczącym przydziału adresu) uaktualniał wewnętrzną serwerową bazę DNS, jeszcze przed wysłaniem odpowiedzi `DHCPACK` niosącej przydzielony adres. Dzięki temu wszystkie systemy korzystające ze wspomnianego serwera jako klienty DHCP (choć niekoniecznie) mogą szybko uzyskiwać aktualne adresy IP innych klientów, na podstawie nazw mnemonicznych tychże klientów.

6.5. PPP przez Ethernet (PPPoE)

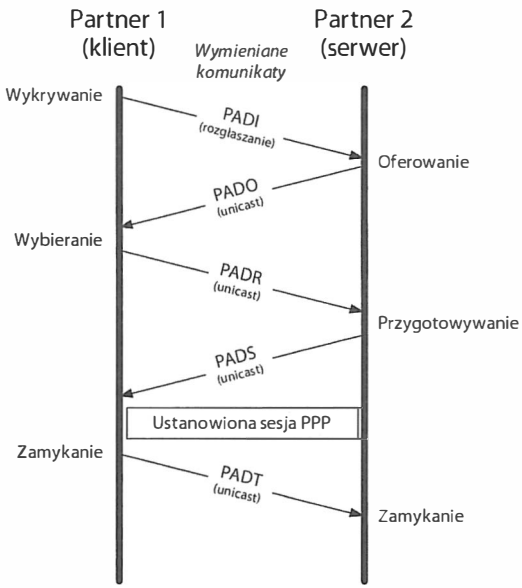
DHCP jest najczęściej używanym mechanizmem konfigurowania systemów klienckich w sieciach LAN i niektórych sieciach WAN, jednakże dla większości sieci WAN — takich jak np. DSL (*Digital Subscriber Line* — cyfrowa linia abonencka) — częściej wykorzystywana jest inna metoda, stanowiąca połączenie protokołu PPP z Ethernetem (patrz rozdział 3.), zwana *PPP przez Ethernet* (*PPP over Ethernet*, w skrócie PPPoE). Metoda ta stosowana jest w sytuacji, gdy urządzenie łączące klienta z siecią WAN (np. modem DSL) pełni rolę przełącznika lub mostka, nie routera. PPP jest preferowanym środkiem ustanawiania połączenia między klientem a dostawcą Internetu (ISP), ponieważ umożliwia bardziej drobiazgową kontrolę nad konfiguracją i bardziej szczegółowy audyt niż inne narzędzia konfiguracyjne, także DHCP. Aby komputer klienta mógł w tej konfiguracji łączyć się z Internetem, musi implementować obsługę adresów IP oraz ich trasowanie. Przykład prostej — i jednocześnie typowej — konfiguracji PPPoE przedstawiono na rysunku 6.28.



Rysunek 6.28. *Uproszczony schemat (z perspektywy klienta) usługi DSL wykorzystującej protokół PPPoE. Domowy komputer PC implementuje protokół PPPoE i uwierzytelnia użytkownika wobec ISP. Komputer ten może pełnić także role routera, serwera DHCP, serwera DNS i (lub) urządzenia NAT w sieci domowej*

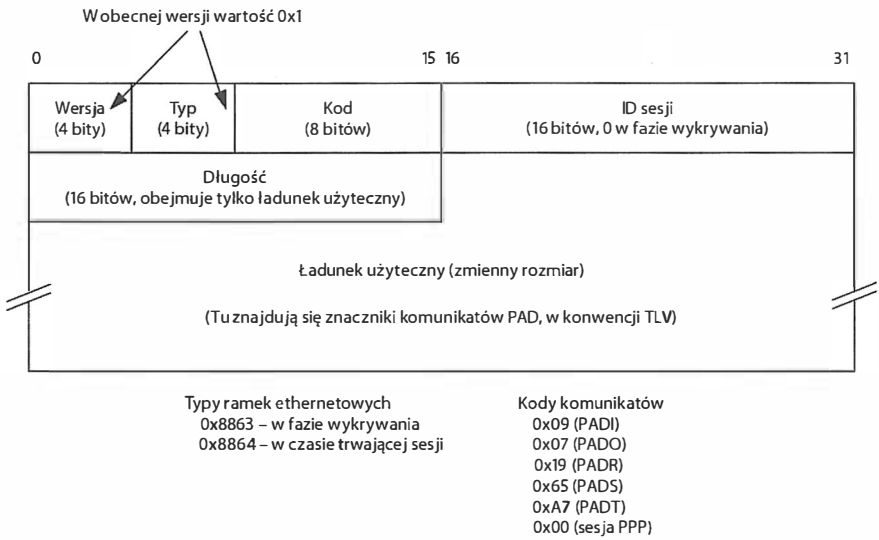
Na rysunku tym przedstawiono połączenie dostawcy Internetu z jednym z wielu klientów. Cyfrowe łącze DSL zrealizowane zostało na bazie tradycyjnej przewodowej linii telefonicznej (niekiedy określanej nostalgicznie mianem „starej poczciwej telefonii" — *Plain Old Telephone Service*, w skrócie POTS). Analogowy sygnał rozmowy współegzystuje (na zasadzie multipleksowania częstotliwości) na tej linii z cyfrowym sygnałem DSL o znacznie wyższej częstotliwości; ponieważ tradycyjne telefony nie są konstrukcyjnie przygotowane na obecność tego sygnału, który mógłby zakłócać ich pracę (i jakość rozmowy telefonicznej), podłączane są do wspomnianej linii za pomocą specjalnego filtra, który ten sygnał zatrzymuje. Modem DSL pełni rolę mostka na porcie *koncentratora dostępowego* (AC — *Access Concentrator*) łączącego linię telefoniczną abonenta z wyposażeniem dostawcy. Modem i koncentrator implementują także protokół PPPoE, którym klient zarządza z poziomu swego komputera, połączonego z modemem na zasadzie punkt-punkt za pomocą kabla ethernetowego.

Po pomyślnym ustanowieniu połączenia w warstwie niższej między modemem DSL a dostawcą Internetu komputer klienta rozpoczyna wymianę (z modemem) komunikatów protokołu PPPoE — jej scenariusz przedstawiono na rysunku 6.29, zgodnie z dokumentem [RFC2516] (dokument ten nie definiuje jednak żadnego standardu, lecz ma status *informational*).



Rysunek 6.29. *Scenariusz funkcjonowania protokołu PPPoE, rozpoczynający się od etapu wykrywania i prowadzący do ustanowienia sesji PPP. Odpowiedzią na komunikat PADI jest oferta zawarta w komunikacie PADO. Komunikat PADR wyraża wybór serwera dokonany przez klienta spośród oferowanych serwerów. Komunikat PADS zawiera potwierdzenie nawiązania sesji wysłane przez wybrany serwer. Rozpoczęta właśnie sesja trwa aż do wysłania komunikatu PADT lub zerwania połączenia w warstwie łącza danych*

Ustanowienie sesji PPP następuje w drodze wymiany czterech komunikatów PAD (*PPPoE Active Discovery*) składających się na tzw. fazę wykrywania (*discovery phase*): PADI (inicjacja — *PAD Initiation*), PADO (oferowanie — *PAD Offer*), PADR (żądanie — *PAD Request*) i PADS (potwierdzenie — *PAD Session Confirmation*). Ustanowiona sesja PPP trwa do czasu wysłania przez którąś ze stron komunikatu PADT (zakończenie — *PAD Termination*) lub zerwania połączenia. Protokół PPPoE posługuje się komunikatami o formacie przedstawionym na rysunku 6.30; komunikaty te przenoszone są w ramkach ethernetowych jako ładunek użyteczny.



Rysunek 6.30. *Format komunikatu PPPoE przenoszonego jako ładunek użyteczny w ramce ethernetowej. W polu Rozmiar/Typ ramki (patrz rysunek 3.3 w rozdziale 3.) znajduje się wartość 0x8863 w fazie wykrywania oraz 0x8864 w czasie trwającej sesji PPP. Komunikaty PAD zawierają opcje w formacie TLV, przenoszące informację konfiguracyjną podobnie do DHCP. ID sesji ustalany jest przez serwer i komunikowany klientowi w komunikacie PADS*

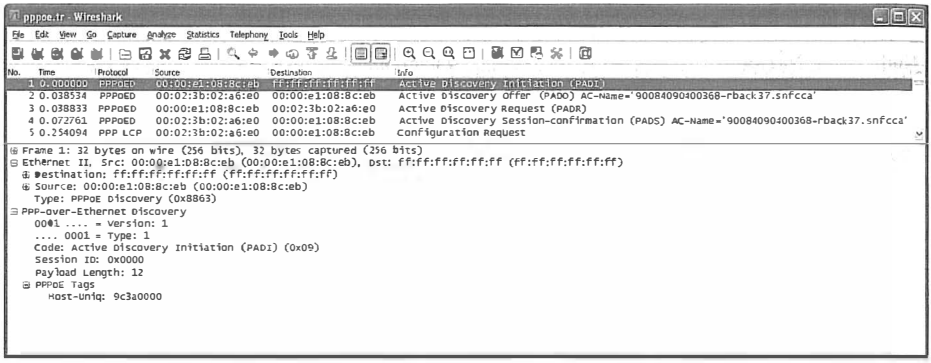
W obecnej wersji PPPoE w każdym z pól *Wersja* i *Typ* znajduje się wartość 0x1 (binarnie 0001). Wartość w polu *Kod* identyfikuje typ komunikatu, zgodnie z legendą u dołu rysunku, z prawej strony. Pole *ID sesji* ma początkowo (w komunikatach PADI i PADO) wartość 0; począwszy od komunikatu PADR, zawiera unikatowy 16-bitowy identyfikator utrzymywany we wszystkich przyszłych komunikatach, aż do zakończenia sesji. Każdy z komunikatów PAD może zawierać jeden lub kilka *Znaczników*, każdy w formacie TLV (*Type-Length-Value* — typ-długość-dane): w formacie tym pierwsze słowo 16-bitowe wskazuje *typ* (TAG_TYPE) znacznika, na kolejnych 16 bitach zapisana jest długość (TAG_LENGTH) danych, potem następują właściwe dane. Zdefiniowane typy znaczników zebrano w tabeli 6.2.

Tabela 6.2. *Zdefiniowane kody typów znaczników w komunikatach PAD protokołu PPPoE*

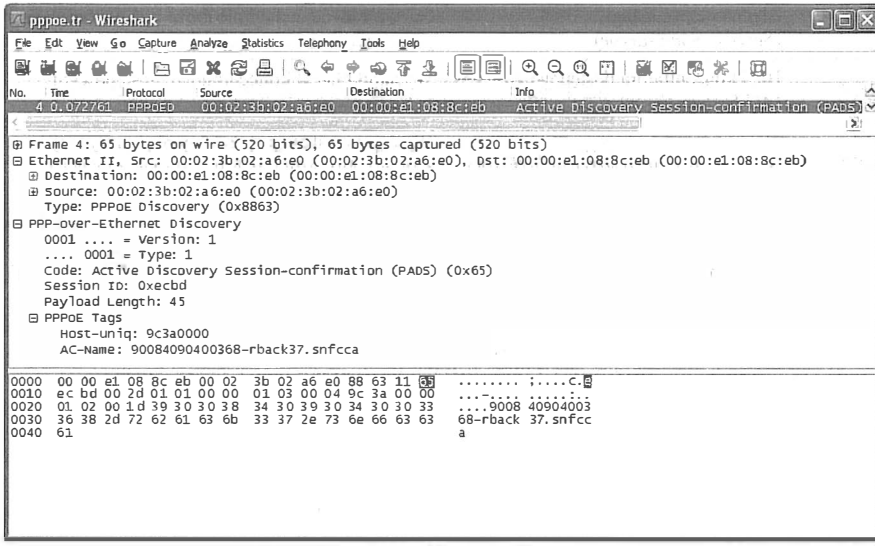
Kod	Nazwa	Znaczenie
0x0000	Koniec listy (*End-of-List*)	Identyfikuje znacznik jako ostatni; nie posiada danych, pole TAG_LENGTH musi zawierać wartość 0.
0x0101	Nazwa usługi (*Service-Name*)	Zawiera nazwę usługi (w kodowaniu UTF-8) na użytek dostawcy Internetu.
0x0102	Nazwa koncentratora (*AC-Name*)	Zawiera 8-znakowy łańcuch w kodowaniu UTF-8, identyfikujący koncentrator dostępowy.
0x0103	Wyróżnik hosta (*Host-Uniq*)	Binarne dane, używane przez klienta do kojarzenia komunikatów, nieinterpretowany przez koncentrator dostępowy.
0x0104	„Ciasteczko" koncentratora (*AC-Cookie*)	Binarne dane wykorzystywane przez koncentrator dostępowy do ochrony przed atakami DoS; powtarzane przez klienta.
0x0105	Informacja specyficzna dla producenta (*Vendor-Specific*)	Znacznik niezalecany — szczegóły w dokumencie [RFC2516].
0x0110	Identyfikator przekaźnika w sesji (*Relay-Session-ID*)	Może być dodany przez agenta przekazującego komunikaty PAD.
0x0201	Błędna nazwa usługi (*Service-Name-Error*)	Nazwa żądanej usługi nie może być honorowana przez koncentrator dostępowy.
0x0202	Błąd koncentratora dostępowego (*AC-System-Error*)	Wystąpił błąd w koncentratorze dostępowym w trakcie wykonywania żądanej akcji.
0x0203	Niesprecyzowany błąd (*Generic-Error*).	Wystąpił nieskategoryzowany błąd krytyczny, którego opis dostępny jest w postaci łańcucha znaków w kodowaniu UTF-8.

Aby zobaczyć PPPoE w akcji, możemy monitorować wymianę komunikatów między domowym komputerem a koncentratorem dostępowym z rysunku 6.28. Pierwszy pakiet PPP w fazie wykrywania (*discovery*) widoczny jest na rysunku 6.31. Widzimy spodziewaną wymianę komunikatów PADI, PADO, PADR i PADS. Każdy z nich zawiera znacznik wyróżnika hosta (*Host Uniq*) o wartości 9c3a0000; komunikaty docierające z koncentratora zawierają znacznik *AC-Name* z nazwą 90084090400368-rback37.snfcca. Szczegóły komunikatu PADS widoczne są na rysunku 6.32.

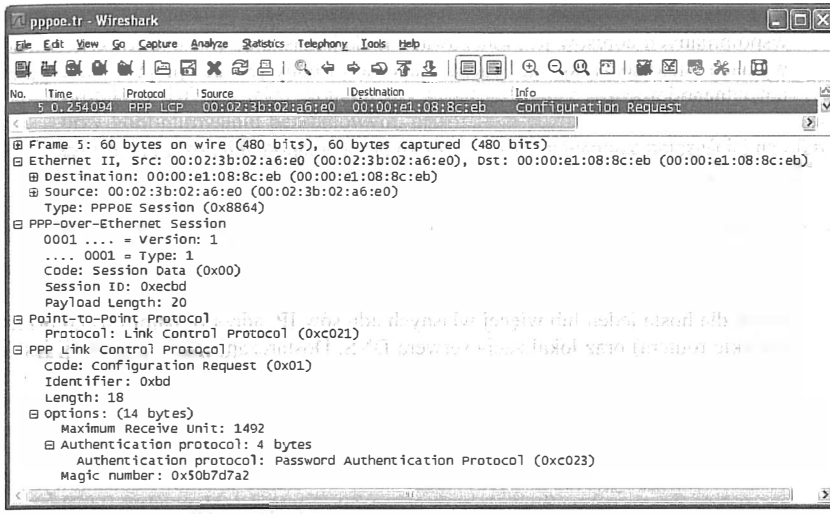
Komunikat PADS, zatwierdzający nawiązanie sesji, zawiera jej identyfikator 0xecbd, widoczna jest także nazwa koncentratora dostępowego (znacznik *AC-Name*), z którego komunikat został wysłany. Sesja PPP została rozpoczęta — na rysunku 6.33 widoczny jest pierwszy jej pakiet. Sesja rozpoczyna się wysłaniem przez klienta komunikatu *Configuration Request* (patrz rozdział 3.) oznaczającego żądanie uwierzytelnienia wobec koncentratora za pomocą protokołu PAP (*Password Authentication Protocol*) — czyli w sposób niezbyt bezpieczny. Gdy dialog uwierzytelniający zostaje pomyślnie zakończony i wymienione zostają różnorodne parametry łącza (m.in. MRU), wywoływany jest protokół IPCP w celu przydzielenia klientowi i skonfigurowania adresu IP — zwróćmy uwagę, że konieczne jest jeszcze dostarczenie kilku informacji pomocniczych, m.in. adresu IP serwera DNS; szczegóły tej operacji zostają ustalone przez dostawcę Internetu, być może użytkownik będzie musiał wykonać ją samodzielnie.



Rysunek 6.31. *Wymiana informacji w ramach PPPoE rozpoczyna się od wysłania przez klienta komunikatu PADI na adres rozgłoszeniowy; kolejne komunikaty używają już adresów unicast. W widocznej wymianie użyto znaczników Host-Uniq i AC-Name. Pierwszym komunikatem w ramach nawiązanej sesji jest komunikat piąty w kolejności, rozpoczynający proces konfigurowania łącza PPP, prowadzący do przydzielenia systemowi adresu IPv4 za pomocą protokołu IPCP (patrz rozdział 3.)*



Rysunek 6.32. *Komunikat PADS potwierdzający ustanowienie skojarzenia między klientem a koncentratorem dostępowym. W komunikacie tym zawarty jest (ustalony przez serwer) identyfikator sesji Oxecbd, wykorzystywany w następnych pakietach sesji PPP*



Rysunek 6.33. *Pierwszym komunikatem nawiązanej sesji PPPoE jest żądanie konfiguracyjne (Configuration Request); w ramce ethemetowej pole Typ/Rozmiar zmienia wartość z 0x8863 na 0x8864, identyfikator sesji ustawiony zostaje na wartość 0xecbd. Klient zgłasza żądanie uwierzytelnienia za pomocą (relatywnie mało bezpiecznego) protokołu PAP*

6.6. Ataki ukierunkowane na konfigurowanie systemu

Konfiguracja systemów to bardzo wdzięczne pole do popisu dla hakerskiej pomysłowości. Nieautoryzowany niesforny klient może np. zarzucić serwer DHCP masywnymi żądaniami przydziału wszelkich możliwych adresów IP z oferowanego przez ten serwer zakresu — i już mamy znakomity przepis na spowodowanie ataku przeciążeniowego (DoS). Zagrożenia kryjące się w protokole DHCP są o tyle niebezpieczne, że projektując go w wersji dla IPv4, zakładano całkowite zaufanie w ramach sieci, a nowym wersjom DHCP podchodzącym bardziej ostrożnie do kwestii bezpieczeństwa daleko jeszcze do rozpowszechnienia (a mechanizmy zabezpieczające cieszą się jeszcze mniejszym powodzeniem z powodów, o których pisaliśmy już w punkcie 6.2.7). Można więc stwierdzić, że sam DHCP jako taki prezentuje ograniczony raczej poziom defensywności; skuteczności obrony przed nadużyciami jego funkcjonalności należy raczej upatrywać w zabezpieczeniach na poziomie warstwy łącza danych — np. WPA2 — chroniących przed nieuprawnionym dostępem do konkretnej sieci.

W IETF trwają prace zmierzające do zapewnienia bezpieczeństwa protokołowi IPv6 *Neighbor Discovery* i generalnie całemu mechanizmowi SLAAC. Problemy bezpieczeństwa tego protokołu opisywane są w dokumencie [RFC3756] pochodzącym z roku 2004, niespełna rok później ukazał się dokument [RFC3971] definiujący protokół *Secure Neighbor Discovery* (SEND). Protokół ten łączy użycie IPsec w odniesieniu do pakietów *Neighbor Discovery* z generowaniem adresów IP na drodze kryptograficznej (CGA —

Cryptographically Generated Addresses), opisywanym w [RFC3972]. Generatorem wspomnianych adresów jest kluczowana funkcja haszująca, a sam proces generowania wymaga znajomości tajnego klucza i jako taki trudny jest do spreparowania przez potencjalnego intruza.

6.7. Podsumowanie

Aby host lub router mogły funkcjonować w kontekście Internetu — i generalnie w kontekście każdej sieci wykorzystującej protokół IP — potrzebują przynajmniej pewnego minimum informacji konfiguracyjnych: dla routera jest to odpowiednia informacja adresowa, dla hosta jeden lub więcej własnych adresów IP, adres IP najbliższego przeskoku (zwykle routera) oraz lokalizacja serwera DNS. Dostarczanie tych informacji jest zadaniem protokołu DHCP, dostępnego w wersjach IPv4 i IPv6, znacząco różniących się od siebie. Odpowiednio skonfigurowany serwer DHCP zajmuje się przydzielaniem klientom (na ich żądanie) adresów IP na zasadzie ich dzierżawy przez ustalony okres; klient zamierzający użytkować przydzielony adres w dalszym ciągu może wystąpić do serwera o przedłużenie dzierżawy. Serwer DHCP może także dostarczać klientowi dodatkowych informacji, niezwiązanych bezpośrednio z przydziałem adresów, takich jak maska podsieci, lokalizacja domyślnych routerów, serwera DNS i agenta domowego oraz domyślna nazwa domeny i informacja specyficzna dla producenta. Funkcjonowanie protokołu DHCP może obejmować również przekaźniki (agenty przekazywania) niezbędne w sytuacji, gdy serwer DHCP znajduje się w innej sieci (podsieci) niż klienty korzystające z jego usług (ponieważ klientom nie przyporządkowano jeszcze adresów IP, nie można w tej sytuacji użyć routerów do ich skomunikowania z serwerem DHCP). Serwer DHCP może także scedować część akcji przydziału adresów na router, delegując do niego określony zakres przestrzeni adresowej.

W wersji IPv6 typowy host wykorzystuje wiele adresów. Adresy lokalne dla łącza mogą być przez niego generowane automatycznie, przy użyciu predefiniowanego prefiksu oraz specyficznych informacji lokalnych, takich jak adres MAC czy liczby pseudolosowe. Analogicznie odbywa się generowanie adresów globalnych, z tą jednak istotną różnicą, że prefiks adresu dostarczany jest w ramach bądź to komunikatów ogłoszeniowych ICMP (*advertisements*) routera, bądź też przez serwer DHCP.

Serwery DHCP mogą także funkcjonować w trybie „bezstanowym": nie realizują wówczas przydziału adresów i nie utrzymują żadnych informacji związanych ze stanem tego przydziału dla poszczególnych klientów (stąd nazwa trybu), dostarczają natomiast klientom rozmaite informacje uzupełniające związane z konfiguracją.

PPPoE to połączenie protokołu PPP z Ethernetem — komunikaty PPP transmitowane są w ramkach ethernetowych. Technika ta wykorzystywana jest przez komputer użytkownika do nawiązywania połączenia z dostawcą Internetu, najczęściej w kontekście usługi DSL. Modem DSL z portem ethernetowym pełni wówczas rolę mostka lub przełącznika. Nawiązanie połączenia PPPoE między komputerem klienta a koncentratorem dostępowym jest rezultatem wymiany czterech komunikatów PAD, po czym w ramach sesji PPP ramki ethernetowe przenoszą dane różnych protokołów, m.in. IP; sesja kończy się na żądanie jednego z uczestników lub automatycznie wskutek przerwania połączenia w warstwie łącza danych. Użytkownik zaopatrywany jest w niezbędną informację konfiguracyjną

za pośrednictwem odpowiednich mechanizmów protokołu PPP, np. protokołu IPCP opisywanego w rozdziale 3.

Protokół DHCP, a także komunikaty ICMPv6 *Router Advertisement* wykorzystywane przy konfigurowaniu bezstanowym, oryginalnie pozbawione są mechanizmów bezpieczeństwa i jako takie podatne są na rozmaite ataki, polegające najczęściej na oferowaniu fałszywych adresów przez „podstawione" serwery DHCP czy też paraliżowaniu legalnych serwerów DHCP przez wyczerpujące zarzucanie ich żądaniami coraz to nowych adresów. Wielu tym atakom można obecnie przeciwdziałać za pomocą nowych technik zabezpieczających, m.in. uwierzytelniania DHCP i bezpiecznego protokołu SEND; niestety, w praktyce techniki te wykorzystywane są jeszcze nader rzadko.

6.8. Bibliografia

[802.21-2008], *IEEE Standard for Local and Metropolitan Area Networks — Part 21: Media Independent Handover Services*, listopad 2008.

[F07] R. Faas, *Hands On: Configuring Apple's NetBoot Service, Part 1*, „Computerworld", wrzesień 2007.

[GC89] C. Gray, D. Cheriton, *Leases: An Efficient Fault-Tolerant Mechanism for Distributed File Cache Consistency*, Proc. ACM Symposium on Operating System Principles (SOSP), 1989.

[IARP] *http://www.iana.org/assignments/arp-parameters*

[IBDP] *http://www.iana.org/assignments/bootp-dhcp-parameters*

[ID4LQ] K. Kinnear, B. Volz, M. Stapp, D. Rao, B. Joshi, N. Russell, P. Kurapati, *Bulk DHCPv4 Lease Query*, Internet draft-ietf-dhc-dhcpv4-bulk-leasequery, w przygotowaniu, kwiecień 2011.

[ID4RI] B. Joshi, R. Rao, M. Stapp, *The DHCPv4 Relay Agent Identifier Suboption*, Internet draft-ietf-dhc-relay-id-suboption, w przygotowaniu, czerwiec 2011.

[ID6PARAM] *http://www.iana.org/assignments/dhcpv6-parameters*

[IDDN] G. Daley, E. Nordmark, N. Moore, *Tentative Options for Link-Layer Addresses in IPv6 Neighbor Discovery*, Internet draft-ietf-dna-tentative (expired), w przygotowaniu, październik 2009.

[IDL2RA] B. Joshi, P. Kurapati, *Layer 2 Relay Agent Information*, Internet draft-ietf-dhc-l2ra, w przygotowaniu, kwiecień 2011.

[IEPARAM] *http://www.iana.org/assignments/enterprise-numbers*

[MKB928233] Microsoft Knowledge Base Article 928233, *http://support.microsoft.com*

[MS-DHCPN] Microsoft Corporation, *[MS-DHCPN]: Dynamic Host Configuration Protocol (DHCP) Extensions for Network Access Protection (NAP)*, *http://msdn.microsoft.com/en-us/library/cc227316.aspx*, październik 2008.

[RFC0826] D. Plummer, *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware*, Internet RFC 0826/STD 0037, listopad 1982.

[RFC0951] W.J. Croft, J. Gilmore, *Bootstrap Protocol*, Internet RFC 0951, wrzesień 1985.

[RFC1542] W. Wimer, *Clarifications and Extensions for the Bootstrap Protocol*, Internet RFC 1542, październik 1993.

[RFC2131] R. Droms, *Dynamic Host Configuration Protocol*, Internet RFC 2131, marzec 1997.

[RFC2132] S. Alexander, R. Droms, *DHCP Options and BOOTP Vendor Extensions*, Internet RFC 2132, marzec 1997.

[RFC2241] D. Provan, *DHCP Options for Novell Directory Services*, Internet RFC 2241, listopad 1997.

[RFC2242] R. Droms, K. Fong, *NetWare/IP Domain Name and Information*, Internet RFC 2242, listopad 1997.

[RFC2516] L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, R. Wheeler, *A Method for Transmitting PPP over Ethernet (PPPoE)*, Internet RFC 2516 (informational), luty 1999.

[RFC2563] R. Troll, *DHCP Option to Disable Stateless Auto-Configuration in IPv4 Clients*, Internet RFC 2563, maj 1999.

[RFC2937] C. Smith, *The Name Service Search Option for DHCP*, Internet RFC 2937, wrzesień 2000.

[RFC3004] G. Stump, R. Droms, Y. Gu, R. Vyaghrapuri, A. Demirtjis, B. Beser, J. Privat, *The User Class Option for DHCP*, Internet RFC 3004, listopad 2000.

[RFC3011] G. Waters, *The IPv4 Subnet Selection Option for DHCP*, Internet RFC 3011, listopad 2000.

[RFC3046] M. Patrick, *DHCP Relay Agent Information Option*, Internet RFC 3046, styczeń 2001.

[RFC3118] R. Droms, W. Arbaugh (red.), *Authentication of DHCP Messages*, Internet RFC 3118, czerwiec 2001.

[RFC3203] Y. T'Joens, C. Hublet, P. De Schrijver, *DHCP Reconfigure Extension*, Internet RFC 3203, grudzień 2001.

[RFC3315] R. Droms (red.), J. Bound, B. Volz, T. Lemon, C. Perkins, M. Carney, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, Internet RFC 3315, lipiec 2003.

[RFC3396] T. Lemon, S. Cheshire, Enco*ding Long Options in the Dynamic Host Configuration Protocol (DHCPv4)*, Internet RFC 3396, listopad 2002.

[RFC3442] T. Lemon, S. Cheshire, B. Volz, *The Classless Static Route Option for Dynamic Host Configuration Protocol (DHCP) Version 4*, Internet RFC 3442, grudzień 2002.

[RFC3633] O. Troan, R. Droms, *IPv6 Prefix Options for Dynamic Host Configuration Protocol (DHCP) Version 6*, Internet RFC 3633, grudzień 2003.

[RFC3646] R. Droms (red.), *DNS Configuration Options for Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*, Internet RFC 3646, grudzień 2003.

[RFC3693] J. Cuellar, J. Morris, D. Mulligan, J. Peterson, J. Polk, *Geopriv Requirements*, Internet RFC 3693 (informational), luty 2004.

[RFC3736] R. Droms, *Stateless Dynamic Host Configuration Protocol (DHCP) Service for IPv6*, Internet RFC 3736, kwiecień 2004.

[RFC3756] P. Nikander (red.), J. Kempf, E. Nordmark, *IPv6 Neighbor Discovery (ND) Trust Models and Threats*, Internet RFC 3756 (informational), maj 2004.

[RFC3925] J. Littlefield, *Vendor-Identifying Vendor Options for Dynamic Host Configuration Protocol Version 4 (DHCPv4)*, Internet RFC 3925, październik 2004.

[RFC3927] S. Cheshire, B. Aboba, E. Guttman, *Dynamic Configuration of IPv6 Link-Local Addresses*, Internet RFC 3927, maj 2005.

[RFC3971] J. Arkko (red.), J. Kempf, B. Zill, P. Nikander, *SEcure Neighbor Dicovery (SEND)*, Internet RFC 3971, marzec 2005.

[RFC3972] T. Aura, *Cryptographically Generated Addresses (CGA)*, Internet RFC 3972, marzec 2005.

[RFC4030] M. Stapp, T. Lemon, *The Authentication Suboption for the Dynamic Host Configuration Protocol (DHCP) Relay Agent Option*, Internet RFC 4030, marzec 2005.

[RFC4039] S. Park, P. Kim, B. Volz, *Rapid Commit Option for the Dynamic Host Configuration Protocol Version 4 (DHCPv4)*, Internet RFC 4039, marzec 2005.

[RFC4174] C. Monia, J. Tseng, K. Gibbons, *The IPv4 Dynamic Host Configuration Protocol (DHCP) Option for the Internet Storage Name Service*, Internet RFC 4174, wrzesień 2005.

[RFC4280] K. Chowdhury, P. Yegani, L. Madour, *Dynamic Host Configuration Protocol (DHCP) Options for Broadcast and Multicast Control Servers*, Internet RFC 4280, listopad 2005.

[RFC4291] R. Hinden, S. Deering, *IP Version 6 Addressing Architecture*, Internet RFC 4291, luty 2006.

[RFC4361] T. Lemon, B. Sommerfield, *Node-Specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)*, Internet RFC 4361, luty 2006.

[RFC4388] R. Woundy, K. Kinnear, *Dynamic Host Configuration Protocol (DHCP) Leasequery*, Internet RFC 4388, luty 2006.

[RFC4429] N. Moore, *Optimistic Duplicate Address Detection (DAD) for IPv6*, Internet RFC 4429, kwiecień 2006.

[RFC4436] B. Aboba, J. Carlson, S. Cheshire, *Detecting Network Attachment in IPv4 (DNAv4)*, Internet RFC 4436, marzec 2006.

[RFC4649] B. Volz, *Dynamic Host Configuration Protocol (DHCPv6) Relay Agent Remote-ID Option*, Internet RFC 4649, sierpień 2006.

[RFC4702] M. Stapp, B. Volz, Y. Rekhter, *The Dynamic Host Configuration Protocol (DHCP) Client Fully Qualified Domain Name (FQDN) Option*, Internet RFC 4702, październik 2006.

[RFC4704] B. Volz, *The Dynamic Host Configuration Protocol for IPv6 (IPv6) Client Fully Qualified Domain Name (FQDN) Option*, Internet RFC 4704, październik 2006.

[RFC4776] H. Schulzrinne, *Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Option for Civic Addresses Configuration Information*, Internet RFC 4776, listopad 2006.

[RFC4833] E. Lear, P. Eggert, *Timezone Options for DHCP*, Internet RFC 4833, kwiecień 2007.

[RFC4862] S. Thomson, T. Narten, T. Jinmei, *IPv6 Stateless Address Autoconfiguration*, Internet RFC 4862, wrzesień 2007.

[RFC4941] T. Narten, R. Draves, S. Krishnan, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, Internet RFC 4941, wrzesień 2007.

[RFC5007] J. Brzozowski, K. Kinnear, B. Volz, S. Zeng, *DHCPv6 Leasequery*, Internet RFC 5007, wrzesień 2007.

[RFC5010] K. Kinnear, M. Normoyle, M. Stapp, *The Dynamic Host Configuration Protocol Version 4 (DHCPv4) Relay Agent Flags Suboption*, Internet RFC 5010, wrzesień 2007.

[RFC5107] R. Johnson, J. Kumarasamy, K. Kinnear, M. Stapp, *DHCP Server Identifier Override Suboption*, Internet RFC 5107, luty 2008.

[RFC5175] B. Haberman (red.), R. Hinden, *IPv6 Router Advertisement Flags Option*, Internet RFC 5175, marzec 2008.

[RFC5192] L. Morand, A. Yegin, S. Kumar, S. Madanapalli, *DHCP Options for Protocol for Carrying Authentication for Network Access (PANA) Authentication Agents*, Internet RFC 5192, maj 2008.

[RFC5222] T. Hardie, A. Newton, H. Schulzrinne, H. Tschofenig, *LoST: A Location-to-Service Translation Protocol*, Internet RFC 5222, sierpień 2008.

[RFC5223] H. Schulzrinne, J. Polk, H. Tschofenig, *Discovering Location-to-Service Translation (LoST) Servers Using the Dynamic Host Configuration Protocol (DHCP)*, Internet RFC 5223, sierpień 2008.

[RFC5460] M. Stapp, *DHCPv6 Bulk Leasequery*, Internet RFC 5460, luty 2009.

[RFC5569] R. Despres, *IPv6 Rapid Deployment on IPv4 Infrastructures (6rd)*, Internet RFC 5569 (informational), styczeń 2010.

[RFC5677] T. Melia (red.), G. Bajko, S. Das, N. Golmie, JC. Zuniga, *IEEE 802.21 Mobility Services Framework Design (MSFD)*, Internet RFC 5677, grudzień 2009.

[RFC5678] G. Bajko, S. Das, *Dynamic Host Configuration Protocol (DHCPv4 and DHCPv6) Options for IEEE 802.21 Mobility Services (MoS) Discovery*, Internet RFC 5678, grudzień 2009.

[RFC5735] M. Cotton, L. Vegoda, *Special-Use IPv4 Addresses*, Internet RFC 5735/BCP 0153, styczeń 2010.

[RFC5969] W. Townsley, O. Troan, *IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) — Protocol Specification*, Internet RFC 5969, sierpień 2010.

[RFC5985] M. Barnes (red.), *HTTP-Enabled Location Delivery (HELD)*, Internet RFC 5985, wrzesień 2010. Information Server (LIS)", Internet RFC 5986, wrzesień 2010.

[RFC5986] M. Thomson, J. Winterbottom, *Discovering the Local Location Information Server (LIS)*, Internet RFC 5986, wrzesień 2010.

[RFC6059] S. Krishnan, G. Daley, *Simple Procedures for Detecting Network Attachment in IPv6*, Internet RFC 6059, listopad 2010.

[RFC6106] J. Jeong, S. Park, L. Beloeil, S. Madanapalli, *IPv6 Router Advertisement Options for DNS Configuration*, Internet RFC 6106, listopad 2010.

[RFC6148] P. Kurapati, R. Desetti, B. Joshi, *DHCPv4 Lease Query by Relay Agent Remote ID*, Internet RFC 6148, luty 2011.

[RFC6153] S. Das, G. Bajko, *DHCPv4 and DHCPv6 Options for Access Network Discovery and Selection Function (ANDSF) Discovery*, Internet RFC 6153, luty 2011.

[RFC6221] D. Miles (red.), S. Ooghe, W. Dec, S. Krishnan, A. Kavanagh, *Lightweight DHCPv6 Relay Agent*, Internet RFC 6221, maj 2011.

[RFC6225] J. Polk, M. Linsner, M. Thomson, B. Aboba (red.), *Dynamic Host Configuration Protocol Options for Coordinate-Based Location Configuration Information*, Internet RFC 6225, marzec 2011.

[RFC6276] R. Droms, P. Thubert, F. Dupont, W. Haddad, C. Bernardos, *DHCPv6 Prefix Delegation for Network Mobility (NEMO)*, Internet RFC 6276, lipiec 2011.

Rozdział 7.
Firewalle i translacja adresów sieciowych (NAT)

7.1. Wprowadzenie

Wczesne lata Internetu to jego akademicka kolebka: większość projektantów i deweloperów wywodziło się z uniwersytetów, „korzystanie" z Internetu wiązało się w znacznej mierze z jego ulepszaniem w ramach skoordynowanych wysiłków. Internet nie był jakoś specjalnie uodporniony na ataki, bo też i mało kto widział sens organizowania takich ataków. Sytuacja ta zmieniła się jednak radykalnie na przełomie lat 80. i 90. ubiegłego wieku, wraz z masowo rosnącym zainteresowaniem Internetem ze strony przeciętnych użytkowników i równie masowo rosnącymi próbami skompromitowania jego bezpieczeństwa. Próbami bardzo często udanymi, bo też motywacje i determinacja potencjalnych intruzów zyskały sobie mimowolnego sojusznika w postaci błędów i luk w implementacji protokołów. Ogromne zróżnicowanie wersji programowania, zwłaszcza w dużych firmach, nie sprzyjało wystarczająco szybkiemu instalowaniu poprawek; dla starszych systemów często poprawek takich po prostu nie konstruowano. Podczas gdy programiści łatali obnażone w ich produktach luki, niestrudzeni hakerzy wciąż odkrywali nowe. Stało się oczywiste, że ruch internetowy, na interakcje z którym wystawione zostały liczne hosty użytkowników, musi być w jakiś sposób kontrolowany — trafiające do hosta pakiety muszą podlegać *filtrowaniu* zgodnie z inteligentnie określonymi kryteriami. Tak narodziła się koncepcja *firewalla*, zwanego także *zaporą sieciową* (lub, w tłumaczeniu dosłownym, „ścianą przeciwogniową", przez analogię do konstrukcji budowlanych utrudniających rozprzestrzenianie się pożaru). Firewall to router poddający restrykcyjnej kontroli pakiety, które docierają do niego w celu ich forwardowania.

Obok kwestii bezpieczeństwa pojawił się nowy problem, może nie równie palący, lecz w dłuższej perspektywie — gdyby nie został rozwiązany — wieszczący po prostu katastrofę. Otóż zestaw możliwych adresów IP, liczonych w miliardach, zdawał się sukcesywnie zmierzać do wyczerpania. Poza oczywistym w tej sytuacji rozwiązaniem dalekosiężnym, czyli opracowaniem nowej wersji protokołu IP (znanej dziś pod akronimem IPv6), pożądane było znalezienie rozwiązań doraźnych, zastępczych, mniej uniwersalnych, lecz możliwych do skonstruowania i wdrożenia w krótszej perspektywie. Wśród wielu konkurencyjnych rozwiązań najbardziej udana okazała się technika *translacji adresów sieciowych*, znana powszechnie pod postacią akronimu NAT (od *Network Address*

Translation). Jej generalnym przesłaniem jest podział Internetu na *domeny adresowe* (*address realms*) i zapewnienie unikatowości adresów IP w obrębie każdej z domen jako alternatywy do unikatowości adresu IP każdego z hostów w skali całego Internetu. Te same adresy mogą więc być używane wielokrotnie, w różnych domenach, co opisaną groźbę wyczerpania adresów w znaczącym stopniu oddala.

Jak niebawem pokażemy, funkcjonalności NAT i firewalli mają wiele wspólnego, toteż często łączy się je w ramach pojedynczych urządzeń, popularnych zwłaszcza w sieciach domowych i małych przedsiębiorstwach. Zacznijmy jednak od szczegółowego opisu obu wymienionych technik.

7.2. Firewalle

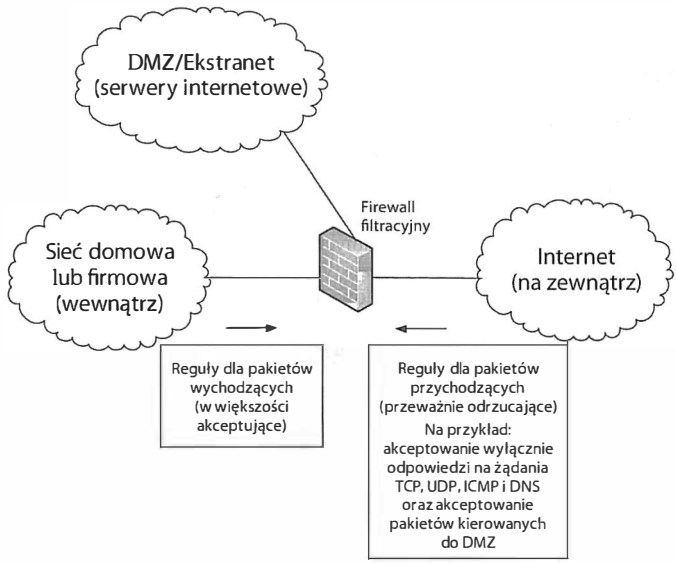
Wobec dużych trudności w zapewnieniu aktualności używanego oprogramowania, a szczególnie dostatecznie szybkiego wykrywania i łatania luk, podstawową strategią walki z atakami internetowymi jest restrykcyjna kontrola informacji docierającej z Internetu do hosta i wysyłanej przez ten host. Jest to zadanie firewalli, które są dziś w powszechnym użyciu w kilku odmianach.

Dwa podstawowe typy firewalli to *firewall filtracyjny* i *firewall proxy*. Zasadniczą cechą, która je różni, jest ich umiejscowienie w warstwowym modelu odniesienia. Firewall filtracyjny jest w istocie routerem internetowym, odrzucającym pakiety spełniające określone kryteria (lub niespełniające określonych kryteriów). Firewall proxy pełni natomiast rolę pośrednika między klientem a Internetem, z perspektywy klienta jest to więc serwer multihomed, czyli jeden z punktów końcowych połączenia TCP lub UDP. Firewall filtracyjny jest zatem mechanizmem warstwy sieciowej, podczas gdy firewall proxy plasuje się na poziomie warstwy transportowej i nie ingeruje w trasowanie pakietów na poziomie protokołu IP.

7.2.1. Firewalle filtrujące pakiety

Istotą funkcjonowania firewalla filtracyjnego jest „odfiltrowywanie", czyli odrzucanie niektórych pakietów, na podstawie pewnych kryteriów składających się na definicje *filtrów*. Proste filtry opierają się na klasyfikowaniu zawartości określonych pól nagłówków warstw sieciowej lub transportowej, najczęściej adresów IP, opcji, typów komunikatów ICMP oraz numerów portów reprezentujących określone usługi TCP i UDP. Jak niebawem pokażemy, proste firewalle filtracyjne działają w sposób bezstanowy, czyli traktują każdy pakiet indywidualnie, bez związku z pakietami poprzednio przetworzonymi. Bardziej skomplikowane firewalle traktują przetwarzany aktualnie pakiet w kontekście historii dotychczasowych pakietów i również zaliczają go na poczet historii dla przyszłych pakietów. Dzięki temu możliwe staje się wykrywanie określonych prawidłowości — tych dotyczących określonego połączenia w warstwie transportowej i tych obserwowanych w poszczególnych fragmentach datagramu IP (patrz rozdział 10.). Fragmentacja znacząco komplikuje konstrukcję i działanie firewalli, a w przypadku firewalli bezkontekstowych często stanowi skuteczny mechanizm ich oszukiwania.

Działanie typowego firewalla filtracyjnego przedstawiono na rysunku 7.1. Wspomniany firewall jest routerem internetowym z trzema interfejsami, prowadzącymi (odpowiednio) do wnętrza (*inside*), na zewnątrz (*outside*) i do tzw. strefy zdemilitaryzowanej (DMZ — *DeMilitarized Zone*). DMZ to podsieć zawierająca serwery udostępniane klientom za pośrednictwem Internetu bądź serwery składające się na firmowy ekstranet (bądź jedno i drugie). Administratorzy sieci instalują w firewallu filtry w postaci *list kontroli dostępu* (ACL — *Access Control Lists*) szczegółowo określających zasady akceptowania i odrzucania pakietów. Zasady te zwykle zmierzają do eliminowania tych przychodzących z zewnątrz pakietów, które mogłyby być dla sieci szkodliwe, natomiast dość liberalnie podchodzą do pakietów wychodzących na zewnątrz.

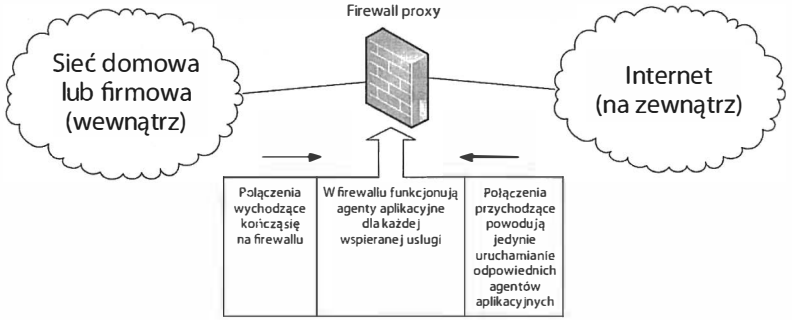


Rysunek 7.1. *Konfiguracja z typowym firewallem filtrującym pakiety. Firewall pełni rolę routera na styku sieci ze światem zewnętrznym, często w sąsiedztwie „strefy zdemilitaryzowanej" (DMZ) lub firmowego ekstranetu. Zadaniem firewalla jest powstrzymywanie niepożądanego ruchu przychodzącego z zewnątrz, natomiast ruch wychodzący traktowany jest z reguły bardziej liberalnie. W strefie DMZ tylko wybrane usługi dostępne są z poziomu Internetu*

7.2.2. Firewalle proxy

Firewall proxy nie jest routerem w ścisłym tego słowa znaczeniu, lecz raczej hostem implementującym jedną lub kilka *bram aplikacyjnych* (ALG — *Application-Layer Gateways*), przenoszących ruch określonego typu między dwoma połączeniami lub skojarzeniami na poziomie warstwy aplikacji. Typowy firewall proxy, w przeciwieństwie do routera, nie zajmuje się forwardowaniem pakietów IP, choć funkcja taka nie jest wykluczona w bardziej zaawansowanych urządzeniach.

Zasadę działania firewalla proxy przedstawiono na rysunku 7.2. Klient, rezydujący po „wewnętrznej" stronie firewalla, skonfigurowany jest na łączenia się z proxy, a nie z rzeczywistym serwerem świadczącym odnośną usługę (aplikacje przystosowane do współpracy



Rysunek 7.2. *Firewall proxy funkcjonuje podobnie do hosta multihomed, stanowiąc punkt końcowy połączeń TCP i skojarzeń UDP na poziomie warstwy aplikacji. Jest więc raczej bramą aplikacyjną niż konwencjonalnym routerem IP. Każda usługa świadczona lub wykorzystywana przez aplikacje musi mieć zapewnioną obsługę w ramach firewalla proxy*

z firewallami proxy zazwyczaj oferują związaną z tym opcję konfiguracyjną). Drugi, „zewnętrzny" interfejs firewalla połączony jest z Internetem. Podobnie jak w przypadku forwardowania pakietów IP przez router, firewall komunikuje się z aplikacjami „wewnątrz" za pomocą lokalnych (prywatnych) adresów IP dla chronionej domeny, natomiast w komunikacji z Internetem wykorzystywane są adresy globalne. Funkcja forwardowania pakietów, jeśli nawet implementowana przez firewall, jest zwykle wyłączona.

Chociaż firewalle proxy mogą być całkiem bezpieczne (i w istocie w przekonaniu wielu użytkowników gwarantują one z zasady większe bezpieczeństwo niż firewalle filtracyjne), to jednak bezpieczeństwo takie okupione jest brakiem elastyczności i generalnie ułomnością samej koncepcji. Jako że firewall w opisywanym stylu musi implementować agenta dla każdej usługi na poziomie warstwy transportowej, pojawienie się nowej usługi wymaga instalowania reprezentującego tę usługę agenta, zaś każdy klient korzystający ze wspomnianej usługi musi być skonfigurowany na odnajdywanie tegoż agenta (np. za pomocą protokołu WPAD — *Web Proxy Auto-Discovery Protocol*, patrz [XIDAD] — lub podobnego mechanizmu przechwytującego ruch określonego typu bez względu na adres docelowy). W efekcie firewall proxy spisuje się doskonale w odniesieniu do usług dobrze rozpoznanych w danym środowisku, lecz może wymagać znaczącej interwencji operatora w odniesieniu do pozostałych.

Przykładami dwóch najczęściej używanych firewalli proxy są *firewalle proxy HTTP* (patrz [RFC2616]), zwane także *proxy WWW* (*Web proxy*) oraz *firewalle SOCKS* (patrz [RFC1928]). Pierwsza z wymienionych grup funkcjonuje wyłącznie w związku z protokołami HTTP i HTTPS, skądinąd bardzo popularnymi i najczęściej używanymi; z perspektywy klienta taki firewall proxy jest serwerem WWW, z punktu widzenia serwera WWW jest natomiast klientem korzystającym z usługi. Firewalle proxy WWW realizują często funkcję *cacheowania stron WWW*: kopia pobranej z Internetu strony WWW zapisywana jest w pamięci podręcznej, powtórne odwołanie się klienta do tej strony skutkować będzie udostępnieniem mu tejże kopii, bez ponownego odwoływania się do serwera WWW. Zmniejsza to wydatnie czas oczekiwania na wyświetlenie strony i tym samym poprawia komfort użytkownika. Niektóre proxy WWW realizują również funkcję *filtrowania treści*, blokując dostęp do określonych serwisów figurujących na różnego rodzaju „czarnych listach", natomiast inne (np. *psiphon* czy *CGIProxy*) realizują funkcję wręcz odwrotną — *tunelowanie* umożliwiające użytkownikom omijanie wspomnianych filtrów.

Protokół SOCKS jest bardziej generyczny i obsługujący więcej usług. W użyciu są aktualnie dwie wersje: 4. i 5. Wersja 4. zapewnia podstawowe wsparcie dla omijania proxy (*proxy traversal*), wersja 5. wprowadza rygorystyczne uwierzytelnianie oraz obsługę UDP i adresowania IPv6. Aplikacja, w której zamierza się ten protokół wykorzystywać, musi zostać utworzona pod tym kątem (co w anglojęzycznej terminologii określa się imiesłowem „socksified") i przystosowana do konkretnej wersji protokołu oraz lokalizacji proxy. Oprócz pośrednictwa w udostępnieniu konkretnej usługi, protokół SOCKS oferuje klientowi także odnajdywanie serwera DNS.

7.3. Translacja adresów sieciowych

Translacja adresów sieciowych, w skrócie NAT (*Network Address Translation*) to zasadniczo technika umożliwiająca używanie takich samych adresów sieciowych w różnych obszarach Internetu, bez niebezpieczeństwa powstawania niejednoznaczności. Oryginalną intencją projektantów NAT było spowolnienie (tymczasowe, aż do rozpowszechnienia IPv6) tempa, w jakim począł się kurczyć zasób dostępnych adresów IPv4, a podstawowe założenie prowadzące do spełnienia tego celu polegało na wykorzystywaniu k globalnych adresów IP na potrzeby (prywatnej) sieci zawierającej N komputerów, przy czym — oczywiście — k jest wyraźnie mniejsze od N, a w skrajnym (i spotykanym najczęściej) przypadku $k = 1$. Wspomniana sieć oddzielona jest od Internetu specjalnym routerem, nazywanym tu (po prostu) *urządzeniem NAT* lub (krótko) *NAT*. Komunikacja tego routera ze „swymi" komputerami odbywa się w oparciu o *prywatne* (lokalne) adresy tychże komputerów: lokalny komputer, który *w roli klienta* chce skorzystać z usługi w Internecie, wysyła w tym celu żądanie, ze swym lokalnym adresem w roli adresu źródłowego, resztą zajmuje się już urządzenie NAT. Oferowanie usługi świadczonej przez lokalny serwer jest już nieco bardziej skomplikowane — wrócimy do tej kwestii w punkcie 7.3.4.

Próba poradzenia sobie z kurczącym się z dnia na dzień zapasem adresów IPv4 to tylko jeden z celów, jaki przyświecał projektantom NAT. Drugim celem, związanym bezpośrednio z rezygnacją z „klasowego" adresowania IPv4 i pojawieniem się techniki CIDR (patrz rozdział 2.), było poprawienie skalowalności trasowania w Internecie. Mechanizm NAT stał się popularny również z tego względu, że w naturalny (nomen omen) sposób oferuje ograniczone funkcje firewalla, wymagającego jedynie minimalnej konfiguracji. Paradoksalnie, wysiłek włożony w opracowywanie i wdrażanie urządzenia NAT znacząco przyhamował prace nad nowym protokołem IPv6 — protokołem, dla którego NAT miało być tylko tymczasowym zastępnikiem i który po szerokim rozpowszechnieniu miał zdegradować NAT do roli rozwiązania historycznego (patrz [RFC4864]).

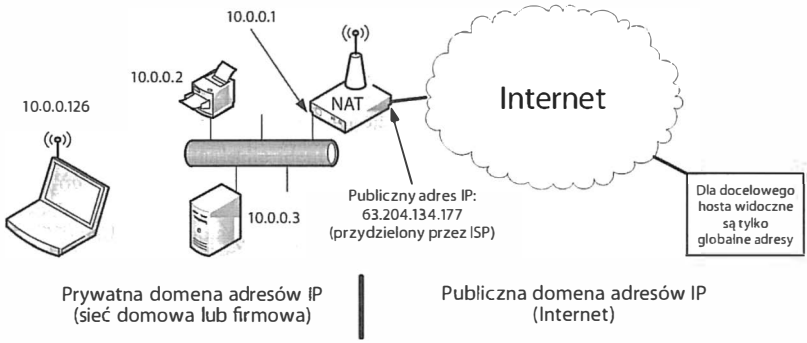
Mimo szerokiej popularności i wielu zalet, NAT nie jest — oczywiście — pozbawione wad. Przede wszystkim sytuacja, gdy któryś z komputerów w lokalnej sieci pełnić ma rolę serwera świadczącego usługę dostępną w Internecie, wymaga specjalnego konfigurowania urządzenia NAT — tak aby wspomniany serwer identyfikowany był przez globalny adres IP, a nie oryginalny prywatny, mający znaczenie tylko wewnątrz jego macierzystej sieci. Ponadto każdy pakiet na drodze od komputera w sieci lokalnej do Internetu i z powrotem musi przechodzić przez to samo urządzenie NAT, nieustannie realizujące translację między oryginalnym adresem prywatnym wspomnianego komputera a adresem globalnym, pod którym komputer ten widoczny jest w Internecie. Wreszcie — translacja adresów sieciowych wymaga niekiedy głębokiej ingerencji w zawartość przenoszoną przez

forwardowane pakiety (o czym dokładniej za chwilę), co wyraźnie narusza złotą zasadę Internetu *smart edge, dumb middle*, czyli obojętność protokołu enkapsulującego względem enkapsulowanego ładunku użytecznego; między innymi, zmodyfikowanie adresu IP wiąże się z koniecznością uaktualnienia sumy kontrolnej w nagłówku IP (dlaczego? — to wyjaśnimy w rozdziałach 10. i 13.).

Technice NAT daleko jest do „przezroczystości" tak charakterystycznej dla wielu protokołów internetowych. NAT nie jest np. obojętna dla protokołów aplikacyjnych, szczególnie tych, które przenoszą informację adresową wewnątrz ładunku użytecznego. Koronnymi przykładami takich protokołów są FTP (*File Transfer Protocol*) opisywany w [RFC0959] i SIP opisywany w [RFC5411]: wymagają one specjalnej funkcji bramy aplikacyjnej, nadpisującej dane aplikacji w sposób umożliwiający niemodyfikowanie ich przez NAT bądź modyfikującej te dane pod kątem konkretnego urządzenia NAT. Kompletna (prawdopodobnie) lista problemów i wymagań związanych z NAT znajduje się w dokumencie [RFC3027]. Niezależnie jednak od wielu problemów, NAT jest obecnie techniką szeroko rozpowszechnioną, implementowaną przez większość routerów (na czele z tanimi routerami dla sieci domowych); dokument [RFC3235] zawiera także zalecenia dla programistów w zakresie tworzenia aplikacji „przyjaznych dla NAT" (*NAT-friendly*). Należy ponadto zaznaczyć, że mimo wielu niedostatków NAT zapewnia wsparcie dla podstawowych protokołów (m.in. poczty elektronicznej i WWW) wykorzystywanych codziennie przez miliony użytkowników.

Istotą działania NAT jest konwertowanie informacji identyfikacyjnej zawartej w pakietach przechodzących przez router, najczęściej dla obu kierunków. W najbardziej podstawowej formie tą „informacją identyfikacyjną" jest adres IP — źródłowy w pakietach wychodzących i docelowy w pakietach przychodzących (jak na rysunku 7.3). W rezultacie adres źródłowy, który początkowo jest prywatnym adresem hosta, zamieniony zostaje na jeden z globalnych adresów, jakie ma do dyspozycji urządzenie NAT; z perspektywy Internetu pakiet wydaje się więc wygenerowany w węźle opatrzonym adresem globalnym. Gdy do urządzenia NAT trafia pakiet stanowiący odpowiedź na pakiet, o którym mowa w poprzednim zdaniu, translacja odbywa się w kierunku przeciwnym: globalny adres docelowy zostaje przez NAT skonwertowany do postaci oryginalnego adresu lokalnego, dzięki czemu odpowiedź trafia do właściwego adresata.

Większość urządzeń NAT wykonuje zarówno opisaną translację adresów, jak i *filtrowanie pakietów*. Kryteria tego filtrowania zależne są od aktualnego stanu NAT i mogą mieć zróżnicowaną granulację — przykładowo traktowanie pakietów niepowiązanych (czyli pakietów przychodzących z Internetu, niebędących odpowiedziami na pakiety wcześniej wysłane „z wnętrza") uzależnione jest zwykle od adresów IP (źródłowego i docelowego) oraz numerów portów (źródłowego i docelowego) zawartych w pakiecie. Wspomniane kryteria mogą być nie tylko inne dla różnych urządzeń NAT, lecz także zróżnicowane w czasie dla tego samego urządzenia NAT. Stawia to poważne wyzwania przed autorami aplikacji, jeśli te mają być zdolne do współdziałania z wieloma różniącymi się implementacjami NAT.

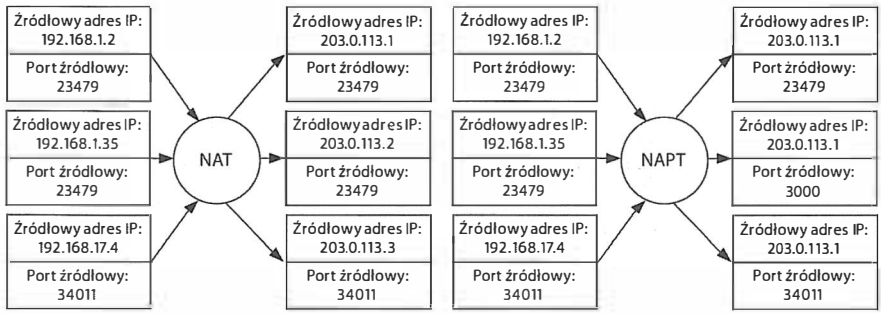


Rysunek 7.3. *NAT oddziela prywatne adresy i wykorzystujący je system od Internetu. Pakiety zawierające w nagłówkach adresy prywatne nie mogą być trasowane w Internecie bezpośrednio, lecz muszą być odpowiednio konwertowane do postaci adresów globalnych w urządzeniu NAT. W rezultacie dla hostów w Internecie pakiety te wyglądają tak, jakby wygenerowane zostały pod globalnymi adresami*

7.3.1. NAT podstawowe i NAPT

Szczegóły funkcjonowania NAT pozostawały niesprecyzowane przez wiele lat, niemniej jednak na podstawie obserwowania wielu konkretnych implementacji można pokusić się o coś na kształt ich taksonomii. I tak, w ramach *tradycyjnego NAT* wyróżnić można dwie odmiany: *NAT podstawowe* oraz *NAT „portowe"*, oznaczane skrótem NAPT (od *Network And Port Translation*, patrz [RFC3022]). W ramach podstawowego NAT konwertowaniu podlegają jedynie adresy IP zawarte w pakietach — adresy prywatne konwertowane są na adresy globalne i vice versa. Przyczynek do oszczędnego wykorzystywania adresów IP zasadza się na spostrzeżeniu, że zazwyczaj nie wszystkie hosty należące do sieci korzystają z Internetu jednocześnie, więc w danej chwili tylko niektóre z nich wymagają konwersji adresów prywatnych na globalne. Liczba globalnych adresów przydzielonych do dyspozycji urządzeniu NAT może być więc wystarczająca do funkcjonowania sieci w typowych warunkach i jednocześnie znacząco mniejsza od liczby hostów w tej sieci. W bardziej zaawansowanym i jednocześnie znacznie popularniejszym wariancie NAPT przedmiotem manipulacji stają się, oprócz adresów IP, także (zawarte w pakiecie) identyfikatory warstwy transportowej (czyli m.in. porty TCP i UDP oraz identyfikatory zapytań ICMP), tak jak na rysunku 7.4. Dzięki dodatkowemu czynnikowi różnicującemu, zapotrzebowanie urządzenia NAT na globalne adresy znacząco maleje: nawet w sytuacji kilku tysięcy hostów wymagających równoczesnego dostępu do Internetu wystarczający staje się zbiór kilku zaledwie globalnych adresów — w wielu konfiguracjach jest to nawet pojedynczy adres! Przyczynek do oszczędnego eksplorowania zasobu globalnych adresów IP jest tu więc iście drastyczny. Dalej w tym rozdziale będziemy używać terminu NAT na określenie obu wymienionych wariantów, wyraźnie zaznaczając sytuacje, w których istotne będzie ich rozróżnienie.

Prywatne adresy hostów w sieci interpretowane są jedynie przez urządzenie NAT i niewidoczne nigdzie na zewnątrz, mogą być więc w zasadzie wybierane dowolnie. Dowolność ta wiąże się jednak z pewnym niebezpieczeństwem: jeśli któryś host będzie chciał odwołać się do pewnego globalnego adresu w Internecie, a adres ten używany będzie wewnątrz sieci jako adres prywatny, wspomniany host uzyska połączenie ze swym lokalnym



Rysunek 7.4. *NAT w wariancie podstawowym (lewa strona rysunku) oznacza mapowanie lokalnych adresów IP na adresy globalne z dostępnej puli, bez zmiany numerów portów. W wariancie NAPT, znanym także pod nazwą maskarady, konwersji podlegają nie same adresy, lecz pary „adres:port". W sytuacji przedstawionej po prawej stronie rysunku, gdzie dostępny jest tylko jeden globalny adres, w wersji podstawowej NAT pierwsze i drugie połączenie zostałoby odwzorowane na parę 230.0.113.1:23479, co spowodowałoby utratę jednoznaczności mapowania*

partnerem, a nie z serwerem w Internecie — adresy lokalne *przesłaniają* swe (ewentualne) odpowiedniki w Internecie! W celu zapobieżenia takim konfliktom wydzielono specjalne zakresy adresów IPv4 przeznaczonych wyłącznie do adresowania prywatnego (patrz [RFC1918] oraz tabela 2.7 w rozdziale 2.): `10.0.0.0/8`, `172.16.0.0/12` i `192.168.0.0/16`. Zakresy te stosowane są też często jako domyślne dla puli przydziałów we wbudowanych serwerach DHCP (o których obszernie pisaliśmy w rozdziale 6.).

Jak wcześniej wspominaliśmy, NAT oferuje pewien stopień bezpieczeństwa, podobnie jak firewall. Domyślnie wszystkie hosty wewnątrz prywatnej sieci ukryte są przed Internetem; w większości konfiguracji identyfikowane są one wyłącznie za pomocą prywatnych adresów, w konsekwencji czego ich komunikacja z Internetem może odbywać się jedynie za pośrednictwem urządzenia NAT, zgodnie ze zdefiniowanymi regułami. Choć reguły te mogą być bardzo zróżnicowane, najczęściej sprowadzają się do liberalnego traktowania pakietów wychodzących i powrotnych (czyli stanowiących odpowiedzi na wysłane pakiety wychodzące), a niemal konsekwentnego odrzucania nadchodzących z zewnątrz prób inicjowania nowych połączeń. Stanowi to zabezpieczenie przed „próbkowaniem" sieci wewnętrznej, czyli eksplorowaniem używanych w niej adresów prywatnych, określaniem liczby hostów itp. — wielu użytkowników pragnie utrzymywać w tajemnicy topologię swych sieci, w czym urządzenia NAT wydatnie im pomagają.

Ponieważ NAT wiąże się z dość głęboką ingerencją w zawartość forwardowanych pakietów, nie sposób omawiać jego elementów funkcjonalnych w oderwaniu od konkretnych protokołów, których prawidłowej obsługi się wymaga. Celowe więc będzie omówienie NAT w kontekście najczęściej używanych protokołów transportowych oraz w kontekście mieszanych środowisk IPv4/IPv6. Wiele z behawioralnych aspektów NAT stało się przedmiotem badań i opracowań na forum grupy roboczej IETF o nazwie BEHAVE (to skrót od *Behavior Engineering for Hindrance Avoidance*, dosł. „inżynieria zachowań [zmierzająca do] unikania przeszkód"). Od roku 2007 grupa ta opublikowała serię dokumentów wyjaśniających i ujednoznaczniających wiele aspektów zachowania NAT. Dokumenty te zalecane są jako wskazówki dla twórców aplikacji oraz implementacji NAT, w celu zapewnienia spójnej współpracy między nimi.

7.3.1.1. NAT a TCP

TCP to podstawowy protokół transportowy Internetu, działający w trybie połączeniowym i identyfikujący każdy z „końców" połączenia za pomocą pary „adres IP:numer portu" (znaczenie numerów portów wyjaśnialiśmy w punkcie 1.3.3). Połączenie TCP jest więc jednoznacznie identyfikowane przez czwórkę „źródłowy adres IP, port źródłowy, docelowy adres IP, port docelowy".

Nawiązywanie połączenia TCP rozpoczyna się od wysłania pakietu synchronizacyjnego SYN przez jednego z uczestników, zwanego „uczestnikiem czynnym" lub klientem, do drugiego, „biernego" uczestnika zwanego serwerem. Odpowiedzią serwera jest pakiet SYN zawierający potwierdzenie (ACK); klient po odebraniu tego pakietu wysyła do serwera pakiet ACK. Zrealizowanie tego scenariusza, zwanego *trójstopniowym uzgadnianiem* (*three-way handshake*), powoduje nawiązanie połączenia. W podobny sposób przebiega scenariusz rozwiązywania połączenia (czyli po prostu rozłączania), rozpoczynający się od wysłania pakietu FIN przez jednego z uczestników; połączenie można też zakończyć przez wysłanie pakietu resetującego (RST). O szczegółach połączeń TCP piszemy w rozdziale 13. Wymagania dotyczące zachowania się NAT w kontekście protokołu TCP, opisane w dokumencie [RFC5382], koncentrują się głównie wokół trójstopniowego uzgadniania.

Powróćmy do rysunku 7.3 i załóżmy nawiązywanie połączenia TCP z inicjatywy bezprzewodowego klienta o adresie 10.0.0.126, kierowane do serwera www.isoc.org o adresie IPv4 212.110.167.157. Klient wysyła swe żądanie na porcie 9200, kierując je do portu 80 (WWW) na serwerze. Ze strony klienta połączenie jest więc identyfikowane przez parę 10.0.0.126:9200 (w tej notacji, której konsekwentnie będziemy używać w dalszym ciągu, numer portu oddzielony jest dwukropkiem od adresu IP). Analogicznie, po stronie serwera połączenie identyfikowane jest przez parę 212.110.167.157:80. Urządzenie NAT, które spełnia dla klienta rolę domyślnego routera, odbiera pierwszy pakiet, rozpoznaje go jako żądanie nowego połączenia (na podstawie ustawionego bitu SYN w nagłówku TCP — patrz rozdział 13.) i poddaje wstępnej obróbce, zastępując źródłowy adres 10.0.0.126 globalnym trasowalnym adresem 63.204.134.177. Pakiet TCP, identyfikowany teraz przez czwórkę (63.204.134.177:9200, 212.110.167.157:80), zostaje wysłany do Internetu. Oczywiście, wszystko to dzieje się pod warunkiem, że urządzenie NAT zaakceptuje otrzymany od klienta pakiet, co jest prawie pewne wobec generalnie liberalnego traktowania pakietów wychodzących. Jednocześnie w tablicach NAT dodana zostanie nowa pozycja (zwana *odwzorowaniem NAT*) uwzględniająca powstanie obsługi nowego połączenia TCP, czyli nowej *sesji NAT*; jako minimum niezbędne do przetworzenia przyszłej odpowiedzi informacja ta zawierać musi adres prywatny klienta i użyty przez niego port źródłowy. Odpowiedź serwera kierowana jest na adres 63.204.134.177 na port 9200; na podstawie numeru portu urządzenie NAT odnajduje w swych tablicach adres prywatny klienta i po wpisaniu tego adresu do pakietu (w miejsce oryginalnego adresu docelowego) przekazuje ten pakiet właściwemu klientowi. Od tej pory między klientem a serwerem istnieje połączenie TCP, choć naprawdę jest to połączenie między urządzeniem NAT a docelowym serwerem, ale rozróżnienie to jest dla klienta raczej nieistotne. Zwróćmy uwagę, że ingerencja NAT ogranicza się tylko do nadpisywania adresów IP, numer portu klienckiego pozostał niezmienny; cecha ta nazywa się potocznie *zachowywaniem portów* (*port preservation*).

Zakończenie sesji NAT jest problemem nieco bardziej skomplikowanym. Oczywiście, sygnałem do takiego zakończenia mogłoby być dla NAT odebranie pakietu `FIN` lub `RST`, lecz nie zawsze połączenie TCP kończy się w sposób tak jawny — możliwe są przecież zaniki zasilania, zresetowanie komputera itp. Pożądane jest więc implementowanie przez NAT mechanizmów ułatwiających wykrywanie takich *martwych sesji*.

Większość urządzeń NAT wyposażona jest w uproszczoną implementację nawiązywania połączenia TCP. Po odebraniu od klienta pakietu `SYN` uruchomiony zostaje *stoper połączenia* (*connection timer*); gdy w założonym interwale czasowym (wyznaczonym przez stoper) nie nadejdzie odpowiedź od serwera, próba nawiązania połączenia uznawana jest za niebyłą, w przeciwnym razie stoper połączenia zostaje zatrzymany, uruchomiony zostaje natomiast *stoper sesji* (*session timer*) odmierzający znacznie dłuższy interwał czasu, liczony przeważnie w godzinach. Gdy interwał ten upłynie, urządzenie NAT może jeszcze upewnić się, że odnośna sesja istotnie jest martwa, wysyłając do klienta dodatkowy *pakiet próbkujący*; otrzymanie od klienta odpowiedzi `ACK` świadczyć będzie o tym, że mimo wszystko sesja jest wciąż aktywna, w wyniku czego NAT uruchomi stoper sesji na nowo. Gdy jednak klient nie wyśle potwierdzenia w założonym interwale (określonym przez trzeci stoper — *stoper zamykania* [*close timer*]), sesja jest ostatecznie zamykana, a informacja o jej stanie kasowana bezpowrotnie.

Dokument [RFC5382] autorstwa grupy BEHAVE zawiera zalecenie takiego skonfigurowania połączenia TCP, by co pewien czas (domyślnie co 2 godziny) wysyłane były pakiety *keepalive* świadczące o jego żywotności (patrz rozdział 17.); w przeciwnym razie połączenie TCP musi być domyślnie uważane za aktywne. Gdy jednak połączenie jest otwierane lub zamykane, okres bezczynności ograniczony jest do 4 minut — tyle czasu mają wysłane pakiety na dotarcie do urządzenia NAT. W konsekwencji, na mocy jednego z wymagań (REQ-5) wspomnianego dokumentu, urządzenie NAT ma prawo uznać nawiązaną wcześniej sesję za martwą i zamknąć ją po okresie jej bezczynności nie krótszym niż 2 godziny i 4 minuty, natomiast w przypadku sesji połowicznie otwartej lub będącej w trakcie zamykania okres ten skraca się do 4 minut.

Jednym z najtrudniejszych aspektów współdziałania NAT z protokołem TCP jest bezpośrednia komunikacja aplikacji peer-to-peer rezydujących na komputerach znajdujących się w różnych sieciach prywatnych, czyli komputerach chronionych przez różne urządzenia NAT (patrz [RFC5128]). Wiele z tych aplikacji wykonuje operację *równoczesnego otwierania* (*simultaneous open*): każda z aplikacji, działając jako klient, wysyła pakiet `SYN` mniej więcej równocześnie z innymi. Protokół TCP przygotowany jest na taką sytuację — wysyłanie odpowiedzi w postaci pakietów `SYN + ACK` powoduje szybsze, w porównaniu z trójstronnym uzgadnianiem, zakończenie negocjacji. Niestety, wiele istniejących implementacji NAT nie radzi sobie z takim obrotem sprawy; w cytowanym już dokumencie [RFC5382] znajduje się w związku z tym wymaganie (etykietowane REQ-2), by implementacja NAT honorowała wszelkie poprawne wymiany pakietów TCP, a szczególnie wspomniane równoczesne otwieranie, bo pełni ono istotną rolę w działaniu wielu aplikacji peer-to-peer (w większości gier sieciowych). Jednocześnie dokument ten zaleca ignorowanie (bez ostrzeżenia) przychodzących z zewnątrz pakietów o niewiadomym dla NAT pochodzeniu. I tu pojawia się problem, bo w sytuacji równoczesnego otwierania zewnętrzny pakiet `SYN` może nadejść do urządzenia NAT szybciej niż ten od lokalnego komputera — wbrew pozorom, jest to bardziej prawdopodobne, niż się wydaje, gdy weźmie się pod uwagę fakt niezsynchronizowania zegarów w poszczególnych komputerach

klienckich. Rozwiązanie problemu także jest prostsze, niż można by się spodziewać: po odrzuceniu zewnętrznego pakietu SYN urządzenie NAT oczekuje przez 6 sekund na pojawienie się komplementarnego pakietu SYN ze strony lokalnego klienta; jeśli pakiet taki nie nadejdzie, urządzenie NAT sygnalizuje zewnętrznemu hostowi błąd otwarcia.

7.3.1.2. NAT a UDP

Wymogi dotyczące współpracy NAT z adresami unicast UDP sformułowane zostały w dokumencie [RFC4787]. Pakiety UDP stwarzają w kontekście NAT wiele tych samych problemów, co pakiety TCP, z jedną istotną różnicą: protokół UDP jest protokołem bezpołączeniowym, nie występuje więc jawne nawiązywanie i rozwiązywanie połączenia — w pakietach UDP nie ma znaczników odpowiadających bitom SYN, FIN czy RST. Pakiet identyfikowany jest nie przez cztery, lecz tylko przez dwa elementy — adres IP i numer portu nadawcy. W tych warunkach z urządzenia NAT usuwane są informacje dotyczące tych wiązań, które nie były „ostatnio" używane — z każdym wiązaniem skojarzony jest stoper, zwany *stoperem mapowania* (*mapping timer*). Istnieje znacząca rozbieżność pomiędzy implementacjami w kwestii rozumienia pojęcia „ostatnio": w dokumencie [RFC4787] wymaga się ustawienia wspomnianego stopera na interwał co najmniej 2-minutowy i zaleca jednocześnie wartość 5 minut jako optymalną. Dokument ten określa również wymaganie, by stoper ten był obowiązkowo resetowany w momencie nadejścia pakietu z wnętrza sieci (oczywiście, pakietu o danym identyfikatorze „adres IP:port"); jako opcję rozważa się resetowanie także w przypadku nadejścia rzeczonego pakietu z zewnątrz.

Wspominaliśmy już w rozdziale 5., że datagramy IP (a więc także pakiety UDP enkapsulowane w datagramach IP) mogą być fragmentowane (w rozdziale 10. zajmiemy się szczegółowo tą kwestią). Każdy z fragmentów datagramu IP traktowany jest — co prawda — jako oddzielny datagram, lecz jednocześnie w kontekście NAT pojawia się nowy problem — ten mianowicie, że nagłówek protokołu warstwy wyższej (w tym przypadku UDP) znajduje się jedynie w pierwszym fragmencie, następne fragmenty nie zawierają już tego, co dla NAT najistotniejsze — adresu IP i numeru portu. A to generalnie oznacza niemożność poprawnej obsługi fragmentacji przez NAT, bez zastosowania dodatkowych mechanizmów.

7.3.1.3. NAT a inne protokoły transportowe (DCCP i SCTP)

Poza najpopularniejszymi protokołami Internetu — TCP i UDP — jeszcze dwa inne doczekały się formalnego sprecyzowania zachowania NAT w ich kontekście. Pierwszy z nich to opisywany w dokumencie [RFC4340] protokół DCCP (*Datagram Congestion Control Protocol* — protokół kontrolowania przeciążeń [w transmisji] datagramów): jego współdziałanie z NAT jest treścią dokumentu [RFC5597], natomiast dokument [RFC5596] definiuje jego modyfikację pod kątem obsługi procedury równoczesnego otwierania, podobnie do TCP. Drugi ze wspomnianych protokołów to SCTP (*Stream Control Transmission Protocol* — protokół sterowania transmisją strumieni), zdefiniowany w dokumencie [RFC4960]. Jest to protokół transportowy podobny do TCP, lecz umożliwiający równoległe przesyłanie wielu strumieni danych w ramach pojedynczego połączenia. Szczegóły jego współdziałania z NAT opisane są w publikacjach [HBA09] i [IDSNAT].

7.3.1.4. NAT a ICMP

ICMP (*Internet Control Message Protocol* — internetowy protokół komunikatów kontrolnych), opisywany szczegółowo w rozdziale 8., dostarcza informacji o statusie pakietów IP, może być także wykorzystywany w sieci do celów diagnostycznych i pomiarowych. Zachowanie NAT w kontekście protokołu ICMP zdefiniowane zostało w dokumencie [RFC5508] i obejmuje dwa rodzaje komunikatów — komunikaty informacyjne i komunikaty o błędach.

Komunikat o błędzie zawiera zwykle kopię (pełną lub częściową) pakietu uważanego za przyczynę błędu; wysyłany jest z węzła, w którym błąd stwierdzono, do nadawcy pakietu. Wydaje się to nieskomplikowane, lecz przestaje takie być, gdy NAT wkracza do gry, bo zawarte we wspomnianym pakiecie informacje identyfikacyjne (adresy IP i być może numery portów) muszą zostać przez NAT skonwertowane do postaci lokalnej, charakterystycznej dla nadawcy, dla którego w przeciwnym razie treść komunikatu nie miałaby większego sensu (operacja ta we wspomnianym dokumencie nazwana została *ICMP fix-up*).

Podobny problem występuje w przypadku komunikatów informacyjnych, większość z nich jednak to odpowiedzi serwerów na konkretne żądania klientów, rozróżniane za pomocą unikatowych identyfikatorów żądania (*Query ID*), pełniących rolę podobną do numerów portów TCP i UDP. Urządzenie NAT, rozpoznając wychodzący pakiet żądania ICMP, rejestruje nową sesję żądania ICMP (*ICMP Query Session*) i uruchamia związany z nią stoper wyznaczający limit czasowy oczekiwania na nadejście odpowiedzi. Wspomniany dokument zawiera zalecenie, by wartość tego limitu była konfigurowalna oraz wymaganie, by nie była ona mniejsza niż 60 sekund.

7.3.1.5. NAT a pakiety tunelowane

Gdy ruch związany z tunelowaniem (o którym pisaliśmy w rozdziale 3.) poddawany jest ingerencji ze strony NAT, sprawy komplikują się jeszcze bardziej, bo ingerencja ta nie ogranicza się do zewnętrznych nagłówków IP, lecz musi także obejmować nagłówki pakietów tunelowanych, stanowiących ładunek użyteczny. Przykładowo pole *Identyfikator partnera* w nagłówku GRE (patrz rysunek 3.27) może — przy przechodzeniu pakietu przez NAT — powodować kolizję z innymi połączeniami. Jak łatwo sobie wyobrazić, dodanie kolejnego poziomu tunelowania przekłada się na kolejny stopień komplikacji w funkcjonowaniu NAT.

7.3.1.6. NAT a multicasting

Rozpatrywaliśmy dotychczas funkcjonowanie NAT w odniesieniu do adresów unicast, lecz urządzenia NAT można także przystosować do obsługi ruchu multicast (o czym piszemy w rozdziale 9.), choć czyni się to raczej rzadko. Związane z tym wymagania zawarte są w dokumencie [RFC5135], przedstawiającym koncepcję rozszerzenia oryginalnego NAT o proxy realizujące protokół IGMP (opisywany w dokumencie [RFC4605] oraz w rozdziale 9.). W pakietach przybywających z zewnątrz urządzenie NAT *nie modyfikuje* docelowych adresów IP i docelowych numerów portów; w pakietach wychodzących modyfikacja taka odbywa się, podobnie jak w pakietach unicast UDP.

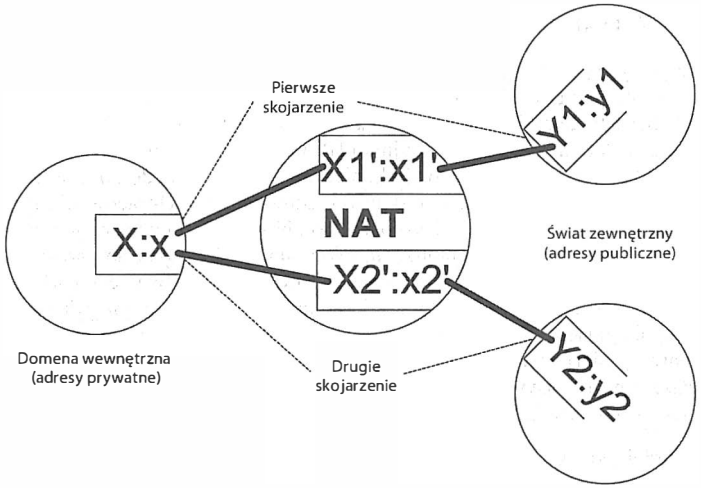
7.3.1.7. NAT a IPv6

Wziąwszy pod uwagę ogromną popularność NAT w odniesieniu do IPv4, nie sposób nie zadać pytania o jego przyszłość w kontekście coraz większego rozpowszechnienia IPv6. Kwestia ta budzi wiele kontrowersji, stanowisko IAB w tej sprawie przedstawione zostało w dokumencie informacyjnym [RFC5902]. Zdaniem wielu projektantów protokołów, NAT pojawiło się jako zło konieczne — konieczne do tymczasowego oddalenia perspektywy wyczerpania się dostępnych adresów IPv4. Opracowanie IPv6 oddaliło tę perspektywę praktycznie w nieskończoność, NAT straciło więc rację bytu — zwłaszcza gdy rozważy się ogrom komplikacji, jakie wprowadziło ono do przejrzystej (w miarę) architektury Internetu. Komplikacje te stanowią zbyt wysoką cenę także z perspektywy drugiego fundamentu NAT, jakim są jego funkcje firewalla, umożliwiające m.in. ukrywanie topologii sieci i ochrony prywatności: funkcje te można realizować równie efektywnie (a nawet lepiej) przy użyciu alternatywnego rozwiązania o nazwie *Local Network Protection* (LNP), opisywanego w dokumencie [RFC4864], a zapewniającego kolekcję opartych na IPv6 technik rozszerzających wspomnianą funkcjonalność NAT.

Z drugiej jednak strony, nie sposób nie doceniać korzyści płynących z odseparowania wewnętrznej domeny adresowej od „reszty świata" i dokonujących się w niej zmian wynikających np. ze zmiany dostawcy Internetu. Odpowiednia konwersja adresów „na brzegu" sieci (czyli w urządzeniu NAT) może być też wielce pomocna w komunikowaniu się różnych domen adresowych; przymiotnik „odpowiednia" oznacza tu przede wszystkim brak komplikacji wynikających z ingerowania w treść przesyłanych pakietów — komplikacji tak charakterystycznych i nieuniknionych w IPv4. Jedną z realizacji tego założenia (a właściwie — próbą realizacji, bo mechanizm znajduje się wciąż w stadium eksperymentalnym) jest technika unikatowych lokalnych adresów unicast IPv6 (*Unique Local IPv6 Unicast Addresses*, w skrócie ULA — patrz [RFC4193]), generowanych w drodze konwersji prefiksów, zwanej *NPTv6* (od *Network Prefix Translation*) opisywanej w dokumencie [RFC6296]. Konwersja między adresami różnych domen (czyli de facto między różnymi prefiksami) wykonywana jest na drodze algorytmicznej, bez wykorzystywania jakiejkolwiek informacji o stanie połączenia, nieodłącznej w tradycyjnym NAT. Podstawową zaletą tej konwersji (oprócz — oczywiście — bezstanowego charakteru) jest modyfikowanie adresów IP w taki sposób, że nie zmieniają się sumy kontrolne enkapsulowanych pakietów TCP i UDP. Redukuje to znacznie złożoność translacji, ponieważ nie jest konieczne modyfikowanie danych pakietów powyżej warstwy sieciowej, nie są ponadto wykorzystywane numery portów warstwy transportowej. Jednakże aplikacje wykorzystujące zewnętrzne adresy nadal muszą uwzględniać istnienie NAT bądź korzystać z bram aplikacyjnych (ALG). Ponadto NPTv6 samo z siebie nie oferuje żadnego mechanizmu filtrowania pakietów ani innych elementów funkcjonalności firewalla, więc brak ten musi zostać uzupełniony przez wdrożenie dodatkowych mechanizmów tej kategorii.

7.3.2. Klasy behawioralne translacji adresów i portów

Szczegóły działania poszczególnych implementacji NAT dość znacznie różnią się między sobą, głównie w zakresie sposobu odwzorowywania adresów i (ewentualnie) numerów portów. Jednym z celów, jakie postawiła sobie grupa robocza BEHAVE, było skategoryzowanie tych mechanizmów i zaproponowanie ich optymalnych wariantów. Aby to lepiej zrozumieć, przyjrzyjmy się rysunkowi 7.5, przedstawiającemu w ogólny sposób ideę mapowania NAT. Notacja *X*:*x* oznacza połączenie adresu IP *X* z portem o numerze *x* (w przypadku protokołu ICMP *x* jest identyfikatorem żądania).



Rysunek 7.5. *Szczegóły mapowania NAT decydują o zaliczeniu go do jednej z klas behawioralnych. W sytuacji przedstawionej na rysunku host identyfikowany lokalnie jako $X : x$ (czyli za pomocą adresu IP X oraz portu x) inicjuje dwa połączenia zewnętrzne, z hostami identyfikowanymi globalnie jako $Y_1 : y_1$ oraz $Y_2 : y_2$. Na potrzeby pierwszego połączenia lokalnego identyfikacja $X : x$ skonwertowana zostaje do postaci globalnej $X_1' : x_1'$, na potrzeby drugiego połączenia — do postaci globalnej $X_2' : x_2'$. Jeśli $X_1' : x_1'$ zawsze jest równe $X_2' : x_2'$, czyli wynik mapowania zależny jest wyłącznie od jego argumentu, odwzorowanie należy do klasy behawioralnej niezależności od punktu docelowego. Jeśli wynik mapowania zależny jest od identyfikacji lokalnej oraz adresu (lecz nie portu) docelowego, wówczas równość $X_1' : x_1' = X_2' : x_2'$ gwarantowana jest tylko wtedy, gdy $Y_1 = Y_2$, odwzorowanie należy wówczas do klasy zależności od adresu docelowego. Gdy wynik odwzorowania zależny jest od obu par $X : x$ oraz $Y : y$, odwzorowanie to należy do klasy pełnej zależności od punktu docelowego. Ponadto, gdy NAT dysponuje pulą wielu adresów globalnych, reguły odwzorowania wiązać mogą ze sobą adresy $X' = Y'$: gdy temu samemu adresowi Y odpowiada zawsze ten sam adres X', mówimy o parowaniu adresów*

Zasadniczo X jest jednym z prywatnych adresów, z zakresów definiowanych w [RFC1918]. W celu osiągnięcia portu y w węźle o adresie IP Y NAT tworzy (w swych wewnętrznych tablicach) odwzorowanie pary $X : x$ na parę $X_1' : x_1'$, gdzie X_1' jest (zwykle) publicznym trasowalnym adresem IP, natomiast x_1' jest wybranym przez NAT numerem portu. Jeżeli wewnętrzny host o adresie X żąda połączenia na porcie x z portem y_1 zdalnego hosta o adresie Y_1 — czyli połączenia $X : x \rightarrow Y_1 : y_1$ — a wkrótce potem połączenia $X : x \rightarrow Y_2 : y_2$ (czyli na tym samym porcie źródłowym), w ogólnym przypadku urządzenie NAT odwzorowuje parę $X : x$ na parę $X_1' : x_1'$ na potrzeby pierwszego połączenia i na parę $X_2' : x_2'$ na potrzeby drugiego. Najczęściej $X_1' = X_2'$, bo NAT dysponuje tylko jednym globalnym adresem; jeżeli ponadto $x_1' = x_2'$, to mówimy o powtórnym wykorzystaniu (*reusing*) mapowania. W szczególnym przypadku, gdy $x_1' = x_2' = x$, mamy do

czynienia ze wspomnianym wcześniej zachowywaniem portów. Zachowywanie portów nie zawsze jest wykonalne, przykładowo w sytuacji przedstawionej w prawej części rysunku 7.4 (gdzie dostępny jest tylko jeden adres globalny — 230.0.113.1) zachowywanie portu 23479 doprowadziłoby do niejednoznaczności, czyli nierozróżniania żądań pochodzących od adresów 192.168.1.2 i 192.168.1.35; NAPT wykonuje w związku z tym nadpisanie numeru portu dla drugiego połączenia.

W tabeli 7.1 przedstawiono podział (wzorowany na [RFC4787]) możliwego zachowania NAT na trzy grupy, charakteryzujące się różnym podejściem do mapowania źródłowej pary „adres IP-port" zależnie od okoliczności. Tak się składa, że klasyfikacja ta pozostaje w ścisłym związku z różnymi strategiami filtrowania pakietów (które omawiać będziemy w punkcie 7.3.3) we wspomnianych okolicznościach, przedstawiamy więc łącznie oba aspekty funkcjonowania NAT — translacyjny i filtracyjny. Od wszystkim popularnych protokołów transportowych — m.in. TCP i UDP — wymaga się, by aspekt translacyjny należał do pierwszej kategorii (niezależność od punktu docelowego), podobna cecha zalecana jest także w odniesieniu do ICMP. Wymaganie to uzasadnione jest względami bardziej niezawodnego funkcjonowania aplikacji próbujących określać zewnętrzne adresy dla swoich połączeń — powrócimy do tej kwestii w podrozdziale 7.4.

Tabela 7.1. *Ogólna klasyfikacja zachowania NAT w aspekcie translacji adresów i filtrowania pakietów*

Klasa behawioralna NAT	Aspekt translacyjny	Aspekt filtracyjny
Niezależność od punktu docelowego	$X_1' : x_1' = X_2' : x_2'$ dla wszystkich $Y_i : y_j$ (wymagane)	Akceptowanie wszystkich pakietów kierowanych do $X : x$, pod warunkiem że zdefiniowano mapowanie dla adresu docelowego $X_1' : x_1'$ (zalecane dla większej przezroczystości NAT).
Zależność od adresu docelowego	$X_1' : x_1' = X_2' : x_2'$, jeżeli $Y_1 = Y_2$	Akceptowanie wszystkich pakietów kierowanych do $X : x$ przychodzących z $Y_1 : y_1$, pod warunkiem że X kontaktował się z Y_1 (zalecane dla bardziej rygorystycznego filtrowania).
Zależność od adresu docelowego i portu docelowego	$X_1' : x_1' = X_2' : x_2'$, jeżeli $Y_1 : y_1 = Y_2 : y_2$	Akceptowanie wszystkich pakietów kierowanych do $X : x$ przychodzących z $Y_1 : y_1$, pod warunkiem że X kontaktował się z $Y_1 : y_1$

Urządzenie NAT może mieć do dyspozycji jeden lub więcej globalnych adresów IP — ich zbiór nazywany jest *pulą adresową NAT*. Jest to zbiór inny niż pula przydziałowa DHCP (pisaliśmy o tym w rozdziale 6.), choć oba zbiory mogą być wykorzystywane przez pojedyncze urządzenie łączące funkcje NAT i DHCP. W sytuacji gdy pula adresowa NAT liczy więcej niż jedną pozycję, powstaje pytanie, czy dany host inicjujący kilka równoczesnych połączeń zawsze otrzyma dla każdego z nich ten sam globalny adres IP, czy też różne adresy dla każdego z połączeń? W pierwszym przypadku mamy do czy-

nienia z *parowaniem adresów* (*address pairing*), w drugim strategia translacyjna nazywana jest *arbitralną*. Strategia arbitralna może sprawiać problem polegający na tym, że jeżeli kilka połączeń wychodzących z danego hosta kierowanych jest do tego samego hosta zdalnego, ten ostatni może wyciągnąć błędny wniosek, iż każde z połączeń dotyczy innego urządzenia. Oczywiście, gdy NAT dysponuje tylko jednym adresem globalnym, ten problem nie występuje.

Niektóre z urządzeń NAT stosują ryzykowną technikę *przeciążania portów* (*port overloading*) polegającą na tym, że gdy kilka hostów równocześnie inicjuje połączenia, dla każdego z nich para „adres źródłowy-port" mapowana jest na tę samą wartość. Schemat ten działa poprawnie pod warunkiem, że każde z połączeń adresowane jest do innego punktu docelowego (czyli pary „adres docelowy-port"); w sytuacji gdy dwa połączenia kierowane są do tego samego punktu docelowego, niemożliwe staje się prawidłowe zaadresowanie ruchu powrotnego — urządzenie NAT nie jest w stanie rozróżnić, do którego z nadawców powinno kierować otrzymywane pakiety. Z oczywistych względów implementowanie takiej strategii mapowania jest obecnie niedopuszczalne.

Niektóre urządzenia NAT oferują opcję zachowywania *parzystości* portu — jeśli x jest portem o numerze parzystym (odpowiednio: nieparzystym), to port x' także jest portem o numerze parzystym (odpowiednio: nieparzystym). Cecha ta może być pomocna w niektórych aplikacjach stosujących specyficzne numerowanie portów: przykładem takiej aplikacji jest protokół RTP (*Real-Time Protocol*) oryginalnie wykorzystujący wiele portów, choć w dokumencie [RFC5761] opisana została technika multipleksowania jego ruchu do pojedynczego portu. Zachowywanie parzystości portów jest zalecaną, lecz niewymaganą cechą NAT; wydaje się, że jej znaczenie będzie coraz mniejsze w obliczu upowszechniania nowych metod omijania NAT (*NAT traversal*).

7.3.3. Zachowanie filtracyjne NAT

Ustanowienie sesji NAT dla połączenia TCP, skojarzenia UDP czy różnych form ruchu ICMP wiąże się nie tylko z mapowaniem adresów i portów; jeśli urządzenie NAT spełniać ma również funkcje firewalla — co jest dziś rozwiązaniem standardowym — konieczne staje się także określenie strategii traktowania pakietów przychodzących z zewnątrz.

Filtrowanie pakietów i mapowanie adresów są ze sobą ściśle powiązane: jeśli w danej chwili nie jest zdefiniowane żadne mapowanie, nie sposób określić, do którego z wewnętrznych hostów adresowany jest przychodzący pakiet, trzeba go więc (z braku lepszej alternatywy) po prostu odrzucić. Powiązanie filtrowania z mapowaniem adresów (portów) odzwierciedlone zostało także w identycznej klasyfikacji poszczególnych strategii (patrz tabela 7.1). Klasyfikację tę rozpoczyna *filtrowanie niezależne od punktu końcowego*: przychodzący pakiet jest akceptowany, jeżeli tylko zdefiniowane jest odwzorowanie między jego adresem-portem docelowym ($X' : x'$) a rzeczywistym adresem-portem wewnątrz sieci ($X : x$) — nieistotne jest pochodzenie pakietu ($Y : y$). W wariancie bardziej rygorystycznym — *filtrowaniu zależnym od adresu zewnętrznego* — pakiet przychodzący z adresu Y (niezależnie od portu) do punktu ($X' : x'$) akceptowany jest wtedy, gdy host wewnętrzny kontaktował się uprzednio na punkcie ($X : x$), odwzorowanym w ($X' : x'$), z dowolnym portem pod adresem Y. W wariancie najbardziej rygorystycznym — *filtrowaniu zależnym od punktu zewnętrznego* — istotny jest także

port zewnętrzny: pakiet przychodzący z punktu $Y:y$ do punktu ($X':x'$) jest mianowicie akceptowany tylko wtedy, gdy host wewnętrzny kontaktował się uprzednio na punkcie ($X:x$), odwzorowanym w ($X':x'$), z punktem ($Y:y$). Pakiet niespełniający warunku określonego przez bieżącą strategię filtrowania jest odrzucany.

7.3.4. Serwery w lokalnej domenie adresowej

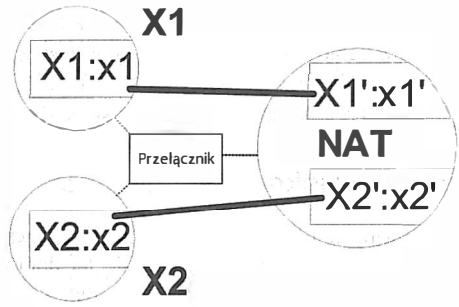
Ponieważ wewnętrzna domena adresowa zasłonięta jest przez urządzenie NAT przed światem zewnętrznym, należący do tej domeny host, który ma świadczyć określone usługi (np. usługi serwera WWW) poprzez Internet, nie jest w stanie tego robić bez udziału NAT z dwóch powodów. Powróćmy na chwilę do rysunku 7.3: po pierwsze, host o adresie 10.0.0.3 mający na porcie 80 funkcjonować jako serwer WWW jest standardowo niedostępny dla świata zewnętrznego, bowiem urządzenie NAT będzie konsekwentnie odfiltrowywać pakiety kierowane na adres docelowy 10.0.0.3. Po drugie, zgodnie z klasyfikacją adresów IPv4 (patrz tabela 2.7 w rozdziale 2.) adresy o prefiksie 10.0.0.0/8 nie są (jako adresy prywatne) w ogóle obsługiwane przez globalny mechanizm trasowania w Internecie.

Ten stan rzeczy można — oczywiście — zmienić przez odpowiednie skonfigurowanie urządzenia NAT. Należy mianowicie spośród dostępnych dla NAT adresów globalnych wybrać jeden — oznaczmy go S — i umieścić w tablicach odwzorowania oraz filtrowania statyczną pozycję w taki sposób, by pakiety przychodzące na adres docelowy S, z portem docelowym 80, kierowane były do lokalnego hosta pełniącego rolę serwera WWW. Serwer ten będzie wówczas widoczny w Internecie pod adresem globalnym S (i prawdopodobnie pod nazwą mnemoniczną DNS skojarzoną z tym adresem). Oczywiście, gdy urządzenie NAT dysponuje tylko jednym globalnym adresem, on właśnie będzie pełnił rolę adresu S. Opisana technika nazywa się *forwardowaniem portu* — urządzenie NAT „wyczulone" jest na określony numer (numery) portu docelowego w pakiecie, skutkujący forwardowaniem pakietu do odpowiedniego hosta lokalnego.

Nietrudno zauważyć podstawowe ograniczenie opisanej techniki; dla każdego z dostępnych globalnych adresów forwardowanie określonego portu możliwe jest tylko w odniesieniu do jednego lokalnego hosta (serwera): jeżeli więc urządzenie NAT dysponować będzie tylko jednym globalnym adresem, tylko jeden z lokalnych hostów będzie mógł świadczyć usługi na porcie 80. Gdy właściciel takiej sieci będzie chciał udostępnić większą liczbę serwerów WWW, będzie musiał posłużyć się innymi numerami portów (albo wystąpić do swego dostawcy Internetu o przydzielenie większej liczby globalnych adresów IP, jeśli każdy z serwerów koniecznie ma być dostępny na standardowym porcie 80).

7.3.5. Upinanie ruchu — pętla zwrotna NAT

Interesująca (z perspektywy NAT) sytuacja występuje, gdy klient zamierza połączyć się z serwerem znajdującym się po tej samej stronie NAT, czyli wewnątrz prywatnej sieci, tak jak na rysunku 7.6, gdzie $X_1:x_1$ identyfikuje stronę kliencką, zaś $X_2:x_2$ stronę serwerową. Oczywiście, jeśli informacja $X_2:x_2$ znana jest klientowi, może on połączyć się z serwerem bezpośrednio, za pomocą przełącznika, bez udziału NAT. Często jednak dostępna jest tylko *publiczna* informacja na temat serwera, na rysunku oznaczona $X_2':x_2'$,



Rysunek 7.6. *NAT implementujące upinanie: host X_1 odwołuje się do lokalnie dostępnego hosta (serwera) X_2 za pośrednictwem jego globalnej identyfikacji $X_2' : x_2'$, a nie bezpośrednio przez przełącznik*

wówczas klient odwołuje się do urządzenia NAT, które — na podstawie istniejącego mapowania między $X_2' : x_2'$ a $X_2 : x_2$ — kieruje żądanie klienckie do serwera X_2. Opisane zjawisko nosi nazwę *pętli zwrotnej NAT* (*NAT loopback*) lub (bardziej obrazowo) *upinania ruchu* (w analogii do upinania włosów — *hairpinning*).

Skoro do gry wchodzi NAT, a klient identyfikowany jest dwojako — w postaci oryginalnej $X_1 : x_1$ i w postaci „zamapowanej" $X_1' : x_1'$ — to natychmiast pojawia się interesujące pytanie, *która* z tych postaci użyta zostanie w charakterze adresu-portu źródłowego w pakiecie, który urządzenie NAT prześle do serwera? Ponieważ jednym z celów NAT jest skrywanie prywatnych adresów domeny, użyta zostanie postać zamapowana $X_1' : x_1'$ — takie wymaganie znajduje się w dokumencie [RFC5382], precyzującym działanie NAT w kontekście protokołu TCP. Wymaganie to uzasadnione zostało względami aplikacji, identyfikującymi swych partnerów na podstawie ich *globalnych* adresów — w naszym przykładzie serwer X_2 będzie raczej oczekiwał połączeń z adresu globalnego X_1' niż adresu lokalnego X_1, skoro mu wiadomo, że w tych połączeniach pośredniczyć będzie NAT.

7.3.6. Edytory NAT

Wśród protokołów, których PDU przenoszone są w Internecie za pomocą datagramów IP, palmę pierwszeństwa zdecydowanie dzierżą TCP i UDP. Oczywiście, szczegóły obsługi pakietów TCP i UDP przez NAT to zagadnienie dobrze poznane i jako takie nie stanowi przyczyny nadmiernej komplikacji, komplikacje pojawiają się natomiast w sytuacji, gdy w ładunku użytecznym tych pakietów znajdują się informacje charakterystyczne dla warstw niższych niż transportowa, np. adresy IP. Typowym przykładem aplikacji przenoszącej taką informację jest protokół przesyłania plików FTP (*File Transfer Protocol*) opisywany w dokumencie [RFC0959]. Aplikacja wykorzystuje dwa połączenia: jedno dla celów sterowania, drugie dla celów transferu zawartości plików; w ramach pierwszego z wymienionych połączeń transferowany jest do partnera adres IP i numer portu źródłowego, wykorzystywany do ustanowienia drugiego z wymienionych połączeń. Rzecz jasna, gdy w grę wchodzi NAT, informacje te muszą zostać nadpisane globalnymi wartościami „zamapowanymi" — oprócz rutynowej ingerencji w nagłówki IP i TCP konieczna jest więc także odpowiednia ingerencja w treść ładunku użytecznego. Implementacje NAT posiadające zdolność wykonywania takiej ingerencji nazywane są *edytorami NAT*. Jedną z subtelności kryjącą się za wspomnianą ingerencją jest konieczność

przewidzenia wszystkich skutków wykonywanych modyfikacji: jeżeli więc np. zmiany wprowadzane przez edytor NAT prowadzą do zmiany rozmiaru pakietu, konieczne jest odpowiednie zmodyfikowanie także numerów sekwencyjnych w danym pakiecie i pakietach następnych, bo protokół TCP wykorzystuje numery sekwencyjne do ścisłej kontroli liczby przesyłanych bajtów (o czym pisać będziemy w rozdziale 15.).

Innym przykładem aplikacji wymagającej ingerencji NAT w treść ładunku użytecznego pakietów jest (definiowany w [RFC2637]) protokół PPTP, o którym pisaliśmy w podrozdziale 3.9.

7.3.7. SPNAT — NAT w infrastrukturze dostawcy

Obserwowaną od kilku lat tendencją jest przenoszenie urządzeń NAT na wyższy poziom — z mieszkań (siedzib) klientów do infrastruktury dostawców Internetu (ISP). Spotyka się różne nazwy tej koncepcji — *Service Provider NAT* (SPNAT), *Carrier-Grade NAT* (CGN) lub *Large-Scale NAT* (LSN) — a jej celem jest dalsze spowolnienie wyczerpywania się i (tak już szczątkowego) zasobu dostępnych adresów IPv4 — pojedynczy adres IPv4 staje się wystarczający dla całej grupy klientów. Tak oto punkt agregacji adresów przenosi się o jeden poziom wyżej.

Z perspektywy klienta SPNAT nie różni się funkcjonalnie od klasycznego NAT, jednak pozbawienie klienta kontroli nad wykorzystywanym przez niego NAT ma swe konsekwencje użytkowe: klient odcięty zostaje od globalnego adresu (przydzielony mu przez dostawcę adres jest prywatnym adresem po „wewnętrznej" stronie NAT), wskutek czego ewentualne uruchomienie serwera dostępnego z Internetu wymaga porozumienia z ISP; klient pozbawiony zostaje możliwości własnego konfigurowania firewalla zintegrowanego z NAT, co stwarza wątpliwości w zakresie bezpieczeństwa (patrz [MBCB08]). Badania przeprowadzone w roku 2009 ([ANM09]) wykazują, że większość użytkowników akceptuje połączenia przychodzące, głównie w ramach aplikacji *peer-to-peer*.

SPNAT, jako technika znacznie zmniejszająca zapotrzebowanie na globalne adresy, to kolejne rozwiązanie o charakterze doraźnym, rozwiązaniem docelowym jest z założenia IPv6. Z różnych przyczyn (o których wcześniej wspominaliśmy) stopień rozpowszechnienia IPv6 jest wciąż daleki od oczekiwań. Początkowo ułatwieniem we wdrażaniu IPv6 miał być tzw. dualny stos (*dual-stack*), czyli równoległa implementacja IPv4 i IPv6 w każdym systemie — rozwiązanie z założenia traktowane jako tymczasowe, w efekcie czego zbyt późne jego upowszechnienie nie zdołało przyczynić się do spowolnienia tempa wyczerpywania dostępnych adresów IPv4. Jako prawdopodobnie bardziej pragmatyczne podejście rozważane jest obecnie łączenie tunelowania, translacji adresów i dualnego stosu w rozmaitych kombinacjach. W podrozdziale 7.6 omówimy kilka metod tej kategorii, po uprzednim zaprezentowaniu metod zmierzających do lepszego wykorzystywania istniejących NAT i jednocześnie radzenia sobie z problemami, których są przyczyną.

7.4. Omijanie NAT

Jako alternatywę dla skomplikowanych bram aplikacyjnych (ALG) i edytorów NAT aplikacja może we własnym zakresie realizować technikę *omijania NAT* (*NAT Traversal*). Zwykle sprowadza się to do próby uzyskania przez (lokalną) aplikację informacji o glo-

balnym adresie i numerze portu, na które NAT odwzoruje jej lokalne odpowiedniki, i odpowiedniego zmodyfikowania zawartości pakietów wychodzących — tak jak robią to edytory NAT. Jeśli aplikacja rozproszona jest po sieci, czyli obejmuje swym zasięgiem wiele klientów i serwerów, po części znajdujących się także na zewnątrz NAT, serwery te mogą być używane do kopiowania danych między klientami bądź też do wymiany między klientami informacji umożliwiającej ich bezpośrednie skomunikowanie się. Jak łatwo się domyślić, nieustanne „przerzucanie się" danymi przez klientów za pośrednictwem serwera to rozwiązanie raczej mało efektywne i podatne na rozmaite nadużycia, logicznie więc bardziej preferowane są podejścia ukierunkowane na bezpośrednią komunikację klientów.

Bezpośrednie komunikowanie się aplikacji klienckich jest popularną, stosowaną od dawna techniką w dziedzinie wymiany plików peer-to-per, w grach interaktywnych i w niektórych komunikatorach. Techniki tego komunikowania są jednak zwykle ograniczone do konkretnych aplikacji, co oznacza, że autorzy każdej nowej aplikacji rozproszonej zmuszeni są opracowywać własne metody omijania NAT, co — jak łatwo skonstatować — prowadzi, z jednej strony, do dublowania wysiłku programistycznego, z drugiej natomiast, wieszczy prawdopodobne problemy ze współdziałaniem różnych aplikacji w tej samej sieci. W celu rozwiązania tego problemu opracowano standardowe podejście do omijania NAT, zależne od kolekcji oddzielnych, dobrze zdefiniowanych protokołów, które omówimy w kolejnych punktach. Rozpoczniemy jednak od jednego z bardziej solidnych, choć nieuznanego jako standard, podejścia wykorzystywanego przez aplikacje rozproszone.

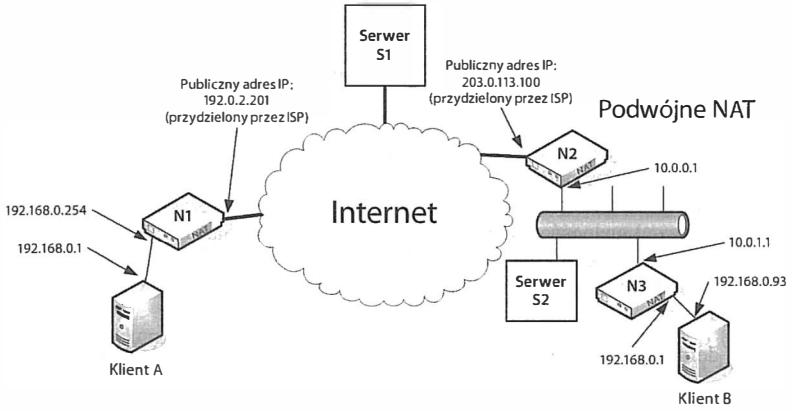
7.4.1. Otworki i wybijanie dziur

Typowe urządzenie NAT integruje funkcjonalność translacji adresów i filtrowania pakietów. Gdy dla danej aplikacji zdefiniowane zostanie odwzorowanie NAT, pakiety tej aplikacji będą mogły przepływać przez urządzenie NAT w obu kierunkach. Odwzorowanie to ma jednak charakter efemerydalny: dotyczy tylko jednej aplikacji i istnieje tylko przez czas jej wykonywania. Tego typu furtki dla pakietów nazywane są popularnie *otworkami* (*pinholes*) ze względu na niewielki zakres przepuszczanego ruchu, zwykle ograniczający się do określonej kombinacji par „adres IP-port". Zwykle otworki tworzone są i usuwane dynamicznie, w konsekwencji ustalonej komunikacji między programami.

Analogiczna metoda, pozwalająca na wykorzystywanie otworków do bezpośredniego komunikowania się dwóch (lub więcej) systemów chronionych urządzeniami NAT, nosi popularną nazwę „wybijania dziury" (*hole punching*) i opisywana jest w dokumencie [RFC5128] — w sekcji 3.3 dla protokołu UDP i w sekcji 3.4 dla protokołu TCP. Klienci zamierzający skomunikować się ze sobą kontaktują się najpierw ze znanym im, określonym serwerem, który doprowadza do wymienienia się przez nich informacjami adresowymi; uzbrojeni w tę informację mogą już nawiązać bezpośrednią komunikację. Tak właśnie funkcjonuje m.in. popularny Skype.

Proces „wybijania dziury" zilustrowany został na rysunku 7.7. Klient **A** kontaktuje się z serwerem **S1**, po chwili robi to klient **B**. Dzięki wspomnianemu serwerowi klient **A** uzyskuje publiczny adres IP klienta **B** (`203.0.113.100`), a klient **B** uzyskuje publiczny adres IP klienta **A** (`192.0.2.201`). Teoretycznie obaj klienci mogą już nawiązać połączenie, nie zapominajmy jednak, że obaj znajdują się po „wewnętrznej" stronie swych urządzeń NAT: klient **A** oddzielony jest od świata zewnętrznego przez urządzenie **N1** utrzymujące

odwzorowanie (**A, S1**), zaś klient B za dwoma takimi urządzeniami — **N2** i **N3** — utrzymującymi odwzorowanie (**B, S1**). Jeżeli funkcje firewalla w którymś z urządzeń **N1**, **N2** i **N3** należeć będą do kategorii innej niż „niezależne od punktu docelowego", to pakiety przychodzące ze źródła innego niż serwer **S1** będą konsekwentnie odrzucane, więc do **B** nie będą docierać pakiety wysyłane przez **A** i *vice versa* — bezpośrednie połączenie między **A** i **B** nie będzie funkcjonować.



Rysunek 7.7. *Aplikacje oddzielone od siebie barierami NAT mogą wymagać pomocy zewnętrznego serwera w bezpośrednim skomunikowaniu się. W technice „wybijania dziury" serwer, często wyspecjalizowany pod kątem określonej aplikacji, dostarcza obu zainteresowanym aplikacjom informacji kontaktowych. Niektóre z aplikacji próbują „zafiksować" (czyli rozpoznać i utrzymywać) zewnętrzny adres i port NAT, co w pewnych sytuacjach może być posunięciem chybionym: na rysunku z perspektywy serwera S1 zewnętrznym adresem klienta A jest 192.0.2.201, a adresem zewnętrznym klienta B — 203.0.113.100. Z perspektywy serwera S2 adresem zewnętrznym B jest jednak 10.0.1.1*

7.4.2. Jednostronne fiksowanie adresów (UNSAF)

Aplikacje wykorzystują różne metody do uzyskania informacji na temat globalnych adresów, na jakie odwzorowywane będą ich lokalne adresy po przejściu przez NAT; rozpoznawanie tych adresów i zarządzanie nimi, zwane w oryginale *fixing*, może być realizowane za pomocą metod pośrednich i bezpośrednich. Metody pośrednie opierają się na obserwowaniu zachowania NAT wobec ruchu generowanego przez aplikację, istotą metod bezpośrednich jest natomiast bezpośrednia konwersacja aplikacji z mechanizmem NAT, przy użyciu specjalnych protokołów (które obecnie nie są standardami uznawanymi przez IETF). Dotychczas wysiłki IETF koncentrowały się raczej na opracowywaniu metod pośrednich, szeroko obsługiwanych przez pewne grupy aplikacji, z najpopularniejszymi aplikacjami VoIP na czele. Niektóre z obecnych NAT obsługują metody bezpośrednie; ponieważ są one wykorzystywane do celów podstawowego konfigurowania NAT, omówimy je później, właśnie w kontekście ustawień i konfiguracji.

Aplikacja próbująca „zafiksować" swój adres bez pomocy ze strony NAT czyni to w tzw. sposób jednostronny (*unilateral fashion*), co określane jest skrótem UNSAF (*UNilateral Self-Address Fixing*, patrz [RFC3424]). Jak sugerować może nazwa, metody takie mają raczej charakter prowizoryczny, traktowane są na zasadzie zabiegów doraźnych (*fixing*),

„mniejszego zła" i absolutnie nieprzeznaczone do stosowania w dłuższej perspektywie. Opierają się one na różnych heurystykach, niegwarantujących sukcesu, choćby z tego względu, że implementacje NAT różnych producentów mogą się znacząco różnić w szczegółach. Dokumenty publikowane przez grupę BEHAVE, zmierzające do uściślenia wielu z tych szczegółów, mają sprawić, że metody UNSAF staną sie bardziej niezawodne.

Większość metod UNSAF wykorzystuje technologię klient-serwer w sposób podobny do „wybijania dziury", lecz bardziej ogólny. Spoglądając na rysunek 7.7, możemy zauważyć kilka związanych z tym pułapek — najważniejsza z nich wynika z faktu, iż każde z trzech urządzeń NAT operuje w innych zewnętrznych domenach adresowych. W przypadku klienta **B** kontakt ze światem zewnętrznym przechodzi przez dwa urządzenia NAT, więc odpowiedź uzyskana w ramach metody UNSAF będzie różna, zależnie od tego, czy użyty zostanie serwer **S1**, czy **S2**. Ponadto, ponieważ UNSAF wykorzystuje serwery niezależne od NAT, zawsze istnieje możliwość, że raportowane przez te serwery wyniki (oparte na zachowaniu NAT) zmieniać się będą w czasie i „zafiksowane" adresy utracą swą aktualność. Wobec licznych problemów związanych z UNSAF — i generalnie z NAT — działająca w ramach IETF grupa ekspertów IAB określiła wymagania, jakie spełniać muszą propozycje protokołów związanych z UNSAF. Każda taka propozycja musi:

1. definiować ograniczony problem, którego rozwiązaniu jest dedykowana,
2. definiować strategię awaryjną i plan alternatywny,
3. wskazywać i uzasadniać te decyzje projektowe, które sprawiają, że jest atrakcyjna,
4. identyfikować wymagania w zakresie odpowiednich rozwiązań długoterminowych,
5. uwzględniać zdobyte doświadczenia i rozpoznane problemy.

Ta cokolwiek niezwykła lista wymagań narzuconych na specyfikacje protokołów stanowi w rzeczywistości rezultat długofalowych problemów ze współdziałaniem różnych NAT i rozmaitych technik ich omijania. Niezależnie od wszystkich problemów metody UNSAF są powszechnie wykorzystywane, ponieważ wiele spotykanych w praktyce implementacji NAT zapewnia wystarczający do tego stopień spójności. Pokażemy teraz, jak metody te można wykorzystać do zbudowania solidnych, uniwersalnych technik omijania NAT, maksymalizujących szanse powodzenia komunikacji między systemami oddzielonymi od Internetu przez NAT (być może wielopoziomowo) w konfiguracjach, takich jak przedstawiona na rysunku 7.7.

7.4.3. Omijanie NAT za pomocą STUN

Głównym zastosowaniem UNSAF — i jednocześnie najpopularniejszą techniką omijania NAT — jest mechanizm *Session Traversal Utilities for NAT*, znany pod akronimem STUN i opisywany w [RFC5389]. Obecna postać STUN to wynik ewolucji oryginału o nazwie *Simple Tunneling of UDP through NATs*, obecnie określanego mianem „klasycznego STUN". Klasyczny STUN wykorzystywany był przez pewien czas na potrzeby aplikacji VoIP/SIP, a następnie został rozbudowany do postaci narzędzia wykorzystywanego przez inne protokoły omijania NAT. W aplikacjach wymagających kompletnych rozwiązań zalecane jest jednak stosowanie bardziej zaawansowanych, kompletnych frameworków w rodzaju ICE (opisywanego w punkcie 7.4.5) w połączeniu z *SIP Outbound*, wykorzystujących

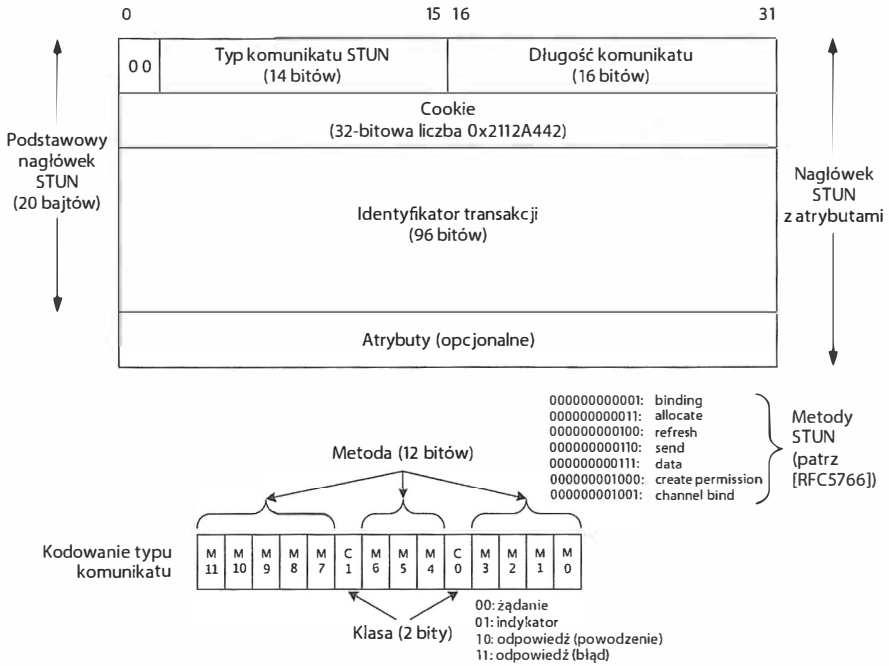
STUN na kilka sposobów, zwanych *użyciami* (*usages*). Użycia rozszerzają podstawowe STUN o dodatkowe operacje, typy komunikatów i kody błędów, zdefiniowane w dokumencie [RFC5389].

STUN jest stosunkowo prostym protokołem typu klient-serwer, umożliwiającym uzyskiwanie zewnętrznego adresu IP i zewnętrznego numeru portu; realizuje także podtrzymywanie sesji NAT poprzez wysyłanie komunikatów *keepalive*. Do prawidłowego działania wymaga współpracującego serwera po „drugiej" stronie NAT i ewentualnie kilku publicznych serwerów STUN osiągalnych przez Internet pod globalnymi adresami IP. Podstawowym zadaniem serwerów STUN jest „odbijanie" otrzymywanych żądań, co umożliwia klientowi uzyskanie niezbędnej informacji adresowej. Podobnie jak metody UNSAF w ogólności, także i STUN nie jest niezawodny. Nie umniejsza to jednak jego atrakcyjności, której główną przesłanką jest brak konieczności modyfikowania routerów sieciowych, protokołów aplikacyjnych czy serwerów: wystarczy zaimplementowany po stronie klienta protokół żądań STUN i przynajmniej jeden serwer STUN dostępny w znanej lokalizacji. Z założenia STUN pomyślany został jako rozwiązanie tymczasowe, do czasu opracowania i zaimplementowania bardziej wymyślnych protokołów, a docelowo — do czasu, aż szerokie rozpowszechnienie IPv6 odeśle do lamusa całą technologię NAT.

STUN wykorzystuje do swego działania protokoły UDP, TCP oraz TLS (patrz rozdział 18.); specyfikacja użycia określa także numery portów: 3478 dla UDP i TCP oraz 3479 dla TCP w połączeniu z TLS. Podstawowy protokół STUN definiuje dwa rodzaje transakcji: żądanie/odpowiedź (*request/response*) oraz indykator (*indication*). Indykatory nie wymagają odpowiedzi i mogą być generowane zarówno przez klienty, jak i przez serwery. Każdy komunikat STUN zawiera określenie typu i długości, *cookie* `0x2112A442` oraz losowy 96-bitowy identyfikator transakcji, umożliwiający kojarzenie żądań z odpowiedziami oraz wykorzystywany do celów debugowania. Każdy komunikat rozpoczyna się dwoma bitami zerowymi i opcjonalnie może zawierać *atrybuty*. Typ komunikatu definiowany jest w kontekście *metody* związanej z określonym użyciem STUN; parametry STUN, obejmujące m.in. numery metod i atrybutów, definiowane są centralnie przez IANA (patrz [ISP]). Atrybuty mają swe własne typy i mogą być zmiennej długości. Podstawowy nagłówek STUN, najczęściej lokowany bezpośrednio za nagłówkiem transportowym UDP w datagramie IP, przedstawiony jest na rysunku 7.8.

Podstawowy nagłówek STUN ma rozmiar 20 bajtów, zaś całkowita długość komunikatu w bajtach (bez uwzględnienia nagłówka) zapisana jest w polu *Długość komunikatu*. 16-bitowy rozmiar pola ogranicza długość komunikatu do $2^{16}-1 = 65\,535$ bajtów, jednakże komunikaty zawsze wyrównywane są do wielokrotności 4 bajtów, wskutek czego limit ten zmniejsza się do 65 532 bajtów. Przy przesyłaniu komunikatów STUN na bazie pakietów UDP/IP datagramy IP formowane są w rozmiarze nie większym niż wartość MTU dla ścieżki (PMTU), a jeżeli wartość ta nie jest znana — w rozmiarze nieprzekraczającym 576 bajtów (dla IPv4) i 1280 bajtów (dla IPv6); ma to zapobiec fragmentowaniu datagramów IP. STUN nie zawiera jednocześnie żadnego zabezpieczenia na wypadek, gdyby rozmiar datagramów generowanych przez serwer okazał się większy niż wartość PMTU; odpowiedzialność za limitowanie tego rozmiaru spoczywa więc na serwerze.

Ponieważ przenoszenie komunikatów STUN za pomocą pakietów UDP/IP nie jest niezawodne (bo nie jest niezawodny sam protokół UDP), aplikacje STUN muszą we własnym zakresie zapewnić wymaganą niezawodność, realizowaną poprzez ponowne wysyłanie



Rysunek 7.8. *Komunikat STUN rozpoczyna się od dwóch zerowych bitów i zwykle enkapsulowany jest w pakiecie UDP, choć możliwe jest jego enkapsulowanie także w segmentach TCP. Typ komunikatu identyfikuje zarówno metodę, jak i klasę. Identyfikator transakcji jest losową, 96-bitową liczbą umożliwiającą kojarzenie żądań z odpowiedziami lub wykorzystywaną do celów debugowania w przypadku indykatorów. Opcjonalnie komunikat STUN zawierać może atrybuty, zależne od konkretnego użycia*

komunikatów, co do których istnieje podejrzenie, że zostały utracone. Interwał czasowy retransmisji ustalany jest w oparciu o szacunkowy czas wędrówki pakietu od klienta do serwera i z powrotem, nazywany *czasem wędrówki* (*round-trip time*, w skrócie RTT). Obliczanie RTT i odpowiednie ustawianie stoperów retransmisji jest istotnym zagadnieniem w przypadku protokołu TCP (o czym pisać będziemy w rozdziale 14.); STUN wykorzystuje podobną technikę, jednak przy użyciu innych parametrów niż domyślne dla TCP — czytelników zainteresowanych szczegółami odsyłamy do dokumentu [RFC5389]. W przypadku przenoszenia komunikatów STUN w pakietach TCP/IP (lub TCP+TLS/IP) niezawodność gwarantowana jest przez sam protokół TCP — możliwa jest równoległa realizacja kilku transakcji STUN, w ramach oddzielnych połączeń TCP.

Atrybuty STUN kodowane są w konwencji TLV (*Type-Length-Value* — typ, długość, wartość) używanej także przez wiele innych protokołów internetowych. Pierwsze dwa bajty określają typ atrybutu, na dwóch następnych kodowana jest długość danych, potem już występują rzeczone dane. Dane dopełniane są do wielokrotności 4 bajtów (dopełniające bajty mają przypadkową zawartość), więc ich długość może wynosić maksymalnie 65 532 bajty). W ramach jednego komunikatu STUN może wystąpić kilka atrybutów tego samego typu, jednakże tylko dla pierwszego wystąpienia gwarantowane jest przetworzenie przez odbiorcę.

Atrybuty o typie mniejszym niż 0x8000 nazywane są *obowiązkowo interpretowalnymi* (*comprehension-required*): jeśli agent STUN odbierający komunikat nie potrafi zinterpretować takiego atrybutu, musi zasygnalizować błąd. Niemożność zinterpretowania atrybutu innego typu (0x8000 lub większego, zwanego *atrybutem opcjonalnie interpretowalnym — comprehension-optional*) może skutkować zignorowaniem tego atrybutu bez ostrzeżenia. Większość obecnie zdefiniowanych atrybutów należy do pierwszej z wymienionych kategorii (patrz [ISP]).

Specyfikacja [RFC5389] definiuje obecnie tylko jedną metodę o nazwie *binding* (wiązanie), wykorzystywaną w obu typach transakcji (żądaniach/odpowiedziach i indykatorach). Definiuje ona także 11 atrybutów, opisanych w tabeli 7.2.

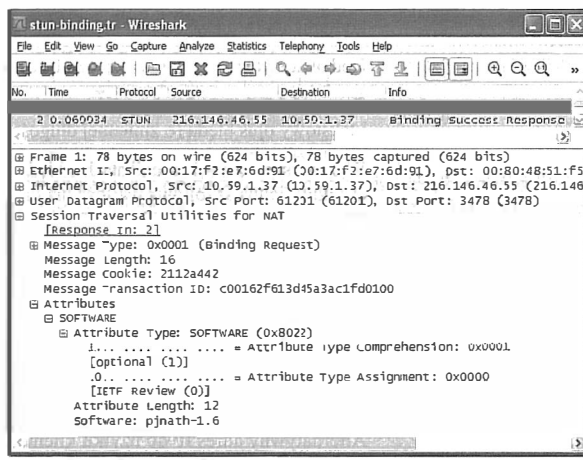
Tabela 7.2. *Atrybuty protokołu STUN, definiowanego w [RFC5389], nazywanego niekiedy STUN2 i stanowiącego zastępnik oryginalnego STUN*

Nazwa	Typ	Znaczenie
MAPPED-ADDRESS	0x0001	Wskazanie rodziny adresów oraz zwrotnego adresu transportowego (IPv4 lub IPv6)
USERNAME	0x0006	Nazwa i hasło użytkownika, wykorzystywane do kontroli integralności komunikatu (długość do 513 bajtów)
MESSAGE-INTEGRITY	0x0008	Kod uwierzytelniania komunikatu STUN (patrz [RFC5389] i rozdział 18.)
ERROR-CODE	0x0009	Znacznik błędu, zawierający 3-bitową klasę błędu, 8-bitowy kod błędu i tekstowy opis błędu (o zmiennej długości)
UNKNOWN-ATTRIBUTES	0x000A	Wykorzystywany w komunikatach o błędach do wskazania nierozpoznanego atrybutu (jedna 16-bitowa wartość dla każdego atrybutu)
REALM	0x0014	Wskazuje nazwę dziedziny uwierzytelniania dla długoterminowego zaufania
NONCE	0x0015	Unikatowa wartość, opcjonalnie włączana do komunikatów żądań i odpowiedzi, w celu zapobiegania atakom powtarzania (patrz rozdział 18.)
XOR-MAPPED-ADDRESS	0x0020	Zakodowana (XOR) wersja atrybutu MAPPED-ADDRESS
SOFTWARE	0x8022	Tekstowy opis oprogramowania wysyłającego komunikat (np. nazwa producenta i numer wersji)
ALTERNATE-SERVER	0x8023	Alternatywny adres IP do użytku klienta, zakodowany tak samo jak MAPPED-ADDRESS
FINGERPRINT	0x8028	Kod CRC-32 dla komunikatu zsumowanego symetrycznie (XOR) z ciągiem wartości 0x5354554E. Atrybut opcjonalny; jeśli jest używany, musi wystąpić jako ostatni

Powróćmy do rysunku 7.5: klient STUN o adresie $X : x$ zainteresowany jest uzyskaniem informacji $X_1' : x_1'$, zwanej *zwrotnym adresem transportowym* (*reflexive transport address*) lub *adresem mapowanym* (*mapped address*). Serwer STUN ($Y_1 : y_1$) umieszcza ten adres zwrotny w ramach atrybutu MAPPED-ADDRESS w komunikacie wysyłanym do klienta. Dokładniej, informacja adresowa zawiera 8-bitowy kod *rodziny adresowej* (0x01 dla IPv4, 0x02 dla IPv6), 16-bitowy numer portu oraz adres IP, 32-bitowy albo 128-bitowy, zależnie od wskazanej rodziny adresowej. Atrybut ten został zachowany dla kompatybilności z klasycznym STUN. Ważniejszym atrybutem jest natomiast XOR-MAPPED-ADDRESS,

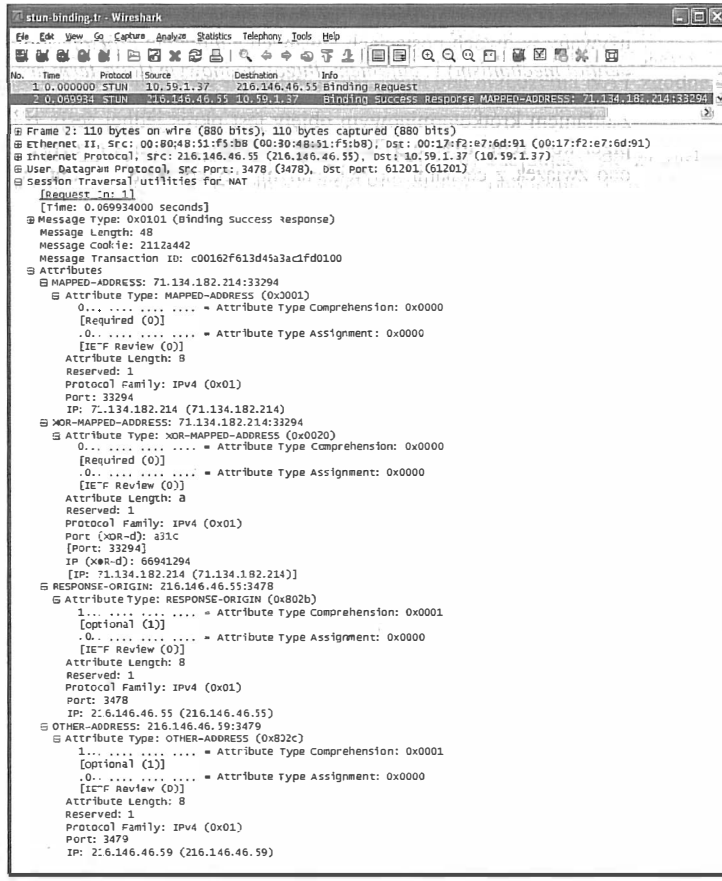
który zawiera tę samą informację adresową, co MAPPED-ADDRESS, lecz po zsumowaniu symetrycznym (XOR) z ciągiem szyfrującym, którym w wersji IPv4 jest wspomniane wcześniej cookie, zaś w wersji IPv6 konkatenacja tego cookie i 96-bitowego identyfikatora transakcji. Celem takiego sumowania jest ochrona przed ingerencją ze strony niektórych bram aplikacyjnych (ALG), które odnajdują w pakiecie wszelkie adresy IP i niektóre z nich nadpisują, a tym samym niszczyć mogą informacje istotne dla STUN. Jak pokazuje doświadczenie, opisane szyfrowanie adresów IP stanowi wystarczające zabezpieczenie przed nadgorliwością ALG.

Klient STUN, najczęściej urządzenie VoIP z aplikacją „softphone"(np. `pjsua` — patrz [PJSUA]), jest początkowo zaopatrzony w adres (adresy) IP lub nazwę (nazwy) pobrane z jednego lub kilku serwerów STUN. Pożądane jest, by serwery te prawdopodobnie „widziały" te same adresy, co partner, z którym aplikacja chce się skontaktować, choć naprawdę trudno to sprawdzić lub zrealizować; zwykle wystarczające jest skorzystanie z serwerów dostępnych publicznie w Internecie (np. pod adresem *stun.ekiga.net*, *stun.xten.com* czy *numb.viagenie.ca*). Niektóre serwery STUN można zlokalizować, używając rekordów SRV usługi DNS (piszemy o tym w rozdziale 11.). Przykład realizacji wiązania STUN w oknie programu Wireshark widoczny jest na rysunku 7.9.



Rysunek 7.9. *Żądanie STUN zawierające 96-bitowy identyfikator transakcji i atrybut SOFTWARE identyfikujący klienta formułującego żądanie. Wartość tego atrybutu jest 10-znakowa, lecz dopełniona zostaje do wielokrotności 4 bajtów, czyli do 12 bajtów; razem z 2 bajtami typu i 2 bajtami długości daje to rozmiar komunikatu 16 bajtów*

Wymiana komunikatów inicjowana jest przez klienta; identyfikatorem transakcji staje się losowo wybrana liczba pseudolosowa `0xc00162f613d45a3ac1fd0100`. Żądanie skierowane jest do serwera `numb.viagenie.ca` (z adresami IPv4 `216.146.46.55` i `216.146.46.59`), który jest jednocześnie serwerem STUN i serwerem TURN (patrz punkt 7.4.4). Żądanie zawiera atrybut SOFTWARE identyfikujący aplikację kliencką — w tym przypadku `pjnath-1.6` oznacza aplikację *PJSIP NAT helper* wchodzącą w skład `pjsua`. Długość komunikatu wynosi 16 bajtów, składają się na nią 2 bajty identyfikujące atrybut (`0x8022`), 2 bajty określające długość danych atrybutu (12) i 12 bajtów danych. Co prawda, łańcuch `pjnath-1.6` liczy tylko 10 znaków, lecz jak wspominaliśmy, rozmiar atrybutu (lub, co na jedno wychodzi, rozmiar danych) zaokrąglany jest w górę do wielokrotności 4 bajtów. Żądanie po przejściu przez NAT i dotarciu do serwera generuje odpowiedź, która po dotarciu do klienta wygląda tak, jak na rysunku 7.10.



Rysunek 7.10. *Odpowiedź STUN zawierająca cztery atrybuty: MAPPED-ADDRESS i XOR-MAPPED-ADDRESS zawierające zwrotny adres transportowy, dwa pozostałe mają znaczenie eksperymentalne (patrz [RFC5780])*

Widoczny na rysunku 7.10 pakiet odpowiedzi niesie informację użyteczną dla klienta, zakodowaną w postaci ciągu atrybutów. Atrybuty MAPPED-ADDRESS i XOR-MAPPED-ADDRESS zawierają zwrotny adres transportowy 71.134.182.214:33294 wykryty przez serwer. Atrybuty RESPONSE-ORIGIN i OTHER-ADDRESS związane są z opisanymi w [RFC5780] eksperymentalnymi funkcjami STUN. Pierwszy z nich zawiera oryginalny (przed zamapowaniem) punkt docelowy 216.146.46.55:3478 umieszczany przez klienta w pakiecie, drugi zawiera alternatywny punkt docelowy (216.146.45.59:3479), udostępniany klientowi w sytuacji, gdy ten zażąda zmiany docelowego adresu IP i (lub) zmiany portu docelowego; atrybut OTHER-ADDRESS jest odpowiednikiem dawnego atrybutu CHANGED-ADDRESS w klasycznym STUN.

STUN oferuje nie tylko *fixing* adresów, lecz także kilka innych funkcji, zwanych oficjalnie *mechanizmami*. Mechanizmy uruchamiane są w kontekście konkretnego użycia STUN, generalnie więc postrzegane są jako cechy opcjonalne. Wśród najważniejszych mechanizmów STUN należy wymienić wykrywanie DNS, przekierowywanie do alternatywnego

serwera oraz uwierzytelnianie i kontrolę integralności komunikatów. Ten ostatni może być używany w dwóch postaciach, zwanych *zaufaniem krótkoterminowym* (*short-term credential mechanism*) i *zaufaniem długoterminowym* (*long-term credential mechanism*).

Zaufaniem krótkoterminowym objęty jest okres jednej sesji, o długości definiowanej przez konkretne użycie STUN; zaufanie długoterminowe rozciąga się na wiele sesji, ma ono związek z cechami indywidualnymi użytkownika — kontem, loginem, identyfikatorem itd. Zaufanie krótkoterminowe wykorzystywane jest najczęściej w kontekście konkretnej wymiany komunikatów, natomiast na podstawie zaufania długoterminowego przydzielane są zwykle konkretne zasoby (np. w ramach TURN, które opisujemy w punkcie 7.4.4). Hasła nigdy nie są przesyłane w jawnej postaci, gdyż wtedy mogłyby zostać przechwycone i skompromitowane.

Mechanizm zaufania krótkoterminowego wykorzystuje atrybuty `USERNAME` i `MESSAGE-INTEGRITY`, oba obowiązkowe w każdym żądaniu. Pierwszy z nich identyfikuje użytkownika i jego cechy zaufania, pośrednio prowadzi także do hasła stanowiącego klucz do obliczania kodu uwierzytelnienia komunikatu (MAC — *Message Authentication Code*, patrz rozdział 18.) Przy zaufaniu krótkoterminowym zakłada się, że niezbędne informacje (m.in. nazwa i hasło użytkownika) zostały wcześniej dostarczone. Wspomniany kod uwierzytelnienia komunikatu stanowi treść atrybutu `MESSAGE-INTEGRITY`; poprawność tego kodu świadczy o tym, że użytkownik posługuje się aktualnymi danymi uwierzytelniającymi.

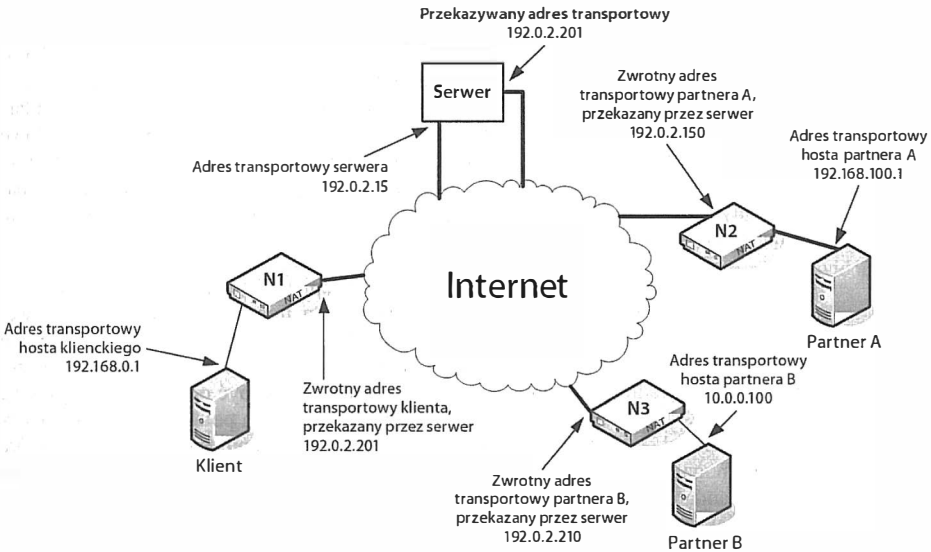
W przypadku zaufania długoterminowego weryfikację aktualności danych uwierzytelniających wykonuje się w inny sposób, za pomocą mechanizmu „wyzwania przez skrót" (*digest challenge*). Klient początkowo formułuje żądanie bez podawania danych uwierzytelniających; serwer żądanie to odrzuca, załączając jednak w odpowiedzi atrybut `REALM`. Zawarta w nim informacja może być wykorzystana przez klienta do określenia, których danych uwierzytelniających należy konkretnie użyć — klient może w różny sposób uwierzytelniać się wobec różnych usług, np. wobec różnych kot VoIP. W ramach atrybutu `REALM` klient otrzymuje także unikatową, nieużywaną wcześniej, wartość *nonce*, wykorzystywaną w późniejszych żądaniach. Mechanizm „wyzwania przez skrót" wykorzystuje tę wartość jako klucz dla obliczenia kodu MAC (przesyłanego pod postacią atrybutu `MESSAGE-INTEGRITY`). Uniemożliwia to ewentualnemu intruzowi wykonanie ataku powtarzania, czyli ponownego przesłania przechwyconego wcześniej zaufanego komunikatu — w nim bowiem użyta została inna wartość *nonce* niż obecnie spodziewana przez serwer (używanie wartości *nonce* do uwierzytelniania i innych podobnych celów omawiamy szczegółowo w rozdziale 18.). Zauważmy, że mechanizm zaufania długoterminowego nie może być używany do ochrony transakcji wskazań STUN, ponieważ transakcje te nie mają formy dialogu „żądanie-odpowiedź".

7.4.4. Omijanie NAT z użyciem przekaźników (TURN)

Gdy zawodzą wszystkie inne metody — bo komunikujący się partnerzy znajdują się za zasłonami niewspółpracujących ze sobą urządzeń NAT — ostatnią szansą na ich skomunikowanie jest wykorzystanie zewnętrznego serwera wykonującego wahadłowe przerzucanie (*shuttle*) danych między dwoma systemami. Metoda ta, definiowana w dokumencie [RFC5766] i określona akronimem TURN (*Traversal Using Relays around NAT* — „obchodzenie NAT przy użyciu przekaźnika"), powstała jako rozszerzenie STUN o specyficzne elementy funkcjonalne i komunikaty. Wymaga jedynie możliwości połączenia

każdego z partnerów ze wspomnianym serwerem, znajdującym się „na zewnątrz" obu urządzeń NAT. Ze względu na niską wydajność technologia ta traktowana jest jako ostateczność i (prawdopodobnie) staje się niepotrzebna, jeżeli oba urządzenia NAT zgodne są ze specyfikacjami grupy BEHAVE.

Zgodnie z rysunkiem 7.11, klient (z lewej strony rysunku) znajdujący się za zasłoną NAT **N1**, kontaktuje się z serwerem (zwykle dostępnym publicznie w Internecie), wskazując mu partnerów (*peers*), z którymi zamierza się komunikować. Lokalizacja tego serwera może być ustalona manualnie, możliwe jest też jego automatyczne odnajdywanie przy użyciu rekordów NAPTR usługi DNS (piszemy o tym w rozdziale 11., na podstawie dokumentu [RFC5928]). W odpowiedzi na żądanie klienta serwer TURN udostępnia mu adresy transportowe partnerów (czyli pary „adres IP-port"), zwane *przekazanymi adresami transportowymi* (*relayed transport address*); klient otrzymuje również swój własny zwrotny adres transportowy, zakłada się także, iż partnerzy również znają swoje zwrotne adresy transportowe. Sama metoda przekazywania adresów nie jest zdefiniowana na gruncie TURN, bo wiąże się z użyciem innych mechanizmów, np. ICE (opisywanego w punkcie 7.4.5).



Rysunek 7.11. *(na podstawie RFC5766]). Serwer TURN pomaga klientowi ukrytemu za problematycznym NAT w komunikacji z innym klientem, za pośrednictwem przekaźnika. Ruch między klientem i serwerem wykorzystywać może protokoły TCP, UDP lub TCP+TLS, zaś ruch między serwerem a partnerami klienta opiera się na pakietach UDP. Przekazywanie ruchu jest techniką nieefektywną, używaną tylko wtedy, gdy zawodzą metody bezpośredniej komunikacji*

Za pomocą odpowiednich poleceń TURN klient na serwerze pośredniczącym tworzy i utrzymuje *alokacje*, reprezentujące poszczególnych potencjalnych partnerów, za pomocą unikatowych, przekazanych przez serwer, ich adresów transportowych. Dane pomiędzy serwerem i partnerami przesyłane są w ramach prostych komunikatów TURN tradycyjnie enkapsulowanych w pakietach UDP/IPv4, choć zdefiniowano rozszerzenia uwzględniające

udział TCP ([RFC6062]) i IPv6, włącznie z przekazywaniem danych między IPv4 a IPv6 ([RFC6156]). Informacja wymieniana między klientem a serwerem enkapsulowana jest w indykatorach TURN, identyfikujących partnerów, którym należy dostarczyć (lub od których należy pobrać) dane. Transakcje między klientem a serwerem mogą być wykonywane przy użyciu protokołów UDP/IP, TCP/IP lub TCP+TLS/IP. Utworzenie nowej alokacji wymaga uwierzytelnienia klienta, co zwykle odbywa się w ramach mechanizmu długoterminowego zaufania STUN.

Kopiowanie danych między klientem a jego partnerami może odbywać się w dwojaki sposób. Pierwszy z nich, opisany w sekcji 10. dokumentu [RFC5766] wykorzystuje indykatory *Send* i *Data*, niepodlegające uwierzytelnianiu. Sposób drugi, opisany w sekcji 11. wspomnianego dokumentu, opiera się na koncepcji *kanałów* (*channels*) jako ścieżek komunikacji między klientem a jego partnerami. Komunikaty przenoszone przez kanały zawierają mniejszy, bo 4-bajtowy nagłówek i jako takie nie są zgodne ze standardowym formatem komunikatów STUN wykorzystywanych przez TURN; mniejszy nagłówek oznacza jednak mniejszą ilość transmitowanych danych i mniejsze opóźnienie, co jest szczególnie istotne z perspektywy technologii uwarunkowanych czasowo, takich jak np. VoIP. Dla jednej alokacji możliwe jest utworzenie do 16535 kanałów.

Klient zwraca się do serwera z żądaniem utworzenia alokacji, specyfikując metodę `Allocate`. Pomyślne zrealizowanie żądania kwitowane jest przez serwer odpowiednim indykatorem; żądanie może jednak zostać odrzucone przez serwer, jeśli klient nie dostarczy właściwych danych uwierzytelniających. Czas życia utworzonej alokacji wynosi standardowo 10 minut i może być przedłużany za pomocą metody `Refresh`. Owo standardowe 10 minut może zostać zmienione przez klienta za pomocą atrybutu `LIFETIME` STUN, wysyłanego w żądaniu; w szczególności określenie zerowego czasu życia dla alokacji oznacza jej natychmiastowe skasowanie. Wygaśnięcie alokacji lub jej skasowanie powoduje automatyczne zwolnienie skojarzonych z nią kanałów.

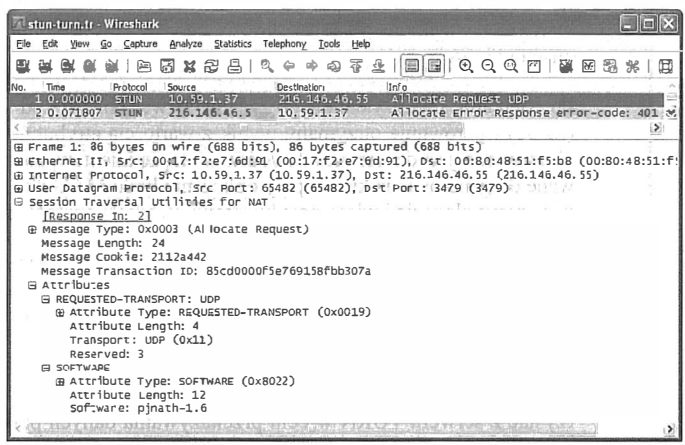
Fizycznie alokacja reprezentowana jest w postaci „piątki" (5-tuple), zawierającej po stronie klienta: transportowy adres IP oraz port jego hosta, transportowy adres i port serwera oraz kod protokołu transportowego używanego do komunikacji z serwerem. Po stronie serwera jest podobnie, jednak zamiast adresu transportowego i portu klienta zapisany jest jego zwrotny adres transportowy i port. Z alokacją mogą być opcjonalnie skojarzone *pozwolenia* (*permissions*) ograniczające wzorce dozwolonych połączeń z udziałem serwera TURN. Dokładniej, każde pozwolenie specyfikuje ograniczenia adresów IP, które wystąpić mogą w roli adresów źródłowych w pakietach otrzymywanych przez serwer TURN. Standardowy czas życia pozwolenia wynosi 5 minut i może być odświeżany, podobnie jak sama alokacja.

TURN rozszerza STUN o nowe elementy: sześć metod, dziewięć atrybutów i sześć kodów błędów, które podzielić można z grubsza na trzy grupy związane (odpowiednio) z zarządzaniem alokacjami, uwierzytelnianiem i manipulowaniem kanałami. Sześć wspomnianych metod to `Allocate` (kod 3), `Refresh` (4), `Send` (6), `Data` (7), `CreatePermission` (8) i `ChannelBind` (9). Dwie pierwsze odpowiedzialne są za tworzenie alokacji i utrzymywanie ich istnienia, dwie następne wiążą się z wymianą danych między serwerem a klientami; metoda `CreatePermission` służy definiowania i odświeżania pozwoleń, natomiast metoda `ChannelBind` wiąże alokację z kanałem identyfikowanym za pomocą 16-bitowego kodu. Sygnalizowane błędy związane są z problemami w funkcjonowaniu TURN, np. z niepomyślnym uwierzytelnianiem czy wyczerpaniem zasobów (np. numerów kanałów). Znaczenie dziewięciu atrybutów TURN wyjaśnione jest w tabeli 7.3.

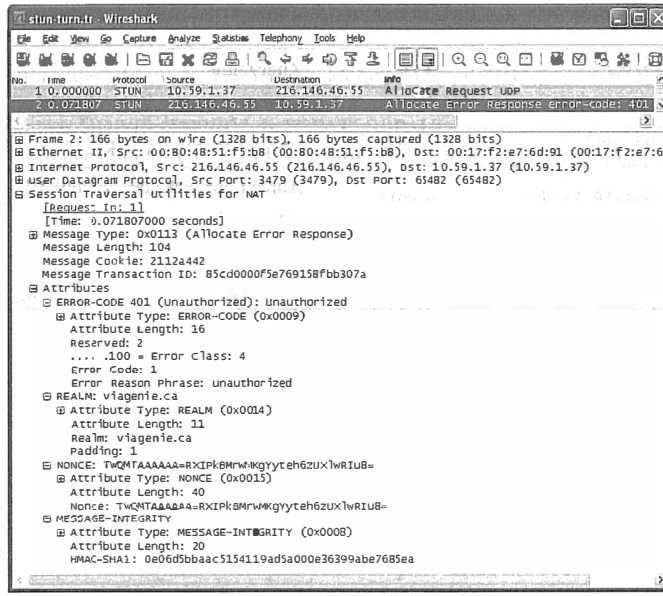
Tabela 7.3. *Nowe atrybuty STUN definiowane przez TURN*

Nazwa	Typ	Znaczenie
CHANNEL-NUMBER	0x000C	Wskazuje kanał, do którego należą odnośne dane
LIFETIME	0x000D	Określa czas życia alokacji bez odświeżania (w sekundach)
XOR-PEER-ADDRESS	0x0012	Adres i port partnera, zakodowany z użyciem sumowania symetrycznego (XOR)
DATA	0x0013	Zawiera dane wymieniane w ramach indykatorów Send i Data
XOR-RELAYED-ADDRESS	0x0016	Adres i port serwera przydzielony klientowi
EVEN-PORT	0x0018	Żądanie przydzielenia parzystego portu (opcjonalnie — pary sąsiednich portów, z których pierwszy jest parzysty) dla zwrotnego adresu transportowego
REQUESTED-TRANSPORT	0x0019	Żądanie określonego protokołu transportowego, w konwencji takiej samej jak w polu *Protokół* nagłówka IPv4 lub polu *Następny nagłówek* nagłówka IPv6
DONT-FRAGMENT	0x001A	Żądanie ustawienia bitu zakazującego fragmentowania w nagłówku IPv4 datagramów przesyłanych między partnerami
RESERVATION-TOKEN	0x0022	Unikatowy identyfikator skojarzony z przekazywanym adresem transportowym, utrzymywany przez serwer i przekazywany klientowi w celach referencyjnych

Żądanie TURN ma postać komunikatu STUN o typie 0x0003 oznaczającym żądanie utworzenia alokacji — co widać na rysunku 7.12. Zgodnie z mechanizmem zaufania długoterminowego, żądanie to pozbawione jest danych uwierzytelniających, więc zostaje przez serwer odrzucone, o czym klient poinformowany zostaje za pomocą komunikatu widocznego na rysunku 7.13.



Rysunek 7.12. *Żądanie TURN utworzenia alokacji (typ 0x0003), zawierające atrybuty REQUESTED-TRANSPORT i SOFTWARE. Nie zawiera ono danych uwierzytelniających i zgodnie z oczekiwaniami zostanie odrzucone*



Rysunek 7.13. *Odpowiedź TURN sygnalizująca odrzucenie żądania (atrybut ERROR-CODE zawiera wartość 401 — brak autoryzacji). Atrybuty REALM i NONCE zawierają informację, którą klient wykorzystuje do sformułowania nowego, tym razem uwierzytelnionego żądania. Atrybut MESSAGE-INTEGRITY związany jest z ochroną integralności komunikatu*

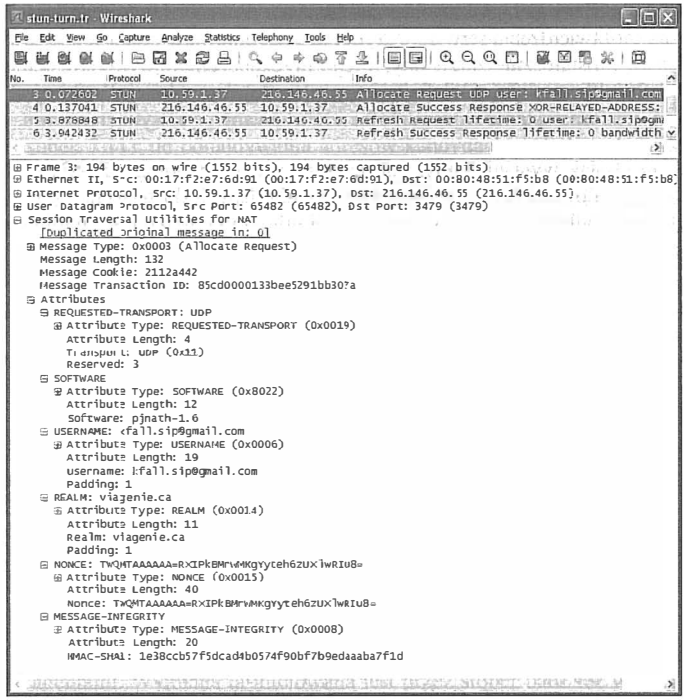
Komunikat sygnalizujący odmowę utworzenia alokacji zawiera atrybut `REALM` o wartości `viagenie.ca`, atrybut `NONCE` z unikatową wartością, którą klient powinien wykorzystać w następnym żądaniu, a także atrybut `MESSAGE-INTEGRITY` stanowiący dla klienta zapewnienie, że komunikat nie został zmodyfikowany i wartości atrybutów `REALM` oraz `NONCE` są autentyczne.

Kolejne żądanie klienta (widoczne na rysunku 7.14) zawiera już dane uwierzytelniające jako treść atrybutu `USERNAME`, w ramach atrybutu `NONCE` przekazywana jest unikatowa wartość udostępniona przez serwer, zaś autentyczność komunikatu poświadczona jest przez atrybut `MESSAGE-INTEGRITY`.

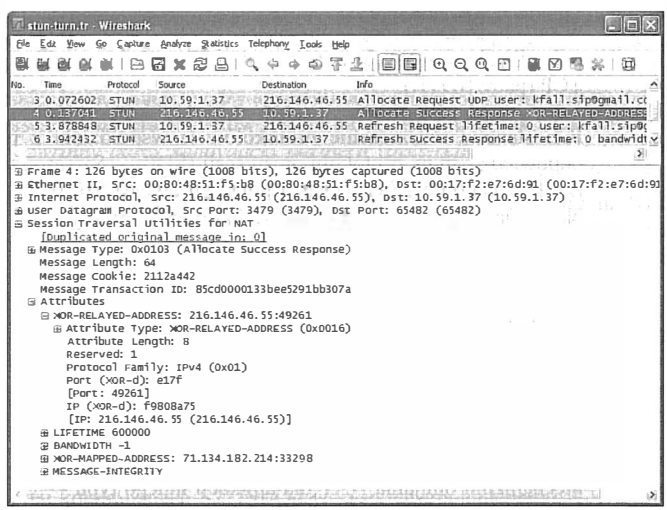
Po odebraniu uwierzytelnionego żądania serwer oblicza kod uwierzytelnienia komunikatu (MAC) i porównuje otrzymany wynik z wartością atrybutu `MESSAGE-INTEGRITY`. Równość obu wartości jest dla serwera dowodem na to, że klientowi znane jest wymagane hasło (używane do generowania klucza przy obliczaniu kodu MAC dla komunikatu). Serwer tworzy więc nową alokację i udostępnia klientowi wynik tej operacji, tak jak na rysunku 7.15.

Zwrócony kod typu komunikatu `0x0103` oznacza powodzenie tworzenia alokacji, zaś przydatne w związku z tym klientowi informacje zawarte są w szeregu atrybutów. Na rysunku 7.15 widoczny jest w postaci rozwiniętej atrybut `XOR-RELAYED-ADDRESS`, zawierający informację o przekazanym adresie transportowym `216.146.46.55:49261` — zauważmy, że program Wireshark wykonał jego ekstrakcję z postaci zakodowanej przez sumowanie symetryczne (XOR) do postaci czytelnej dla użytkownika. Od tej chwili klient może korzystać z serwera TURN w celu przekazywania danych partnerom i odbierania danych od partnerów. Po zakończeniu tej wymiany alokacja staje się niepotrzebna i można ją zwolnić — temu celowi służy komunikat 5., z zerową wartością atrybutu `LIFETIME`, komunikat 6. jest potwierdzeniem pomyślnego usunięcia.

Rysunek 7.14. *Powtórne żądanie TRUN utworzenia alokacji, zawierające atrybuty USERNAME, REALM, NONCE i MESSAGE-INTEGRITY. Ostatni z nich wykorzystywany jest przez serwer do weryfikacji tożsamości klienta; pomyślny wynik tej weryfikacji powoduje zrealizowanie żądania, czyli utworzenie alokacji*



Rysunek 7.15. *Odpowiedź TURN świadcząca o pomyślnym utworzeniu alokacji. Komunikat zawiera atrybut XOR-RELAYED-ADDRESS, którego treścią jest adres IP i port (w postaci zakodowanej) przydzielony przez serwer TURN. Alokacja nieodświeżana przestanie istnieć po upływie czasu określonego przez parametr LIFETIME*



Zauważmy także obecność atrybutu BANDWIDTH w komunikatach 4. i 6. stanowiących potwierdzenia serwera. Atrybut ten, zdefiniowany został w szkicowej wersji [RFC5766], lecz ostatecznie zarzucony, a jego przeznaczeniem było określenie maksymalnej szybkości transmisji (w kilobajtach na sekundę) dozwolonej dla danej alokacji. Być może atrybut ten zostanie w przyszłości zdefiniowany w nowym znaczeniu.

Jak wyjaśnialiśmy, podstawową wadą technologii TURN jest mała wydajność spowodowana koniecznością kopiowania danych do serwera i z serwera — być może bardzo odległego — nawet w sytuacji gdy komunikujący się partnerzy zlokalizowani są blisko siebie. Ponadto do klienta nie docierają niektóre dane wysyłane przez partnerów, m.in. komunikaty ICMP (patrz rozdział 8.) oraz wartości w polach *TTL* (*Limit przeskoków*) i *DS* datagramów IP. Ponadto od klienta korzystającego z serwera TURN wymaga się implementacji uwierzytelnienia zapewniającego ustanawianie długoterminowego zaufania STUN, być może w oparciu o konto tworzone przez administratora serwera. Chroni to przed niekontrolowanym dostępem do serwera TURN, ale też przyczynia się do złożoności konfiguracji i definiowania ustawień.

7.4.5. ICE — interaktywne nawiązywanie połączenia

Wobec dużego zróżnicowania funkcjonujących w praktyce implementacji NAT, a także wielu różnych mechanizmów „omijania" NAT, konieczne stało się opracowanie generycznego mechanizmu umożliwiającego funkcjonowanie aplikacji rozproszonych rozdzielonych barierą NAT. Takim mechanizmem jest opisywany w dokumencie [RFC5245] ICE (*Interactive Connectivity Establishment* — interaktywne nawiązywanie połączenia) pomagający w komunikowaniu się aplikacji bazujących na protokole UDP, oddzielonych barierami NAT. Ogólnie rzecz biorąc, ICE to zbiór heurystyk, za pomocą których aplikacja realizuje technikę UNSAF w sposób (względnie) przewidywalny. ICE wykorzystuje w swych działaniu inne protokoły w rodzaju TURN i STUN. Niedawno (w 2011 roku) pojawiła się propozycja rozszerzenia ICE o obsługę aplikacji bazujących na protokole TCP (patrz [IDTI]).

ICE stanowi rozszerzenie protokołów (i współdziała z protokołami) typu „oferta-odpowiedź", m.in. protokołu opisu sesji (SDP — *Session Description Protocol*) wykorzystywanego do nawiązywania połączeń unicast SIP (patrz [RFC3264]). Protokoły te bazują na wymianie komunikatów, z których pierwszy stanowi ofertę konkretnej usługi, sformułowaną jako ciąg jej parametrów, drugi natomiast jest odpowiedzią precyzującą wybór pożądanych opcji i wartości wspomnianych parametrów. Technologia ICE staje się coraz bardziej popularna w aplikacjach VoIP wykorzystujących do komunikacji protokoły SIP/SDP: zadaniem ICE jest ułatwienie omijania barier NAT przez strumienie multimedialne (m.in. komponenty audio i wideo transmisji RTP i SRTP, patrz odpowiednio [RFC3550] i [RFC3711]), podczas gdy zarządzanie informacją sygnalizacyjną (np. prezentacja numeru dzwoniącego) spoczywa w gestii mechanizmu o nazwie *SIP Outbound* (patrz [RFC5626]). I choć w praktyce technologia ICE była i jest wykorzystywana głównie na potrzeby aplikacji SIP/SDP, może również pełnić rolę generycznego mechanizmu omijania NAT dla aplikacji innego typu — czego przykładem jest współdziałanie ICE/UDP z aplikacją *Jingle* (patrz [XEP-0176]) stanowiącą rozszerzenie protokołu XMPP (*eXtensible Messaging and Presence Protocol*) opisywanego w [RFC6120].

Zasadniczo działanie ICE polega na ustanawianiu komunikacji między dwiema encjami SDP (zwanymi *agentami*) na bazie *kandydackich adresów transportowych*, które każdy agent może wykorzystywać do komunikacji z innymi agentami. Adresem kandydackim może być zarówno lokalny adres transportowy hosta (dla hosta multihomed każdy z takich adresów), zwrotny adres transportowy, jak i adres transportowy przekazany przez serwer (patrz rysunek 7.11); do odnajdywania adresów kandydackich ICE wykorzystywać może protokoły STUN i TURN. ICE następnie wykonuje przypisanie priorytetów poszczególnym adresom, preferując te, które umożliwiają bezpośrednią komunikację (kosztem tych, które wiążą się z użyciem NAT lub przekaźników). Agenty wymieniają się wzajemnie (przy użyciu SDP) swymi listami priorytetowymi adresów kandydackich, w wyniku czego każdy agent dysponuje identycznym zbiorem dwóch takich list. Wykorzystując tę parę list, każdy agent przystępuje do formowania *par kandydackich* adresów (po jednym z każdej listy), po czym każda para oceniana jest pod kątem przydatności dla celów połączenia; *oba agenty sprawdzają poszczególne pary w tej samej kolejności* i szeregują je w kolejności malejących priorytetów. Dana para adresów otrzymuje tym wyższy priorytet, im przez mniejszą liczbę NAT lub przekaźników prowadzi komunikacja między nimi. Ostatecznego wyboru konkretnej pary dokonuje *agent kontrolujący* (*controlling agent*) wybrany przez ICE: nominuje on poszczególne pary w kolejności malejących priorytetów i wybiera bądź to dowolną przydatną parę (jest to tzw. *agresywna nominacja*), bądź też parę najlepszą ze wszystkich możliwych (ta nominacja nazywa się *regularną*). Wybór konkretnego wariantu nominacji następuje na podstawie odpowiedniego znacznika w komunikacie STUN — ustawienie tego znacznika skutkuje wyborem nominacji agresywnej.

Komunikaty związane z negocjowaniem pary adresów mają postać żądań STUN wymienianych między agentami. Wymiana ta inicjowana jest przez stoper bądź też w wyniku komunikatu przychodzącego od agenta-partnera (mamy wówczas do czynienia z tzw. *dopasowywaniem wyzwalanym — triggered check*). Odpowiedzi na żądania dopasowywania także są komunikatami STUN, zawierającymi żądaną informację adresową; w niektórych sytuacjach prowadzi to do udostępnienia agentowi nowego adresu zwrotnego (np. dlatego, że na drodze komunikacji między agentami pojawia się nowy NAT, różny od wykrytego przy pierwszym użyciu serwerów STUN lub TURN). Gdy tak się stanie, agent otrzymuje nowy *zwrotny adres kandydacki partnera* (*peer-reflexive candidate*), który ICE dodaje do aktualnej listy adresów kandydackich. Dialog negocjowania adresów jest przez ICE kontrolowany za pomocą mechanizmu krótkoterminowego zaufania STUN oraz atrybutu STUN `FINGERPRINT`. Gdy wykorzystywany jest TURN, klient ICE wykorzystuje jego mechanizm pozwoleń (*permissions*) w celu ograniczenia liczby wiązań TURN ze zdalnymi adresami kandydackimi.

ICE jest technologią skalowalną, realizującą koncepcję różnicowania implementacji. Opisane powyżej funkcje implementowane są kompletnie w wersji pełnej (*Full*), natomiast wersja ograniczona („lekka" — *Lite*) zaprojektowana jest do wdrażania w systemach niewykorzystujących NAT. Agent implementujący wersję *Lite* nie realizuje funkcji sprawdzania adresów, nie pełni także nigdy roli agenta kontrolującego (*controlling*), chyba że wchodzi w interakcję z inną implementacją wersji *Lite*. Typ implementacji wskazywany jest w przesyłanych komunikatach STUN. Od wszystkich implementacji ICE wymaga się zgodności ze specyfikacją STUN ([RFC5389]), choć implementacje wersji *Lite* mogą co najwyżej pełnić funkcje serwerów STUN.

ICE rozszerza listę atrybutów STUN o zestaw przedstawiony w tabeli 7.4.

Tabela 7.4. *Atrybuty STUN definiowane przez ICE*

Nazwa	Typ	Znaczenie
PRIORITY	0x0024	Obliczona wartość priorytetu odnośnego adresu kandydackiego
USE-CANDIDATE	0x0025	Wskazuje na wybór adresu kandydackiego przez agenta kontrolującego
ICE-CONTROLLED	0x8029	Wskazuje nadawcę komunikatu jako agenta kontrolowanego (*controlled*)
ICE-CONTROLLING	0x802A	Wskazuje nadawcę komunikatu jako agenta kontrolującego (*controlling*)

Komunikaty wymieniane w ramach dopasowywania adresów są żądaniami STUN zawierającymi atrybut PRIORITY, którego wartość równa jest wartości priorytetu obliczonej przez algorytm opisany w sekcji 4.1.2 dokumentu [RFC5245]. Atrybuty ICE-CONTROLLING and ICE-CONTROLLED, włączane do wspomnianych żądań, odróżniają agenta czynnego, czyli *kontrolującego* (*controlling*), od agenta biernego, czyli *kontrolowanego* (*controlled*). Agent kontrolujący może także włączać do żądań atrybut USE-CANDIDATE, wyróżniając w ten sposób adresy, które postanowił przeznaczyć do przyszłego użytku.

7.5. Konfigurowanie NAT i firewalli filtrujących

Mechanizmy NAT wymagają generalnie niewielkich zabiegów konfiguracyjnych (no, może z wyjątkiem funkcji forwardowania portów), ale już w przypadku firewalla odpowiednie skonfigurowanie staje się czynnikiem na miarę „być albo nie być" jego użyteczności. W wielu urządzeniach domowych funkcje NAT, firewalla i trasowania IP zamknięte są w pojedynczym urządzeniu i chociaż konfigurowanie każdej z nich jest niezależne w sensie logicznym, fizycznie zostaje zintegrowane w postaci pojedynczego pliku ustawień czy wspólnego interfejsu wiersza poleceń lub też podporządkowane pojedynczemu programowi narzędziowemu (albo zestawowi takich programów, z których każdy ukierunkowany jest raczej na obsługę urządzenia, a nie jego konkretnej funkcji).

7.5.1. Reguły firewalla

Wykonywane przez firewall filtrowanie pakietów sterowane jest precyzyjnymi regułami określającymi kryteria akceptowania albo odrzucania poszczególnych pakietów. Obecnie administrator konfigurujący router czyni to na bazie jednej lub wielu list kontroli dostępu (ACL — *Access Control Lists*). Każda z takich list zawiera ciąg *reguł* (*rules*), zaś na każdą regułę składają się zwykle *kryteria dopasowywania wzorca* (*pattern-matching criteria*) i akcja (*action*) do wykonania. Kryteria dopasowywania wzorca umożliwiają sformułowanie reguły w kategoriach zawartości pól pakietu, na poziomie warstwy sieciowej lub transportowej — czyli m.in. adresów IP, numerów portów oraz typów komunikatów ICMP — oraz w podziale na *kierunki*: reguła może stosować się do pakietów wychodzących, przychodzących lub obu tych kategorii. Wiele firewalli umożliwia także umiejscowienie stosowania reguły na określonym etapie przetwarzania pakietu, np. przed podjęciem decyzji o trasowaniu albo po jej podjęciu — elastyczność ta staje się użyteczna zwłaszcza w przypadku firewalli z wieloma interfejsami.

Dla każdego przychodzącego pakietu podejmowana jest próba dopasowania odpowiedniej reguły poprzez stosowanie następnych reguł w ściśle określonej kolejności; w większości firewalli analiza ta zatrzymuje się na pierwszej „pasującej" regule nakazującej forwardowanie albo zablokowanie pakietu, a opcjonalnie także zwiększającej określony licznik i zapisującej zdarzenia w dzienniku. Niektóre firewalle mogą ponadto oferować dodatkową funkcjonalność formułowania reguł w kontekście przekazywania (albo blokowania) pakietu kierowanego do określonej aplikacji lub określonego hosta; inwencja różnych producentów jest w tym względzie dość zróżnicowana, choć wielu producentów routerów klasy *enterprise* przyjęło format list ACL zdefiniowany przez firmę Cisco Systems. W zastosowaniach domowych listy ACL konfigurowane są zazwyczaj za pomocą prostego interfejsu opartego na przeglądarce WWW.

Jeden z popularnych programów do budowania konfiguracji firewalli — `iptables` — stanowiący integralną część nowszych dystrybucji Linuksa, przeznaczony jest do definiowania filtrów na bazie frameworku `NetFilter` (patrz [NFWEB]). Stanowi on wynik rozbudowy wcześniejszego programu `ipchains` i umożliwia definiowanie reguł zarówno do filtrowania bezstanowego, jak i z pamięcią o stanie, również w kontekście NAT i NAPT. Prześledzimy jego działanie na kilku przykładach, dających przynajmniej ogólne wyobrażenie o możliwościach współczesnych firewalli i urządzeń NAT.

Program `iptables` opiera się na koncepcji *tabel filtrowania* (*filter tables*) i *łańcuchów filtrowania* (*filter chains*). Każda tabela zawiera pewną liczbę predefiniowanych łańcuchów oraz opcjonalnie łańcuchy definiowane przez użytkownika. Trzy predefiniowane tabele noszą nazwy `filter`, `nat` i `mangle`. Tabela `filter` odpowiedzialna jest za podstawowe filtrowanie pakietów i zawiera trzy predefiniowane łańcuchy, `INPUT`, `FORWARD` i `OUTPUT`, odpowiadające (kolejno) pakietom przeznaczonym dla programów działających w routerze implementującym firewall, pakietom przechodzącym przez router w procesie trasowania oraz pakietom generowanym przez komputer implementujący firewall. Predefiniowanymi łańcuchami tabeli NAT są `PREROUTING`, `OUTPUT` i `POSTROUTING`, zaś tabela `mangle`, zawierająca wszystkie pięć wymienionych łańcuchów, wykorzystywana jest do doraźnego nadpisywania zawartości pakietów.

Każdy łańcuch filtrowania jest w istocie ciągiem reguł, każda zaś reguła określa kryteria dopasowywania i wykonywaną akcję; akacja ta — zwana oficjalnie *targetem* — może polegać na wykonaniu czynności specyfikowanych w łańcuchu definiowanym przez użytkownika lub czynności predefiniowanych określonych przez łańcuchy `ACCEPT`, `DROP`, `QUEUE` i `RETURN`. Napotkanie reguły specyfikującej kryteria spełniane przez pakiet skutkuje natychmiastowym wykonaniem dla tego pakietu stosownej akcji: `ACCEPT` oznacza forwardowanie pakietu, `DROP` powoduje jego odrzucenie, `QUEUE` skutkuje przekazaniem pakietu do przetworzenia przez wskazany program użytkowy, zaś `RETURN` powoduje wznowienie przetwarzania przez łańcuch poprzednio używany — przetwarzanie łańcuchów może być hierarchizowane (zagnieżdżane) na wzór wywoływania podprogramów z programu nadrzędnego.

Kompletny proces konfigurowania firewalla jest niewątpliwie dość złożony i specyficzny dla potrzeb konkretnego użytkownika oraz wykorzystywanych przez niego usług, zatem wszelkie próby jego systematycznego opisu z góry skazane są na niepowodzenie (lub powodzenie szczątkowe). W zamian zaprezentujemy wobec tego kilka przykładów — prostych, lecz ilustrujących możliwości programu `iptables`. Oto przykład skryptu konfiguracyjnego linuksowego firewalla, przeznaczonego do wykonania za pomocą procesora powłoki (np. `bash`):

```
EXTIF="ext0"
INTIF="eth0"
LOOPBACK_INTERFACE="lo"
ALL="0.0.0.0/0" # reguła domyślna dla wszystkich pakietów

# odrzucanie jako domyślna akcja dla każdego pakietu
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# akceptowanie jako reguła domyślna dla ruchu lokalnego
iptables -A INPUT -i $LOOPBACK_INTERFACE -j ACCEPT
iptables -A OUTPUT -i $LOOPBACK_INTERFACE -j ACCEPT

# akceptowanie żądań DHCP przychodzących na wewnętrzny interfejs
iptables -A INPUT -i $INTIF -p udp -s 0.0.0.0 \
--sport 67 -d 255.255.255.255 --dport 68 -j ACCEPT

# odrzucanie podejrzanych/nietypowych pakietów TCP bez ustawionych znaczników
iptables -A INPUT -p tcp --tcp-flags ALL NONE -j DROP
```

W kolejnych wierszach powyższego skryptu widzimy przejawy możliwej elastyczności w definiowaniu kryteriów list filtrowania.

- Na początku każdemu z łańcuchów przypisana zostaje domyślna akcja (opcja `-P`) odrzucania pakietów niespełniających żadnej reguły.
- Następne dwie reguły nakazują domyślne akceptowanie ruchu lokalnego (czyli przychodzącego lub wychodzącego przez pseudointerfejs `lo`) — łańcuchom `INPUT` i `OUTPUT` tabeli `filter` przypisany zostaje target `ACCEPT` (opcja `-j` — od *jump* — oznacza „skok" do wskazanego targetu).
- W kolejnej regule widzimy parę portów (67, 68), nieokreślony źródłowy adres IPv4 (`0.0.0.0`) i rozgłoszeniowy adres docelowy (`255.255.255.255`), czyli elementy charakterystyczne dla protokołu DHCP.
- Ostatnia reguła nakazuje odrzucanie pakietów TCP, w których wyzerowane są wszystkie znaczniki (fraza `-tcp-flags ALL NONE`): w „normalnym" ruchu TCP w pierwszym pakiecie ustawiony jest znacznik `SYN`, w następnych — znacznik `ACK`.

Mimo iż składnia prezentowanych poleceń jest specyficzna dla programu `iptables`, funkcje uruchamiane tymi poleceniami realizowane są (być może z nieznacznymi zmianami) przez większość współczesnych firewalli.

7.5.2. Reguły NAT

W większości prostych routerów funkcje NAT konfigurowane są łącznie z funkcjami firewalla. W systemach linii Windows funkcje NAT ukrywają się pod opcją *Udostępnianie połączenia internetowego* (w skrócie ICS, od *Internet Connection Sharing*), w Linuksie NAT nazywane jest *maskaradą IP*. W Windows XP udostępnianie połączenia internetowego cechuje się pewną osobliwością. Komputerowi, na którym zostaje uruchomione, przypisany zostaje wewnętrzny adres IPv4 `192.168.0.1` i na komputerze tym uruchomione zostają serwery DHCP i DNS. Komputerom, którym ma zostać udostępnione połączenie, przypisane zostają adresy z prefiksem `192.168.0/24`, a ich domyślnym serwerem

DNS staje się węzeł o adresie 192.168.0.1 (czyli komputer udostępniający połączenie). Te sztywne reguły uniemożliwiają uruchomienie ICS w sieci, gdzie usługi DHCP i DNS są już zapewnione przez inne komputery, lub adresy 192.168.0.*x* używane są do innych celów. Zmiana domyślnego prefiksu 192.168.0/24 możliwa jest jedynie przez bezpośrednią edycję zawartości rejestru systemowego.

Udostępnianie połączenia internetowego można włączać i wyłączać za pośrednictwem kreatora konfiguracji sieci bądź na karcie *Zaawansowane* właściwości konkretnego połączenia. Opcjonalnie użytkownik może również zezwolić innym użytkownikom na kontrolowanie (szczególnie wyłączanie) udostępnianego połączenia; cecha ta, znana pod nazwą *odnajdowania urządzeń bram internetowych i sterowania* (*Internet Gateway Device Discovery and Control*, w skrócie IGDDC) i realizowana przy użyciu frameworku *Universal Plug and Play* (UPnP) opisywanego w punkcie 7.5.3, używana jest przez klienta do kontrolowania bramy internetowej. Pod pojęciem „kontrolowania" kryją się możliwości włączania, wyłączania i odczytywania różnorodnych komunikatów informujących o statusie bramy. Wbudowany w Windows firewall (zwany *Zaporą systemu Windows*), funkcjonujący we współpracy z ICS, umożliwia *definiowanie usług* jako funkcję analogiczną do forwardowania portów. Opcja ta, dostępna na karcie ustawień zaawansowanych wspomnianej zapory, umożliwia dodawanie programów usługowych i edytowanie zestawu portów skojarzonych z tymi programami. Jest to więc de facto mechanizm konfigurowania NAPT dla połączeń przychodzących.

W Linuksie funkcja maskarady także skojarzona jest implementacyjnie z firewallem; widoczny poniżej skrypt stanowi prosty przykład konfigurowania maskarady; jako taki służy jedynie do celów prezentacji i nie nadaje się raczej do rzeczywistego zastosowania:

```
EXTIF="ext0"
echo "Domyślna polityka forwardowania: odrzucanie"
iptables -P FORWARD DROP

echo "Włączenie NAT na $EXTIF dla hostów 192.168.0.0/24"
iptables -t nat -A POSTROUTING -o $EXTIF -s 192.168.0.0/24 \
-j MASQUERADE

echo "Polityka forwardowania: odrzucanie nieznanych pakietów"
iptables -A INPUT -i $EXTIF -m state --state NEW,INVALID -j DROP
iptables -A FORWARD -i $EXTIF -m state --state NEW,INVALID -j DROP
```

Pierwsze wywołanie programu `iptables` powoduje domyślne odrzucanie pakietów niepasujących do żadnej reguły. Kolejne wywołanie włącza funkcjonalność NAT (opcja `-t nat`) i zezwala na nadpisywanie adresów hostów w podsieci 192.168.0.0/24 przez dowolny ruch IPv4, po dokonaniu jego trasowania (opcja `POSTROUTING`) do dozwolonego interfejsu.

Ze względu na „stanowy" tryb funkcjonowania NAT możliwe jest dostosowanie reguł zapisywanych w tabeli `filter` w taki sposób, by akceptowany był jedynie ruch w ramach połączeń znanych NAT. Dwa ostatnie wiersze dedykowane są właśnie regułom odrzucania (`-j DROP`) dla pakietów nieprawidłowych (`INVALID`) oraz stanowiących próby inicjowania nowych połączeń (`NEW`) — i to zarówno pakietów przeznaczonych dla programów uruchomionych w urządzeniu NAT (`INPUT`), jak i przychodzących do urządzenia w celu forwardowania (`FORWARD`).

7.5.3. Bezpośrednia interakcja z NAT i firewallami — UPnP, NAT-PMP i PCP

W wielu sytuacjach konieczna jest bezpośrednia interakcja systemu klienta z jego firewallem. Przykładem tego może być opisywane wcześniej „wybijanie dziury" zapobiegające odrzucaniu zewnętrznych połączeń kierowanych do określonego portu lokalnego hosta. Inny przykład to firewall proxy, o którego istnieniu i tożsamości musi być informowany każdy klient, żądający komunikacji ze światem zewnętrznym, od którego wspomniany firewall go odgradza. Opracowano kilka protokołów ułatwiających komunikowanie się klientów z firewallami; do najpopularniejszych należą *Universal Plug and Play* (UPnP) i *NAT Port Mapping Protocol* (NAT-PMP). Standard UPnP opracowany został przez grupę o nazwie UPnP Forum (patrz [UPNP]); NAT-PMP zdefiniowany został przez IETF w szkicowym (przestarzałym już) dokumencie [XIDPMP]. NAT-PMP obsługiwany jest przez większość systemów Mac OS X. UPnP jest integralnym mechanizmem Windows, może też być dodawany jako opcja do systemów Mac OS X; wykorzystywany jest także w urządzeniach elektroniki użytkowej, zgodnych z wytycznymi Digital Living Network Alliance (DLNA), jako baza ich współpracy z domowymi sieciami komputerowymi (patrz [DLNA]).

W ramach UPnP kontrolowanym urządzeniom przydzielane są adresy IP za pośrednictwem serwerów DHCP, a w razie ich braku — w drodze lokalnego konfigurowania łącza (pisaliśmy o tym obszernie w rozdziale 6.). Następnie protokół o nazwie *Simple Service Discovery Protocol* (SSDP — patrz [XIDS]) oznajmia obecność urządzeń punktom kontrolnym (np. komputerom klienta) i umożliwia tym punktom pobranie dodatkowych informacji ze wspomnianych urządzeń. Protokół SSDP funkcjonuje na bazie hybrydy HTTP/UDP (zamiast standardowego HTTP/TCP) w dwóch wariantach: HTTPU i HTTPMU (patrz [XIDMU]). Drugi z wymienionych wariantów wykorzystuje adresowanie multicast: w wersji IPv4 jest to adres `239.255.255.250` i port `1900`, SSDP oparty na IPv6 wykorzystuje natomiast następujące adresy: `ff01::c` (lokalny dla węzła), `ff02::c` (lokalny dla łącza), `ff05::c` (lokalny dla miejsca sieciowego), `ff08::c` (lokalny dla organizacji) i `ff0e::c` (globalny).

Dalsza kontrola i powiadamianie o zdarzeniach (*eventing*) odbywają się w ramach architektury GENA (*General Event Notification Architecture*) wykorzystującej protokół SOAP (*Simple Object Access Protocol* — prosty protokół dostępu do obiektów), który z kolei realizuje mechanizm *zdalnego wywoływania procedur* (RPC — *Remote Procedure Call*) w oparciu o komunikaty kodowane w języku XML (*eXtensible Markup Language*) powszechnie używanym do kodowania treści stron WWW. Mechanizm UPnP wykorzystywany jest w wielu urządzeniach elektroniki użytkowej, m.in. w odtwarzaczach audio i video oraz urządzeniach pamięciowych. Urządzenia łączące funkcjonalność NAT i firewalli sterowane są za pomocą protokołu IGD (*Internet Gateway Device* — urządzenie bramy internetowej, patrz [IGD]). Protokół ten oferuje wiele możliwości, m.in. zdolność do „uczenia się" mapowania NAT i konfigurowania forwardowania portów. Ciekawi tematu czytelnicy mogą dla celów eksperymentalnych pobrać implementację prostego klienta IGD ze strony domowej projektu MiniUPnP ([UPNPC]). Nowsza wersja UPnP IGD (patrz [IGD2]) rozszerza mechanizm UPnP o generalne wsparcie dla IPv6.

Podczas gdy UPnP jest obszernym frameworkiem, łączącym proces NAT z wieloma innym specyfikacjami, NAT-PMP jest alternatywą specyficznie ukierunkowaną na programową komunikację między urządzeniami NAT. NAT-PMP jest elementem specyfikacji

Apple o nazwie *Bonjour*, ukierunkowanej na zerową konfigurację sieci; nie wykorzystuje on procesu odnajdywania bramy, bowiem zarządzane przezeń urządzenie samo zwykle jest domyślną bramą systemową (o czym informacja dostarczana jest przez DHCP). W celu rozpoznawania zewnętrznych adresów NAT protokół NAT-PMP wykorzystuje prostą komunikację typu żądanie-odpowiedź na porcie UDP 5350. Realizuje on także proste powiadamianie obserwatorów o ewentualnej zmianie tych adresów, wysyłając komunikat multicast UDP na adres `224.0.0.1` reprezentujący wszystkie hosty (*All Hosts*), na porcie 5351. Idea funkcjonowania NAT-PMP może być rozszerzona na konfigurację SPNAT (patrz punkt 7.3.7), co stanowi istotę protokołu PCP (*Port Control Protocol* — protokół sterowania portami, patrz [IDPCP]).

7.6. Migracja na adresy IPv6 i współistnienie adresów IPv4/IPv6 z wykorzystaniem NAT

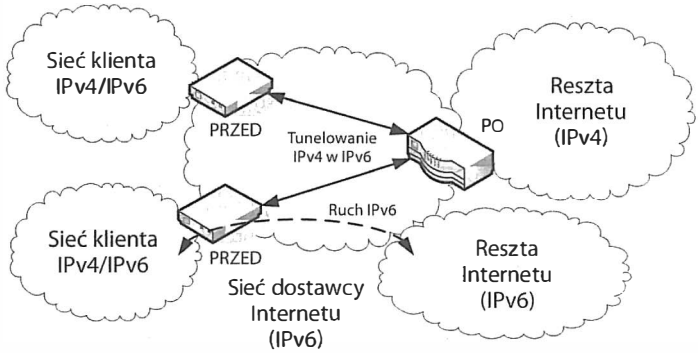
Rozdysponowanie przez IANA ostatniej porcji prefiksów IPv4 między urzędy regionalne — co nastąpiło 1 lutego 2011 roku — stanowiło wyraźny sygnał do przyspieszenia prac w kierunku wdrażania IPv6. Stało się oczywiste, że hosty powinny być wyposażane w funkcjonalność „dualnego stosu" IPv4/IPv6, czyli kompletną implementację obu wersji IP (patrz [RFC4213]), a usługi sieciowe powinny docelowo ewoluować w kierunku wyłącznie IPv6. Oczywiście, IPv4 i IPv6 skazane są na współegzystencję przez okres, który trudno zdefiniować (być może — na zawsze), bo w wielu przypadkach względy ekonomiczne mogą przemawiać za pozostawaniem tej czy innej sieci w wersji IPv4. Koniecznością stało się więc opracowanie mechanizmów, z jednej strony, ułatwiających transformację w kierunku IPv6, z drugiej zaś, umożliwiających bezproblemowe współdziałanie IPv4 i IPv6.

Dwa podstawowe podejścia zmierzające do pogodzenia obu wersji IP to tunelowanie i translacja. W czołówce pierwszej grupy plasują się Teredo (patrz rozdział 10.), Dual-Stack Lite (DS-Lite) i (opisywane w rozdziale 6.) IPv6 Rapid Deployment (6rd), natomiast opisana w [RFC6144] w translacja adresów opiera się na wbudowanych (*embedded*) adresach IPv4, o których wspominaliśmy w punkcie 2.3.6. Omówimy teraz dokładniej DS-Lite oraz wspomniany framework translacyjny.

7.6.1. Dualny stos TCP/IP (DS-Lite)

DS-Lite, zdefiniowany w dokumencie [RFC6333], zaprojektowany został z myślą o dostawcach Internetu, którzy, przestrajając swą infrastrukturę całkowicie na IPv6, muszą jednocześnie zapewnić świadczenie dotychczasowych usług także klientom pozostającym przy IPv4. Cel ten zrealizowano, łącząc technikę SPNAT z techniką tunelowania IPv4 w IPv6, zwaną *softwire tunelling* (patrz [RFC5571]), w efekcie czego spora liczba klientów używających IPv4 może zostać obsłużona za pomocą niewielkiej liczby deficytowych adresów. Ideę tę przedstawiono schematycznie na rysunku 7.16.

Każda sieć kliencka na rysunku 7.16 z założenia operuje dowolną kombinacją IPv4 i IPv6 (w szczególności — wyłącznie IPv4 albo wyłącznie IPv6), natomiast infrastruktura dostawcy Internetu oparta jest w całości na IPv6. Klienci używający IPv6 uzyskują więc



Rysunek 7.16. *DS-Lite umożliwia dostawcy operującemu infrastrukturą IPv6 świadczenie usług na rzecz klientów wykorzystujących obie wersje protokołu IP*

dostęp do Internetu w ramach konwencjonalnego trasowania, natomiast dla klientów używających IPv4 dostęp do Internetu zorganizowany jest w bardziej złożony sposób. Element oznaczony na rysunku jako PRZED dostarcza klientowi podstawowe usługi IPv4 (m.in. DHCP i DNS proxy), jednocześnie enkapsulując kliencki ruch IPv4 w tunelach zakończonych punktem oznaczonym jako PO. Element PO wykonuje dekapsulację ruchu kierowanego do Internetu oraz enkapsulację ruchu w kierunku przeciwnym, jednocześnie spełniając funkcje NAT, a właściwie SPNAT: wielu klientów może przecież używać tego samego adresu IPv4, a rozróżnienie pakietów płynących z Internetu do poszczególnych klientów odbywa się na podstawie identyfikatorów klienckich. Każdy element PRZED może rozpoznać adres IPv6 urządzenia PO na drugim końcu tunelu, wykorzystując opcję DHCPv6 o nazwie AFTR-name (patrz [RFC6334]).

Warto w tym miejscu przypomnieć technologię IPv6 Rapid Deployment (6rd) opisywaną w podpunkcie 6.2.5.7, ponieważ spełnia ona rolę dokładnie odwrotną — umożliwia klientom wykorzystującym IPv6 korzystanie z usług dostawcy o infrastrukturze opartej na IPv4. Ponadto, w przeciwieństwie do DS-Lite, adres drugiego końca tunelu IPv4 wyznaczany jest algorytmicznie, w sposób bezstanowy. Translacja bezstanowa realizowana jest także w ramach frameworku dla pełnej translacji między IPv4 a IPv6, opisywanego w następnym punkcie.

7.6.2. Translacja między IPv4 a IPv6 przy użyciu NAT i ALG

Podstawową wadą opisywanego w poprzednim punkcie tunelowania jest niemożność bezpośredniego korzystania przez host z usług innego hosta, jeżeli hosty te należą do różnych rodzin adresowych — host implementujący wyłącznie IPv6 może łączyć się bezpośrednio tylko z hostem implementującym IPv6. Jest to — oczywiście — sytuacja bardzo niekorzystna, bo z wielu usług internetowych, istniejących tylko w wersji IPv4, nagle nie można skorzystać w nowych systemach opartych w całości na IPv6. Aby rozwiązać ten problem, podjęto w latach 2008–2010 znaczące wysiłki prowadzące do opracowania frameworku bezpośredniej translacji adresów IPv4 a IPv6. Będący ukoronowaniem tych wysiłków protokół *Network Address Translation — Protocol Translation* (NAT-PT) opisany w [RFC2766] okazał się jednak źródłem złych doświadczeń, co spowodowało ostatecznie jego wycofanie z użycia i nadanie mu statusu historycznego (patrz [RFC4966]).

Framework będący następcą NAT-PT definiowany jest w dokumencie [RFC6144]. Wykorzystuje zarówno stanowe, jak i bezstanowe metody konwersji między adresami IPv4 a IPv6, translacji na potrzeby DNS (patrz rozdział 11.) oraz ewentualnego dodatkowego zachowania (lub ALG) na potrzeby m.in. ICMP i FTP. Omówimy dokładniej szczegóły translacji adresów IP w oparciu o [RFC6145], [RFC6146] oraz translację opisywaną w [RFC6052], o której wspominaliśmy w punkcie 2.5.1. Szczegóły translacji specyficzne dla innych protokołów omawiane będą w kolejnych rozdziałach.

7.6.2.1. Adresy IPv4-konwertowalne i IPv4-przetłumaczalne

W rozdziale 2. wspominaliśmy o wbudowanych (*embedded*) adresach IPv4. Są to w istocie adresy IPv6, które za pomocą specjalnej funkcji mogą być przekształcane na odpowiadające im adresy IPv4. Funkcja ta jest łatwo odwracalna. Wbudowane adresy IPv4 istnieją w dwóch odmianach, zwanych adresami *IPv4-konwertowalnymi* (*IPv4-converted addresses*) i adresami *IPv4-przetłumaczalnymi* (*IPv4-ranslatable addresses*). Między wymienionymi zbiorami adresów istnieje relacja zawierania: adresy IPv4-przetłumaczalne są podzbiorem adresów IPv4-konwertowalnych, które z kolei zawierają się w zbiorze adresów wbudowanych; te zaś są podzbiorem wszystkich adresów IPv6. Adresy IPv4-przetłumaczalne to adresy IPv6, których odpowiedniki IPv4 mogą zostać ustalone na drodze konwersji bezstanowej (o której piszemy w podpunkcie 7.6.2.2).

Jak wspominaliśmy w rozdziale 2., algorytmiczna translacja adresów IPv4 na adresy IPv6 realizowana jest na podstawie prefiksu, którym może być „dobrze znany" prefiks (WKP — *Well-Known Prefix*) `64:ff9b::/96` bądź inny prefiks, będący w posiadaniu dostawcy Internetu i przeznaczony przez niego do tego właśnie celu. WKP używany jest wyłącznie do reprezentowania globalnie trasowalnych adresów IPv4, nie do adresów prywatnych (w sensie dokumentu [RFC1918]) ani do tworzenia adresów IPv4-przetłumaczalnych — zakresem ich obowiązywania jest sieć dostawcy, niecelowe jest więc ich używanie w zakresie globalnym.

Wspomniany prefiks WKP jest o tyle interesujący, że jego szczególna postać czyni go *neutralnym względem internetowej sumy kontrolnej*, której szczegóły wyjaśnialiśmy w rozdziale 5. Jeśli mianowicie potraktujemy prefiks `64:ff9b::/96` jako ciąg szesnastkowych słów 16-bitowych `0x0064`, `0xff9b`, `0x0000`, `0x0000`, `0x0000`, `0x0000`, `0x0000`, `0x0000`, to obliczona dla tego ciągu suma w uzupełnieniu do 1 równa będzie `0xffff`, co po negacji bitowej daje wartość 0. Jeżeli więc poprzedzimy adres IPv4 wspomnianym prefiksem, suma kontrolna tak otrzymanego ciągu będzie taka sama jak suma kontrolna oryginalnego adresu — translacja z użyciem prefiksu WKP nie zmienia więc sumy kontrolnej nagłówka pakietu. Oczywiście, identycznej własności wymaga się od prefiksu translacyjnego wybranego przez dostawcę.

W dwóch kolejnych podsekcjach używać będziemy notacji `To6(A4, P)` na oznaczenie adresu IPv6, który powstaje w wyniku poprzedzenia prefiksem `P` adresu IPv4 `A4`. Analogicznie zapis `To4(A6, P)`oznaczać będzie operację odwrotną, dającą oryginalny adres IPv4, odtworzony na podstawie adresu IPv6 `A6` powstałego za pomocą operacji `To6` z użyciem prefiksu `P`. Jak łatwo się zorientować funkcje `To4()` i `To6()` są wzajemnie odwrotne: z wyjątkiem pewnych szczególnych sytuacji `To6(To4(A6, P), P) = A6` oraz `To4(To6(A4, P), P) = A4`.

7.6.2.2. Translacja bezstanowa

Bezstanowa translacja IP/ICMP (*Stateless IP/ICMP Translation*, w skrócie SIIT — patrz [RFC6145]) to określenie translacji pakietów IPv4 na pakiety IPv6 bez użycia tablic stanów. Translacja ta przeprowadzana jest za pomocą funkcji `To6()` w połączeniu z predefiniowanym schematem konwersji nagłówka jako całości. Ogólnie rzecz biorąc, nie podlegają translacji opcje IPv4 (są ignorowane) ani nagłówki rozszerzeń IPv6 (z wyłączeniem nagłówka *Fragmentacja*); wyjątkiem jest opcja IPv4 *Trasowanie źródłowe* — jeśli występuje, pakiet zostaje odrzucony, a do nadawcy wysłany zostaje odpowiedni komunikat ICMP (*Destination Unreachable, Source Route Failed* — patrz rozdział 8). Szczegóły translacji nagłówka datagramu IPv4 na nagłówek datagramu IPv6 przedstawione są w tabeli 7.5.

Tabela 7.5. *Zasady tworzenia nagłówka IPv6 w drodze bezstanowej translacji nagłówka IPv4*

Pole nagłówka IPv6	Metoda tworzenia
Wersja	Wpisanie wartości 6
DSF/ECN	Skopiowanie z nagłówka IPv4
Etykieta przepływu	Wpisanie wartości 0
Rozmiar ładunku użytecznego	Przepisanie wartości z pola *Całkowity rozmiar* nagłówka IPv4 pomniejszonej o rozmiar nagłówka IPv4 wraz z opcjami
Następny nagłówek	Przepisanie wartości z pola *Protokół* nagłówka IPv4 [z wyjątkiem wartości 1 (ICMPv4), która zastąpiona zostaje wartością 58 (ICMPv6)] albo wpisanie wartości 44 wskazującej nagłówek *Fragmentacja*, jeśli tworzony datagram IPv6 jest fragmentem lub w nagłówku nie jest ustawiony bit *DF*
Limit przeskoków	Przepisanie wartości z pola *TTL* nagłówka IPv4, pomniejszonej o 1 (jeśli wartość pola *TTL* jest zerowa, następuje odrzucenie pakietu i wysłanie komunikatu ICMP, *Time Exceeded* — patrz rozdział 8.)
Źródłowy adres IP	Wpisanie wyniku funkcji `To6(S, P)`, gdzie *S* jest adresem źródłowym w nagłówku IPv4
Docelowy adres IP	Wpisanie wyniku funkcji `To6(D, P)`, gdzie *D* jest adresem docelowym w nagłówku IPv4

Po zakończeniu translacji oryginalny nagłówek usuwany jest z pakietu i zastępowany nowo utworzonym nagłówkiem IPv6. Gdy poddawany translacji pakiet IPv4 przekracza wartość MTU dla następnego łącza, a bit *DF* jest wyzerowany, możliwe jest wyprodukowanie pakietu w kilku fragmentach, z których każdy zawiera nagłówek *Fragmentacja*. Dzieje się tak również wtedy, gdy wspomniany pakiet IPv4 sam jest fragmentem większego pakietu. W dokumencie [RFC6145] znajduje się nawet zalecenie, by nagłówek *Fragmentacja* włączany był do wynikowego pakietu IPv6 zawsze wtedy, gdy w oryginalnym pakiecie IPv4 wyzerowany jest bit *DF*, a nawet wtedy, gdy translator nie wykonuje fragmentacji tworzonego pakietu, a przybywający pakiet IPv4 nie jest pofragmentowany. Stanowi to dla odbiorcy wynikowego pakietu IPv6 wskazówkę, że nadawca oryginalnego pakietu IPv4 prawdopodobnie nie wykorzystuje PMTUD. Gdy do wynikowego pakietu IPv6 włączony jest nagłówek *Fragmentacja*, wartości jego pól ustawione zostają zgodnie z tabelą 7.6.

Tabela 7.6. *Wartości pól nagłówka rozszerzenia Fragmentacja wynikowego pakietu IPv6 przy bezstanowej translacji pakietu IPv4*

Pole nagłówka	Przypisywana wartość
Następny nagłówek	Kopiowana z pola *Protokół* nagłówka IPv4
Offset fragmentu	Kopiowana z pola *Offset fragmentu* nagłówka IPv4
Bit M (More fragments)	Kopiowana z bitu `M` nagłówka IPv4
Identyfikacja	Dwa najmniej znaczące bajty są kopią pola *Identyfikacja* w nagłówku IPv4, dwa najstarsze bajty są wyzerowane

Translacja w kierunku odwrotnym (z IPv6 na IPv4) związana jest z tworzeniem nagłówka pakietu IPv4 na bazie zawartości wybranych pól pakietu IPv6. Oczywiście, ze względu na znacznie większą, w porównaniu z IPv4, przestrzeń adresową IPv6, host implementujący wyłącznie IPv4 będzie miał dostęp jedynie do niewielkiej części hostów IPv6 w Internecie. W tabeli 7.7 opisano sposób ustalania wartości poszczególnych pól wynikowego nagłówka IPv4 na podstawie niepofragmentowanego pakietu IPv6.

Tabela 7.7. *Zasady tworzenia nagłówka IPv4 w drodze bezstanowej translacji nagłówka niepofragmentowanego pakietu IPv6*

Pole nagłówka IPv6	Metoda tworzenia
Wersja	Wpisanie wartości `4`
IHL	Wpisanie wartości `5` (brak opcji IPv4)
DSF/ECN	Skopiowanie z nagłówka IPv6
Całkowity rozmiar	Przepisanie wartości z pola *Rozmiar ładunku użytecznego* nagłówka IPv6 powiększonej o 20
Identyfikacja	Wyzerowanie (ewentualnie wpisanie innej predefiniowanej wartości)
Znaczniki	Ustawienie bitów `M = 0, DF = 1`
Offset fragmentu	Wpisanie wartości `0`
TTL	Przepisanie wartości z pola *Limit przeskoków* nagłówka IPv6, pomniejszonej o `1`, przy czym wynikowa wartość nie może być mniejsza od 1
Protokół	Skopiowanie z pierwszego pola Następny nagłówek, które nie odwołuje się do nagłówka *Fragmentacja*, *Opcje „skok po skoku"*, *Trasowanie* lub *Opcje docelowe*. Wartość `58` (ICMPv6) zostaje zastąpiona wartością `1` (ICMPv4 — patrz rozdział 8.)
Suma kontrolna nagłówka	Wpisanie wartości obliczonej dla utworzonego nagłówka IPv4
Źródłowy adres IP	Wpisanie wyniku funkcji `To4(S, P)`, gdzie *S* jest adresem źródłowym w nagłówku IPv6
Docelowy adres IP	Wpisanie wyniku funkcji `To4(D, P)`, gdzie *D* jest adresem docelowym w nagłówku IPv6

Gdy źródłowy datagram IPv6 zawiera nagłówek *Fragmentacja*, wartości niektórych pól wynikowego nagłówka IPv4 ustalane są inaczej, niż to wynika z tabeli 7.7; różnice przedstawione zostały w tabeli 7.8.

Tabela 7.8. *Nadawanie wartości polom wynikowego nagłówka IPv4 przy bezstanowej translacji nagłówka pofragmentowanego pakietu IPv6*

Pole nagłówka IPv6	Metoda tworzenia
Całkowity rozmiar	Przepisanie wartości z pola *Rozmiar ładunku użytecznego* nagłówka IPv6 powiększonej o 12
Identyfikacja	Skopiowanie dwóch najmniej znaczących bajtów pola *Identyfikacja* w nagłówku rozszerzenia *Fragmentacja*
Znaczniki	Bit `M` staje się kopią swego odpowiednika z nagłówka rozszerzenia *Fragmentacja*, bit `DF` zostaje wyzerowany
Offset fragmentu	Skopiowanie z pola *Offset fragmentu* nagłówka rozszerzenia *Fragmentacja*

Tak więc pofragmentowane datagramy IPv6 przekształcane są przez translator na pofragmentowane datagramy IPv4. Zwróćmy w związku z tym uwagę na pole *Identyfikacja*, które w nagłówku rozszerzenia *Identyfikacja* jest 4-bajtowe, a w nagłówku pakietu IPv4 tylko 2-bajtowe; z konieczności więc dwa najstarsze bajty tego pola są przy konwersji gubione, a to może spowodować, że różne wartości oryginalnego (4-bajtowego) identyfikatora zostaną odwzorowane na tę samą (2-bajtową) wartość wynikową, co w konsekwencji spowoduje zakłócenie procesu reasemblacji fragmentów. Na szczęście jednak, mechanizmy kontroli integralności w warstwach wyższych niezawodnie taką sytuację wykryją, a ryzyko jej wystąpienia nie jest wcale większe niż prawdopodobieństwo konwencjonalnego „zawinięcia"[1] pola *Identyfikacja* w nagłówku IPv4.

7.6.2.3. Translacja wykorzystująca informację o stanie

Ponieważ wiele użytecznych usług internetowych oferowanych jest wciąż tylko w wersji IPv4, istotnego znaczenia nabiera kwestia efektywności komunikowania się hostów implementujących wyłącznie IPv6 z serwerami IPv4. Technologią dedykowaną takiej komunikacji jest *NAT64*, opisywana w dokumencie [RFC6146] i realizująca translację z wykorzystywaniem informacji o stanie, notabene bardzo podobną do translacji bezstanowej opisanej w podpunkcie 7.6.2.2. Narzędzie NAT64 — jako implementujące NAT — zgodne jest ze specyfikacjami grupy BEHAVE; realizuje mapowanie wyłącznie w wariancie („klasie behawioralnej") niezależnym od punktu docelowego, a filtrowanie pakietów wykonywane jest w wariancie zarówno niezależności od punktu docelowego, jak i zależności od adresu docelowego. Oznacza to kompatybilność NAT64 z technikami omijania NAT (ICE, STUN, TURN itp.), ponieważ jednak NAT64 pozbawione jest implementacji tych protokołów, zapewnia jedynie dynamiczną translację adresów dla hostów IPv6 inicjujących połączenia z hostami IPv4.

Translacja adresów w wydaniu NAT64 zbliżona jest do tradycyjnego NAPT (między dwiema rodzinami adresów) z tą różnicą, że translacja IPv4 na IPv6 wykonywana jest szybko i bezpośrednio w sposób opisywany w punkcie 2.5.1 i dokumencie [RFC6052]. Przy konwersji w drugą stronę, wobec ograniczonej dostępności adresów IPv4, wiele różnych adresów IPv6 musi być mapowanych na ten sam adres IPv4, konieczne jest więc również manipulowanie numerami portów TCP i UDP oraz identyfikatorami ICMP (patrz rozdział 8.), co stanowi istotę NAPT.

[1] Czyli powtórzenia się tej samej wartości po wykorzystaniu wszystkich 65536 możliwych — *przyp. tłum.*

NAT64 różni się od swego bezstanowego odpowiednika także sposobem zarządzania fragmentacją pakietów. Odbierane fragmenty pakietów TCP i UDP, o niezerowej transportowej sumie kontrolnej, mogą być bądź to konwertowane niezależnie od siebie, bądź też cacheowane w celu reasemblacji oryginalnego pakietu i jego konwertowania w całości. Ponieważ ten drugi przypadek stanowi doskonałą okazję do przypuszczenia ataku DoS (poprzez zwykłe „zatkanie" pamięci cache lawiną nadsyłanych fragmentów), czas przebywania fragmentu w pamięci ograniczany jest pewną wartością maksymalną, nie mniejszą niż 2 sekundy.

7.7. Ataki na firewalle i NAT

Zważywszy na fakt, że podstawowym zadaniem firewalli jest przeciwdziałanie atakom, można się w ich przypadku spodziewać większej — w porównaniu z innymi systemami — staranności pod względem „szczelności" implementacji i unikania oczywistych niedostatków w zabezpieczeniu. Z jednej strony, to prawda, z drugiej jednak, skuteczność ochrony realizowanej przez firewall zależna jest także od sposobu jego użytkowania (czytaj: konfigurowania) w naturalny sposób obejmującego czynnik ludzki. Nawet najbardziej wymyślne mechanizmy ochronne nie zdadzą się na wiele, jeśli nie zostaną właściwie wykorzystane. Konfigurowanie firewalli nie jest więc zadaniem banalnym, szczególnie w dużych sieciach korporacyjnych, gdzie zasady legalnego dostępu użytkowników do poszczególnych usług mogą zmieniać się nawet kilka razy w ciągu dnia. Należy ponadto pamiętać o tym, że potencjalni intruzi generalnie upatrują szans powodzenia swych ataków w sytuacjach nietypowych czy ekstremalnych dla danego systemu; dla większości firewalli taką nietypową sytuacją jest zarządzanie pofragmentowanym pakietem, bo sposób postępowania z poszczególnymi fragmentami zależny jest w dużym stopniu od informacji zawartych w pierwszym fragmencie (zawierającym podstawowy nagłówek) i właśnie owa kontekstowość jest dla hakerów szczególnie łakomym kąskiem.

W przypadku NAT to, co jest istotą jego użyteczności — maskarada, czyli konwersja adresów IP i być może numerów portów — może stanowić skuteczną broń w arsenale hakera, daje mu możliwość maskowania się przed użytkownikiem „chronionego" hosta — pakiety „wpuszczane" do sieci przez hakera stają się dla tegoż użytkownika nieodróżnialne od legalnych pakietów. Problem ten staje się szczególnie istotny w przypadku firewalli czy NAT konfigurowanych za pomocą programu `ipchains`, bo proste polecenie włączające maskaradę

```
Linux# ipchains -P FORWARD MASQUERADE
```

stwarza taką właśnie sytuację i należy się go wystrzegać: maskarada staje się domyślną polityką forwardowania, niepozostającą w żadnym związku z klasyfikacją obsługiwanego ruchu.

Kolejny typowy problem ze skutecznością firewalli i NAT związany jest z aktualnością utrzymywanej w nich informacji konfiguracyjnej. Jakże często pozostawia się w narzędziach NAT „wybite" dziury dla portów reprezentujących usługi, których świadczenia dawno już zaprzestano. Problemem pokrewnym jest utrata aktualności duplikatów informacji konfiguracyjnej utrzymywanych w pamięciach poszczególnych routerów. Wreszcie innym typowym problemem — i jednocześnie pułapką na nieświadomego operatora czy administratora — jest opcja domyślnego *łączenia* (*merge*) nowo dodawanych reguł z już istniejącymi, co niekiedy prowadzić może do niepożądanych rezultatów.

Problem, jaki dla skuteczności firewalli stwarzają pofragmentowane pakiety, jest bezpośrednią konsekwencją samej konstrukcji procesu fragmentacji (patrz rozdział 10.). Gdy mianowicie datagram dzielony jest na fragmenty, jedynie w pierwszym fragmencie znajdują się informacje istotne z perspektywy bezpieczeństwa, m.in. numery portów. Jest to z kolei rezultat warstwowego charakteru architektury protokołów TCP/IP i związanej z tą architekturą enkapsulacji. Niestety, firewall, otrzymując pośredni fragment datagramu IP, dysponuje nikłą informacją na temat protokołu czy usługi warstwy transportowej, z którą datagram ten jest powiązany. Informację taką zapewnić może kontekstowy („stanowy") model funkcjonowania firewalla, zgodnie z którym wspomnianą informację odczytuje się z pierwszego fragmentu i stosuje do fragmentów następnych — co możliwe jest jednak pod warunkiem, że *początkowy fragment pojawi się jako pierwszy*. Ponieważ sposób funkcjonowania protokołu IP nigdy nie daje takiej gwarancji, konieczne jest cacheowanie tych fragmentów pośrednich, które do firewalla dotrą wcześniej niż fragment początkowy — a to z kolei stwarza okazję do przypuszczenia ataku prowadzącego do wyczerpania zasobów (głównie pamięci podręcznej). Nie wszystkie firewalle prezentują dostateczną inteligencję w tym względzie, odrzucając po prostu fragmenty pośrednie przybyłe wcześniej niż fragment początkowy. Datagramy o dużych rozmiarach powodować więc mogą nieuzasadnioną penalizację legalnego ruchu, co z perspektywy użytkownika jest zwykle sytuacją niezrozumiałą.

7.8. Podsumowanie

Firewalle dostarczają administratorowi sieci mechanizm umożliwiający eliminowanie przepływu informacji, która mogłaby stanowić zagrożenie dla systemu. Firewalle podzielić można na dwie grupy: do pierwszej zaliczają się firewalle filtrujące pakiety, do drugiej należą firewalle proxy. Filtrowanie pakietów może odbywać się w sposób bezstanowy — każdy pakiet traktowany jest wtedy niezależnie od pozostałych — lub wiązać się ze zbieraniem i przetwarzaniem informacji o stanie obsługi poprzednich pakietów. Firewalle filtrujące funkcjonują zwykle jako routery IP. Filtrowanie z informacją o stanie jest — oczywiście — bardziej wyrafinowane, zapewnia obsługę różnych protokołów aplikacyjnych oraz dodatkowe funkcje w rodzaju rejestrowania zdarzeń. Firewall proxy jest odmianą bramy aplikacyjnej; wymaga zaimplementowania obsługi dla każdej potencjalnej usługi świadczonej lub wykorzystywanej przez aplikację, co często wiąże się z dość głęboką ingerencją nie tylko w nagłówki przesyłanych pakietów, lecz także w przesyłaną w nich treść. Protokoły w rodzaju SOCKS umożliwiają współpracę (za pomocą standardowych mechanizmów) aplikacji z firewallami proxy.

Translacja adresów sieciowych (NAT) to mechanizm umożliwiający obsługę relatywnie dużej liczby hostów przy użyciu znacznie mniejszej liczby globalnie trasowalnych adresów IP (często przy użyciu jednego takiego adresu). Translacja taka często łączona jest z funkcjami firewalla, co umożliwia m.in. skrywanie topologii sieci przed światem zewnętrznym poprzez liberalne przepuszczanie ruchu wychodzącego i domyślne blokowanie przychodzących z zewnątrz pakietów, które nie stanowią reakcji na żądanie wewnętrznych hostów. Polityka taka stanowi przeszkodę w sytuacji, gdy któryś z wewnętrznych hostów pełnić ma rolę serwera dostępnego globalnie z Internetu; staje się to możliwe przez odpowiednie skonfigurowanie NAT w taki sposób, by akceptowane były przychodzące z zewnątrz żądania kierowane do określonego portu — mechanizm ten nazywa się *forwardowaniem portów*. NAT okazuje się także pomocne w dziele zapewnienia współeg-

zystencji dwóch rodzin adresów — IPv4 i IPv6 — nieuchronnej w okresie przejściowym do pełnej konwersji Internetu na wersję IPv6 (co z pewnością szybko nie nastąpi). Dodatkowo, technika SPNAT polegająca na przesuwaniu urządzeń NAT z poziomu siedzib klientów na poziom infrastruktury dostawcy Internetu stanowi dalszy krok w kierunku zmniejszenia tempa wyczerpywania się resztek dostępnych adresów IPv4, lecz jednocześnie wprowadza nowe utrudnienie dla klientów zamierzających uruchamiać w swych sieciach globalnie dostępne serwery usługowe.

Mechanizm NAT jest z natury mechanizmem nieprzezroczystym, szczególnie na poziomie warstwy aplikacji, toteż dla wielu aplikacji istotna jest informacja na temat adresów (i portów) zewnętrznych, na jakie zamapowane zostaną przez NAT adresy (i porty) lokalnie używane przez aplikację. Informację tę mogą aplikacje uzyskiwać w sposób pewny i bezpośredni w drodze dialogu z implementacją NAT lub — z braku takowej możliwości — w sposób jednostronny, oparty na heurystykach, a więc niepewny (co znajduje odzwierciedlenie w akronimie określającym takie aplikacje — UNSAF, od *UNilateral Self-Address Fixing*, z jednocześnie wyraźną aluzją do przymiotnika *unsafe* — niebezpieczny). Mimo opracowania przez grupę zadaniową BEHAVE specyfikacji ujednolicających szczegóły działania NAT, nie we wszystkich implementacjach NAT specyfikacje te zostały uwzględnione, co zmusiło autorów aplikacji do poszukiwania rozwiązań umożliwiających omijanie NAT.

Omijanie NAT to w istocie opracowywanie mechanizmów umożliwiających komunikowanie się ze sobą hostów rozdzielonych barierami NAT, często na wielu poziomach. Podstawowym protokołem umożliwiającym taką komunikację jest STUN, z jego specyficznym zastosowaniem o nazwie TURN, zakładającym pośrednictwo zewnętrznego serwera dostępnego globalnie w Internecie. Dodając do tego mechanizm optymalnego wyboru adresów używanych do komunikacji, otrzymujemy kompletny protokół komunikacyjny omijający NAT — przykładem takiego protokołu jest ICE.

ICE uwzględnia wszelkie adresy, jakie mogą być użyte do komunikowania się dwóch węzłów, w tym adresy uzyskane za pomocą STUN i TURN; spośród wszystkich możliwych par adresów wybrana zostaje ta zapewniająca komunikację optymalną lub zbliżoną do optymalnej. Protokół ICE zyskał sobie szeroką popularność w telefonii internetowej VoIP, gdzie funkcjonuje w połączeniu z protokołem sygnalizacyjnym SIP.

Skuteczność firewalli zależna jest przede wszystkim od umiejętnego wykorzystywania oferowanych przez nie możliwości, czyli od właściwego konfigurowania. Podstawowe, domyślne ustawienia firewalla okazują się wystarczające dla większości sieci domowych, choć mogą wymagać pewnej interwencji w przypadku korzystania z niektórych nietypowych usług. Ponadto użytkownik oddzielony od Internetu urządzeniem NAT, a zamierzający udostępniać na zewnątrz serwer usługowy, musi skonfigurować firewall pod kątem funkcji forwardowania portów. Niektóre aplikacje wspomagają użytkownika w dziele konfigurowania firewalla i udostępniają mechanizmy bezpośredniej komunikacji z NAT, oparte na protokołach (m.in.) UPnP i NAT-PMP. Umożliwiają one automatyczne zmienianie ustawień NAT z poziomu aplikacji, bez jawnej interwencji użytkownika. Dla użytkownika domowego, uruchamiającego w ten sposób własny serwer WWW, mogą być ponadto niezbędne dodatkowe usługi w rodzaju dynamicznego DNS (o czym piszemy w rozdziale 11.).

7.9. Bibliografia

[ANM09] S. Alcock, R. Nelson, D. Miles, *Investigating the Impact of Service Provider NAT on Residential Broadband Users*, University of Waikato, unpublished technical report, 2009.

[DLNA] *http://www.dlna.org*

[HBA09] D. Hayes, J. But, G. Armitage, *Issues with Network Address Translation for SCTP*, „Computer Communications Review", styczeń 2009.

[IDPCP] D. Wing (red.), S. Cheshire, M. Boucadair, R. Penno, P. Selkirk, *Port Control Protocol (PCP)*, Internet draft-ietf-pcp-base, w przygotowaniu, lipiec 2011.

[IDSNAT] R. Stewart, M. Tuexen, I. Ruengeler, *Stream Control Transmission Protocol (SCTP) Network Address Translation*, Internet draft-ietf-behavesctpnat, w przygotowaniu, czerwiec 2011.

[IDTI] J. Rosenberg, A. Keranen, B. Lowekamp, A. Roach, *TCP Candidates with Interactive Connectivity Establishment (ICE)*, Internet draft-ietf-mmusicice-tcp, w przygotowaniu, wrzesień 2011.

[IGD] UPnP Forum, *Internet Gateway Devices (IGD) Standardized Device Control Protocol V 1.0*, listopad 2001.

[IGD2] UPnP Forum, *IDG:2 Improvements over IGD:1*, marzec 2009.

[ISP] *http://www.iana.org/assignments/stun-parameters*

[MBCB08] O. Maennel, R. Bush, L. Cittadini, S. Bellovin, *A Better Approach to Carrier-Grade-NAT*, Columbia University Technical Report CUCS-041-08, Sept. 2008.

[NFWEB] *http://netfilter.org*

[PJSUA] *http://www.pjsip.org/pjsua.htm*

[RFC0959] J. Postel, J. Reynolds, *File Transfer Protocol*, Internet RFC 0959/STD 0009, październik 1985.

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, *Address Allocation for Private Internets*, Internet RFC 1918BCP 0005, luty 1996.

[RFC1928] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, L. Jones, *SOCKS Protocol Version 5*, Internet RFC 1928, marzec 1996.

[RFC2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, *Hypertext Transfer Protocol — HTTP/1.1*, Internet RFC 2616, czerwiec 1999.

[RFC2637] K. Hamzeh, G. Pall, W. Verthein, J. Taarud, W. Little, G. Zorn, *Point-to-Point Tunneling Protocol (PPTP)*, Internet RFC 2637 (informational), lipiec 1999.

[RFC2766] G. Tsirtsis, P. Srisuresh, *Network Address Translation—Protocol Translation (NAT-PT)*, Internet RFC 2766 (wycofany przez [RFC4966]), luty 2000.

[RFC3022] P. Srisuresh, K. Egevang, *Traditional IP Network Address Translator (Traditional NAT)*, Internet RFC 3022 (informational), styczeń 2001.

[RFC3027] M. Holdrege, P. Srisuresh, *Protocol Complications with the IP Network Address Translator*, Internet RFC 3027 (informational), styczeń 2001.

[RFC3235] D. Senie, *Network Address Translator (NAT)-Friendly Application Design Guidelines*, Internet RFC 3235 (informational), styczeń 2002.

[RFC3264] J. Rosenberg, H. Schulzrinne, *An Offer/Answer Model with Session Description Protocol (SDP)*, Internet RFC 3264, czerwiec 2002.

[RFC3424] L. Daigle (red.), IAB, *IAB Considerations for UNilateral Self-Address Fixing (UNSAF) across Network Address Translation*, Internet RFC 3424 (informational), listopad 2002.

[RFC3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Internet RFC 3550/STD 0064, lipiec 2003.

[RFC3711] M. Baugher, D. McGrew, M. Naslund, E. Carrara, K. Norrman, *The Secure Real-Time Transport Protocol (SRTP)*, Internet RFC 3711, marzec 2004.

[RFC4193] R. Hinden, B. Haberman, *Unique Local IPv6 Unicast Addresses*, Internet RFC 4193, październik 2005.

[RFC4213] E. Nordmark, R. Gilligan, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, Internet RFC 4213, październik 2005.

[RFC4340] E. Kohler, M. Handley, S. Floyd, *Datagram Congestion Control Protocol (DCCP)*, Internet RFC 4340, marzec 2006.

[RFC4605] B. Fenner, H. He, B. Haberman, H. Sandick, *Internet Group Management Protocol (IGMP)/Multicast Listener Discovery (MLD)-Based Multicast Forwarding (IGMP/MLD Proxying)*, Internet RFC 4605, sierpień 2006.

[RFC4787] F. Audet (red.), C. Jennings, *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*, Internet RFC 4787/BCP 0127, styczeń 2007.

[RFC4864] G. Van de Velde, T. Hain, R. Droms, B. Carpenter, E. Klein, *Local Network Protection for IPv6*, Internet RFC 4864 (informational), maj 2007.

[RFC4960] R. Stewart (red.), *Stream Control Transmission Protocol*, Internet RFC 4960, wrzesień 2007.

[RFC4966] C. Aoun, E. Davies, *Reasons to Move the Network Address Translator-Protocol Translator (NAT-PT) to Historic Status*, Internet RFC 4966 (informational), lipiec 2007.

[RFC5128] P. Srisuresh, B. Ford, D. Kegel, *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*, Internet RFC 5128 (informational), marzec 2008.

[RFC5135] D. Wing, T. Eckert, *IP Multicast Requirements for a Network Address Translator (NAT) and a Network Address Port Translator (NAPT)*, Internet RFC 5135/BCP 0135, luty 2008.

[RFC5245] J. Rosenberg, *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols*, Internet RFC 5245, kwiecień 2010.

[RFC5382] S. Guha (red.), K. Biswas, B. Ford, S. Sivakumar, P. Srisuresh, *NAT Behavioral Requirements for TCP*, Internet RFC 5382/BCP 0142, październik 2008.

[RFC5389] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, *Session Traversal Utilities for NAT (STUN)*, Internet RFC 5389, październik 2008.

[RFC5411] J. Rosenberg, *A Hitchhiker's Guide to the Session Initiation Protocol (SIP)*, Internet RFC 5411 (informational), luty 2009.

[RFC5508] P. Srisuresh, B. Ford, S. Sivakumar, S. Guha, *NAT Behavioral Requirements for ICMP*, Internet RFC 5508/BCP 0148, kwiecień 2009.

[RFC5571] B. Storer, C. Pignataro (red.), M. Dos Santos, B. Stevant (red.), L. Toutain, J. Tremblay, *Softwire Hub and Spoke Deployment Framework with Layer Two Tunneling Protocol Version 2 (L2TPv2)*, Internet RFC 5571, czerwiec 2009.

[RFC5596] G. Fairhurst, *Datagram Congestion Control Protocol (DCCP) Simultaneous-Open Technique to Facilitate NAT/Middlebox Traversal*, Internet RFC 5596, wrzesień 2009.

[RFC5597] R. Denis-Courmont, *Network Address Translation (NAT) Behavioral Requirements for the Datagram Congestion Control Protocol*, Internet RFC 5597/BCP 0150, wrzesień 2009.

[RFC5626] C. Jennings, R. Mahy, F. Audet (red.), *Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)*, Internet RFC 5626, październik 2009.

[RFC5761] C. Perkins, M. Westerlund, *Multiplexing RTP Data and Control Packets on a Single Port*, Internet RFC 5761, kwiecień 2010.

[RFC5766] R. Mahy, P. Matthews, J. Rosenberg, *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*, Internet RFC 5766, kwiecień 2010.

[RFC5780] D. MacDonald, B. Lowekamp, NAT *Behavior Discovery Using Session Traversal Utilities for NAT (STUN)*, Internet RFC 5780 (experimental), maj 2010.

[RFC5902] D. Thaler, L. Zhang, G. Lebovitz, *IAB Thoughts on IPv6 Network Address Translation*, Internet RFC 5902 (informational), lipiec 2010.

[RFC5928] M. Petit-Huguenin, *Traversal Using Relays around NAT (TURN) Resolution Mechanism*, Internet RFC 5928, sierpień 2010.

[RFC6052] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, X. Li, *IPv6 Addressing of IPv4/IPv6 Translators*, Internet RFC 6052, październik 2010.

[RFC6062] S. Perreault (red.), J. Rosenberg, *Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations*, Internet RFC 6062, listopad 2010.

[RFC6120] P. Saint-Andre, *Extensible Messaging and Presence Protocol (XMPP): Core*, Internet RFC 6120, marzec 2011.

[RFC6144] F. Baker, X. Li, C. Bao, K. Yin, *Framework for IPv4/IPv6 Translation*, Internet RFC 6144 (informational), kwiecień 2011.

[RFC6145] X. Li, C. Bao, F. Baker, *IP/ICMP Translation Algorithm*, Internet RFC 6145, kwiecień 2011.

[RFC6146] M. Bagnulo, P. Matthews, I. van Beijnum, *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*, Internet RFC 6146, kwiecień 2011.

[RFC6156] G. Camarillo, O. Novo, S. Perreault (red.), *Traversal Using Relays around NAT (TURN) Extension for IPv6*, Internet RFC 6156, kwiecień 2011.

[RFC6296] M. Wasserman, F. Baker, *IPv6-to-IPv6 Network Prefix Translation*, Internet RFC 6296 (experimental), czerwiec 2011.

[RFC6333] A. Durand, R. Droms, J. Woodyatt, Y. Lee, *Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion*, Internet RFC 6333, sierpień 2011.

[RFC6334] D. Hankins, T. Mrugalski, *Dynamic Host Configuration Protocol for IPv6 (DHCPv6) Option for Dual-Stack Lite*, Internet RFC 6334, sierpień 2011.

[UPNP] *http://www.upnp.org*

[UPNPC] *http://miniupnp.free.fr*

[XEP-0176] J. Beda, S. Ludwig, P. Saint-Andre, J. Hildebrand, S. Egan, R. McQueen, *XEP-0176: Jingle ICE-UDP Transport Method*, XMPP Standards Foundation, czerwiec 2009, *http://xmpp.org/extensions/xep-0176.html*

[XIDAD] P. Gauthier, J. Cohen, M. Dunsmuir, C. Perkins, *Web Proxy Auto-Discovery Protocol*, Internet draft-ietf-wrec-wpad-01, w przygotowaniu (expired), czerwiec 1999.

[XIDMU] Y. Goland, *Multicast and Unicast UDP HTTP Messages*, Internet draft-goland-http-udp-01.txt, w przygotowaniu (expired), listopad 1999.

[XIDPMP] S. Cheshire, M. Krochmal, K. Sekar, *NAT Port Mapping Protocol (NAT-PMP)*, Internet draft-cheshire-nat-pmp-03.txt, w przygotowaniu (expired), kwiecień 2008.

[XIDS] Y. Goland, T. Cai, P. Leach, Y. Gu, S. Albright, *Simple Service Discovery Protocol/1.0 Operating without an Arbiter*, Internet draft-cai-ssdp-v1-03.txt, w przygotowaniu (expired), październik 1999.

Rozdział 8.
ICMPv4 i ICMPv6 — Internet Control Message Protocol

8.1. Wprowadzenie

Protokół IP nie udostępnia sam z siebie mechanizmów umożliwiających śledzenie losu datagramów, które — z różnych przyczyn — nie docierają do miejsc swego przeznaczenia. Nie udostępnia on także żadnych narzędzi do uzyskiwania informacji diagnostycznych, np. listy routerów na trasie datagramu czy też szacowanego czasu jego wędrówki. Z zamiarem uzupełnienia tych niedostatków opracowano specjalny protokół o nazwie *Internet Control Message Protocol* — „internetowy protokół komunikatów kontrolnych" — powszechnie oznaczany akronimem ICMP: w dokumentach [RFC0792] i [RFC4443] zdefiniowano szczegóły współdziałania ICMP z protokołem IP w celu uzyskiwania informacji kontrolnej i diagnostycznej związanej z konfiguracją warstwy IP oraz dyspozycją przesyłanych w tej warstwie pakietów. Protokół ICMP często traktowany jest jako część protokołu IP, co nie jest jednak uzasadnione: to prawda, że IP oraz ICMP są ściśle powiązane i od każdej implementacji IP wymaga się także implementowania ICMP, jednakże ponieważ ICMP wykorzystuje do swych celów datagramy IP, plasuje się niejako ponad warstwą IP — ergo można go uważać za protokół pośredni, zlokalizowany na styku warstw sieciowej i transportowej.

ICMP zapewnia generowanie i dostarczanie komunikatów generalnie związanych z sytuacjami, które nie powinny pozostawać niezauważone, bo na ogół wymagają odpowiedniej reakcji: protokoły warstw wyższych (np. TCP i UDP), powierzając swe pakiety (zawodnej z założenia) warstwie IP, muszą dysponować środkami kontrolowania prawidłowości wykonywania zadań powierzanych tej warstwie i takich właśnie środków dostarcza im ICMP. Zauważmy przy tym, że ICMP nie niweluje przy tym wspomnianej zawodności, bo nie wszystkie sytuacje awaryjne kwitowane są jego komunikatami: przykładem takiej sytuacji jest zagubienie datagramu IP wskutek zapełnienia buforów routera — router odrzuca wówczas nadchodzący datagram bez jakiejkolwiek dodatkowej akcji z nim związanej. Protokoły warstw wyższych muszą więc we własnym zakresie implementować mechanizmy niezawodności, a ICMP jedynie dostarcza im środków pomocnych w tym dziele.

Ze względu na istotną rolę ICMP i jego związek z kluczowymi funkcjami systemu i istotną informacją konfiguracyjną, protokół ten stanowi popularny obiekt zainteresowania ze strony hakerów próbujących za pomocą komunikatów ICMP ingerować w działanie systemów komputerowych. W celu ochrony przed tego typu zagrożeniami administratorzy sieci często blokują w firewallach forwardowanie pakietów ICMP, zwłaszcza w routerach brzegowych (patrz [RFC4890]). Zmniejsza to, co prawda, wspomniane zagrożenie, lecz jednocześnie utrudnia (a często wręcz uniemożliwia) poprawne działanie aplikacji użytkowych wykorzystujących ICMP, m.in. `ping` i `traceroute`.

W dalszym ciągu rozdziału akronim ICMP odnosić się będzie do obu jego wersji — ICMPv4 i ICMPv6 — dedykowanych odpowiednim wersjom protokołu IP. Jak się wkrótce przekonamy, ICMPv6 spełnia w stosunku do IPv6 rolę dalece ważniejszą niż ta, jaką ICMPv4 odgrywa w kontekście protokołu IPv4.

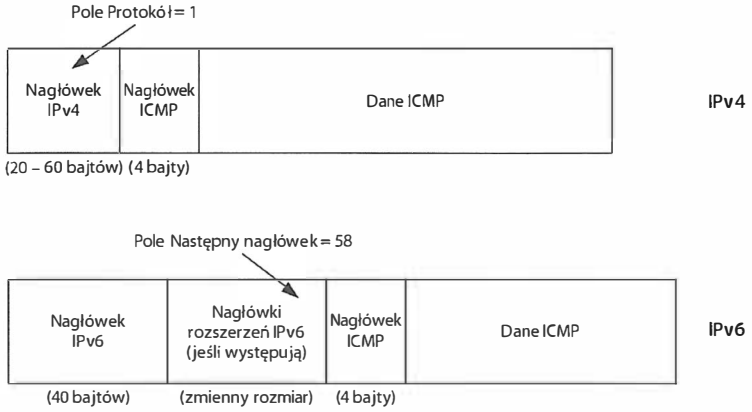
Protokół ICMP zdefiniowano po raz pierwszy w dokumencie [RFC0792], do którego poprawki i wyjaśnienia wnoszą późniejsze dokumenty [RFC1122] i [RFC1812]. Bazowa specyfikacja ICMPv6 znajduje się w dokumencie [RFC4443], natomiast dokument [RFC4884] opisuje koncepcję dodawania obiektów rozszerzeń do komunikatów ICMP — funkcjonalność ta wykorzystywana jest m.in. do przekazywania informacji MPLS (*Multiprotocol Label Switching* — patrz [RFC4950]) oraz do wskazywania interfejsu routera i następnego przeskoku dla trasowanego datagramu (patrz [RFC5837]). W dokumencie [RFC5508] opisano natomiast standardową charakterystykę przetwarzania komunikatów ICMP przez NAT (o czym pisaliśmy również w rozdziale 7.). Funkcjonalność ICMPv6 wykracza poza oryginalne funkcje prostej sygnalizacji błędów, wykorzystywana jest m.in. w ramach protokołu *odnajdywania sąsiadów* (*Neighbor Discovery* — ND, definiowanego w [RFC4861]), stanowiącego funkcjonalny odpowiednik protokołu ARP współdziałającego z IPv4 (patrz rozdział 4.), a także w ramach *odnajdywania routerów* (*Router Discovery* — RD, patrz rozdział 6.) oraz zarządzania adresami multicast (patrz rozdział 9.). Wreszcie, ICMP pełni istotną rolę w zarządzaniu urządzeniami podręcznymi implementującymi mobilne IP.

8.1.1. Enkapsulowanie komunikatów ICMP w datagramach IPv4 i IPv6

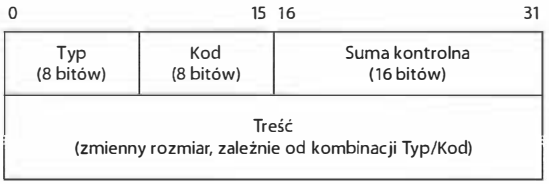
Komunikaty ICMP enkapsulowane są w datagramach IP (obu wersji), jak przedstawiono to na rysunku 8.1.

W datagramie IPv4 wartość 1 w polu *Protokół* identyfikuje ICMPv4; w datagramie IPv6 wartość 58 identyfikująca ICMPv6 znajduje się w polu *Następny nagłówek* ostatniego nagłówka rozszerzeń (lub nagłówka podstawowego, jeśli nagłówki rozszerzeń nie występują). Podobnie jak generalnie wszystkie datagramy IP, także datagramy IP niosące komunikaty ICMP mogą być fragmentowane (patrz rozdział 10.), choć zdarza się to raczej rzadko.

Na rysunku 8.2 przedstawiono format komunikatu ICMP w obu wersjach. Cztery pierwsze bajty mają identyczne znaczenie w obu wersjach, pozostała część komunikatu jest specyficzna dla konkretnej wersji (ICMPv4 albo ICMPv6) oraz konkretnej kombinacji *Typu* i *Kodu*.



Rysunek 8.1. *Enkapsulacja komunikatów ICMP w datagramach IPv4 i IPv6. Suma kontrolna w nagłówku ICMPv4 odejmuje obszar danych, a w wersji ICMPv6 również wybrane pola nagłówka podstawowego IPv6: Adres źródłowy, Adres docelowy i Rozmiar ładunku użytecznego oraz pola Następny nagłówek w ostatnim nagłówku rozszerzeń*



Rysunek 8.2. *Wszystkie komunikaty rozpoczynają się od jednobajtowych pól Typ i Kod, po których następuje dwubajtowa suma kontrolna, uwzględniająca cały komunikat. Znaczenie pól Typ i Kod jest różne w wersjach ICMPv4 i ICMPv6*

W wersji ICMPv4 zarezerwowane są 42 różne wartości pola *Typ* (patrz [ICMPTYPES]), jednak tylko 8 z nich jest w regularnym użyciu i dalej w tym rozdziale pokażemy szczegółowy format każdego z tych typów. Dla danej wartości pola *Typ* mogą występować różne podtypy komunikatów, identyfikowane różnymi wartościami pola *Kod*. Pole *Suma kontrolna* obejmuje w wersji ICMPv4 cały obszar komunikatu (na czas obliczania sumy kontrolnej wartość tego pola przyjmuje się za zerową); w wersji ICMPv6 zakresem *Sumy kontrolnej* objęty jest także *pseudonagłówek*, czyli koncepcyjny obszar stanowiący konkatenację pól *Adres źródłowy*, *Adres docelowy*, *Rozmiar ładunku użytecznego* nagłówka podstawowego oraz pole *Następny nagłówek* ostatniego nagłówka rozszerzeń datagramu IPv6 enkapsulującego komunikat ICMP (zgodnie z ogólną koncepcją pseudonagłówka dla pakietów IPv6, opisaną w sekcji 8.1 dokumentu [RFC2460]). Algorytm obliczania rzeczonej sumy kontrolnej opisany został szczegółowo w punkcie 5.2.2 niniejszej książki. Zauważmy, że po raz pierwszy w tej książce spotykamy się z przypadkiem sumy kontrolnej *całościowej* (*end-to-end*), czyli obejmującej cały obszar konkretnej struktury (komunikatu ICMP). Suma kontrolna komunikatu ICMP pozostaje nienaruszona na całej jego trasie od nadawcy aż do końcowego adresata — pod tym względem różni się ona od sumy kontrolnej nagłówka IPv4, zmieniającej się przy każdym przeskoku. Otrzymany komunikat z niewłaściwą wartością sumy kontrolnej jest zwyczajnie ignorowany bez ostrzeżenia — nie jest generowany komunikat ICMP o odrzuceniu innego

komunikatu ICMP. Zauważmy, że w datagramie IP ładunek użyteczny nie jest chroniony przez sumę kontrolną w nagłówku; ze względu na wyjątkowe znaczenie protokołu ICMP konieczne więc było uczynienie jego integralną częścią własnego mechanizmu kontroli integralności.

8.2. Komunikaty ICMP

Przyjrzymy się teraz bliżej komunikatom ICMP, poświęcając szczególną uwagę wykorzystywanym najczęściej. Generalnie komunikaty ICMP dzielą się na dwie grupy: do pierwszej należą komunikaty oznajmiające problemy z dostarczaniem datagramów IP, zwane popularnie komunikatami o błędach (*error messages*), druga grupa to komunikaty zwane popularnie *informacyjnymi*, wymieniane w ramach dialogów żądanie-odpowiedź, związanych z uzyskiwaniem różnorodnych informacji konfiguracyjnych.

8.2.1. Komunikaty ICMPv4

Do informacyjnych komunikatów ICMPv4 zaliczają się m.in. *Echo Request* (typ 8) i *Echo Reply* (typ 0) oraz dwa komunikaty określane wspólnym mianem *Router Discovery* („odnajdywanie routerów"): *Router Advertisement* (typ 9) i *Router Solicitation* (typ 10). Wśród najczęściej generowanych komunikatów ICMPv4 o błędach wymienić należy natomiast *Destination Unreachable* (typ 3), *Redirect* (typ 5), *Time Exceeded* (typ 11) i *Parameter Problem* (typ 12). Zestawienie standardowych komunikatów ICMPv4 znajduje się w tabeli 8.1.

Tabela 8.1. *Standardowe typy komunikatów ICMPv4. Typy najczęściej występujące wyróżniono gwiazdką, znakiem + opatrzono natomiast typy komunikatów mogących zawierać obiekty rozszerzeń. W kolumnie Rodzaj wskazano rodzaju komunikatu — I oznacza komunikat informacyjny, zaś E komunikat o błędzie*

Typ	Nazwa oficjalna	Dokument	Rodzaj	Znaczenie
0 (*)	*Echo Reply*	[RFC0792]	I	Odpowiedź na żądanie *Echo* (`ping`), zwraca dane
3 (*)(+)	*Destination Unreachable*	[RFC0792]	E	Nieosiągalny host lub protokół
4	*Source Quench*	[RFC0792]	E	Wystąpiło przeciążenie (nieużywany)
5 (*)	*Redirect*	[RFC0792]	E	Konieczność użycia alternatywnego routera
8 (*)	*Echo Request*	[RFC0792]	I	Żądanie *Echo* (`ping`) (opcjonalnie może zawierać dane)
9	*Router Advertisement*	[RFC1256]	I	Wskazanie adresu lub preferencji routera
10	*Router Solicitation*	[RFC1256]	I	Żądanie wysłania komunikatu *Router Advertisement*
11 (*)(+)	*Time Exceeded*	[RFC0792]	E	Wyczerpanie zasobów lub wyzerowanie pola *Czas życia* (TTL)
12 (*)(+)	*Parameter Problem*	[RFC0792]	E	Niepoprawny format pakietu lub jego nagłówka

Dla najczęściej używanych komunikatów (tych oznaczonych gwiazdką w tabeli 8.1) dla danego typu może występować kilka podtypów, rozróżnianych na podstawie zawartości pola *Kod*. Znaczenie tych wartości wyjaśniliśmy w tabeli 8.2; komunikaty niewymienione w tej tabeli mają w polu *Kod* zawsze wartość 0. Komunikaty niektórych typów mogą zawierać dodatkową informację, zgodnie z dokumentem [RFC4884] — w tabeli 8.1 wyróżniliśmy je znakiem +.

Tabela 8.2. *Komunikaty ICMPv4 występujące w kilku odmianach (podtypach) dla danej wartości pola Typ; poszczególne podtypy rozróżniane są na podstawie pola Kod. Najczęściej występujące kombinacje Typ/Kod wyróżniliśmy gwiazdką w kolumnie Typ*

Typ	Kod	Nazwa oficjalna	Znaczenie
3	0	*Net Unreachable*	Nie istnieje żadna trasa do miejsca przeznaczenia
3 (*)	1	*Host Unreachable*	Istniejący, lecz nieosiągalny host
3	2	*Protocol Unreachable*	Nierozpoznany protokół transportowy
3 (*)	3	*Port Unreachable*	Nierozpoznany (nieużywany) port transportowy
3 (*)	4	*Fragmentation Needed and Don't Fragment Was Set* (komunikat PTB)	Wymagana jest fragmentacja pakietu, lecz zabrania jej ustawiony bit DF; komunikat generowany w ramach PMTUD (patrz [RFC1191])
3	5	*Source Route Failed*	Nieosiągalny router pośredni
3	6	*Destination Network Unknown*	Komunikat wycofany przez [RFC1812]
3	7	*Destination Host Unknown*	Docelowy host nie istnieje
3	8	*Source Host Isolated*	Komunikat wycofany przez [RFC1812]
3	9	*Communication with Destination Network Administratively Prohibited*	Komunikat wycofany przez [RFC1812]
3	10	*Communication with Destination Host Administratively Prohibited*	Komunikat wycofany przez [RFC1812]
3	11	*Destination Network Unreachable for Type of Service*	Usługa niedostępna w sieci
3	12	*Destination Host Unreachable for Type of Service*	Usługa nieimplementowana w hoście docelowym
3	13	*Communication Administratively Prohibited*	Komunikacja zabroniona w ramach polityki filtrowania
3	14	*Host Precedence Violation*	Niedozwolone wskazanie preferencji dla danej kombinacji adres źródłowy-adres docelowy, kombinacji port źródłowy-port docelowy lub protokołu warstwy wyższej
3	15	*Precedence Cutoff in Effect*	Zbyt mała wartość *Typu usługi* ([RFC1812])
5	0	*Redirect Datagram for the Network (or Subnet)*	Wskazanie routera alternatywnego

Tabela 8.2. *Komunikaty ICMPv4 występujące w kilku odmianach (podtypach) dla danej wartości pola Typ; poszczególne podtypy rozróżniane są na podstawie pola Kod. Najczęściej występujące kombinacje Typ/Kod wyróżniliśmy gwiazdką w kolumnie Typ — ciąg dalszy*

Typ	Kod	Nazwa oficjalna	Znaczenie
5 (*)	1	*Redirect Datagram for the Host*	Wskazanie routera alternatywnego (host)
5	2	*Redirect Datagram for the Type of Service and Network*	Wskazanie routera alternatywnego (ToS/net)
5	3	*Redirect Datagram for the Type of Service and Host*	Wskazanie routera alternatywnego (ToS/host)
9	0	*Normal Router Advertisement*	Adres routera i informacja o jego konfiguracji
9	16	*Does Not Route Common Traffic*	Router mobilnego IP (patrz [RFC5944]) nie obsługuje „zwykłych" pakietów
11 (*)	0	*Time to Live Exceeded in Transit*	Wyzerowanie pola *Czas życia* w nagłówku IPv4
11	1	*Fragment Reassembly Time Exceeded*	W założonym interwale czasowym dla fragmentacji nie nadeszły niektóre fragmenty pakietu
12 (*)	0	*Pointer Indicates the Error*	Wskaźnik (offset w bajtach) pierwszego problematycznego pola
12	1	*Missing a Required Option*	Komunikat nieużywany
12	2	*Bad Length*	Niepoprawna wartość w polu *Całkowity rozmiar* nagłówka IPv4

Oficjalna lista typów i kodów komunikatów ICMP kontrolowana jest przez IANA (patrz [ICMPTYPES]). Wiele z typów komunikatów zdefiniowanych zostało w pierwszej specyfikacji [RFC0792] z roku 1981, jednak późniejsze doświadczenia z różnymi protokołami (np. z DHCP) kazały tę listę zrewidować, m.in. przez wycofanie niektórych typów z użytku. Projektowanie protokołów IPv6 i ICMPv6 przebiegało już na bazie znacznie bogatszej wiedzy, toteż na gruncie ICMPv6 daje się zauważyć znacząco większą spójność definicji w porównaniu z ICMPv4.

8.2.2. Komunikaty ICMPv6

Zdefiniowane typy komunikatów ICMPv6 przedstawiamy w tabeli 8.3. Zauważmy, że oprócz komunikatów o błędach i komunikatów informacyjnych pojawiają się tu także komunikaty związane z konfigurowaniem hostów i routerów.

Podobnie jak w ICMPv4, komunikaty ICMPv6 także dzielą się na komunikaty o błędach i komunikaty informacyjne, tym razem jednak każdej z tych grup przydzielono ciągły zakres numeracji pola *Typ*: wartości od 0 do 127 oznaczają komunikaty o błędach, wartości powyżej 127 — komunikaty informacyjne. Jak łatwo zauważyć, wiele z komunikatów informacyjnych skojarzonych jest w pary żądanie-odpowiedź.

Tabela 8.3. *Wśród komunikatów ICMPv6 komunikaty o Typie z zakresu 0 – 127 to komunikaty o błędach, Typ komunikatów informacyjnych jest zawsze większy od 127. Znakiem + wyróżniliśmy komunikaty mogące zawierać dodatkową informację. W tabeli nie uwzględniliśmy typów niezdefiniowanych, eksperymentalnych i wycofanych z użytku*

Typ	Nazwa oficjalna	Dokument	Znaczenie
1 (+)	*Destination Unreachable*	[RFC4443]	Nieosiągalny host lub port albo niezdefiniowany protokół
2	*Packet Too Big* (PTB)	[RFC4443]	Wymagana fragmentacja
3 (+)	*Time Exceeded*	[RFC4443]	Wyzerowanie pola *Limit przeskoków* lub wyczerpanie czasu przeznaczonego na reasemblację pakietu
4	*Parameter Problem*	[RFC4443]	Niepoprawny format pakietu lub jego nagłówka
100, 101	*Reserved for private experimentation*	[RFC4443]	Zarezerwowane dla celów eksperymentalnych
127	*Reserved for expansion of ICMPv6 error messages*	[RFC4443]	Zarezerwowane na potrzeby definiowania nowych komunikatów o błędach
128	*Echo Request*	[RFC4443]	Żądanie `ping`, może zawierać dane
129	*Echo Reply*	[RFC4443]	Odpowiedź na żądanie `ping`, zawiera dane
130	*Multicast Listener Query*	[RFC2710]	Żądania uczestników grup multicast (v1)
131	*Multicast Listener Report*	[RFC2710]	Raporty uczestników grup multicast (v1)
132	*Multicast Listener Done*	[RFC2710]	Komunikaty uczestników grup multicast (v1)
133	*Router Solicitation* (RS)	[RFC4861]	RS w ramach mobilnego IPv6
134	*Router Advertisement* (RA)	[RFC4861]	RA w ramach mobilnego IPv6
135	*Neighbor Solicitation* (NS)	[RFC4861]	Odnajdywanie sąsiadów w IPv6 (indagowanie)
136	*Neighbor Advertisement* (NA)	[RFC4861]	Odnajdywanie sąsiadów w IPv6 (ogłaszanie)
137	*Redirect Message*	[RFC4861]	Przekierowanie komunikatu do alternatywnego routera
141	*Inverse Neighbor Discovery Solicitation Message*	[RFC3122]	Żądanie odwrotnego odnajdywania sąsiadów (*Inverse Neighbor Discovery*) — uzyskiwanie adresów IPv6 na podstawie adresu warstwy łącza danych
142	*Inverse Neighbor Discovery Advertisement Message*	[RFC3122]	Odpowiedź odwrotnego odnajdywania sąsiadów (*Inverse Neighbor Discovery*) — zwraca adresy IPv6 skojarzone z określonym adresem warstwy łącza danych
143	*Version 2 Multicast Listener Report*	[RFC3810]	Raporty uczestników grup multicast (v2)
144	*Home Agent Address Discovery Request Message*	[RFC6275]	Żądanie adresu agenta domowego w mobilnym IPv6, wysyłane przez węzeł mobilny

Tabela 8.3. *Wśród komunikatów ICMPv6 komunikaty o Typie z zakresu 0 – 127 to komunikaty o błędach, Typ komunikatów informacyjnych jest zawsze większy od 127. Znakiem + wyróżniliśmy komunikaty mogące zawierać dodatkową informację. W tabeli nie uwzględniliśmy typów niezdefiniowanych, eksperymentalnych i wycofanych z użytku — ciąg dalszy*

Typ	Nazwa oficjalna	Dokument	Znaczenie
145	*Home Agent Address Discovery Reply Message*	[RFC6275]	Udostępnienie adresu agenta domowego w mobilnym IPv6
146	*Mobile Prefix Solicitation*	[RFC6275]	Żądanie prefiksu mobilnego przez węzeł mobilny
147	*Mobile Prefix Advertisement*	[RFC6275]	Oferowanie prefiksu mobilnego przez agenta domowego
148	*Certification Path Solicitation Message*	[RFC3971]	Żądanie bezpiecznego odnajdywania sąsiadów (SEND — *Secure Neighbor Discovery*) dotyczące ścieżki certyfikacji
149	*Certification Path Advertisement Message*	[RFC3971]	Odpowiedź SEND na żądanie ścieżki certyfikacji
151	*Multicast Router Advertisement*	[RFC4286]	Udostępnienie adresu routera multicast
152	*Multicast Router Solicitation*	[RFC4286]	Żądanie adresu routera multicast
153	*Multicast Router Termination*	[RFC4286]	Zakończenie korzystania z routera multicast
154	*FMIPv6 Messages*	[RFC5568]	Komunikaty związane z szybkimi urządzeniami podręcznymi w ramach mobilnego IPv6
200, 201	*Reserved for private experimentation*	[RFC4443]	Zarezerwowane do celów eksperymentalnych
255	*Reserved for expansion of ICMPv6 informational messages*	[RFC4443]	Zarezerwowane na potrzeby definiowania nowych komunikatów informacyjnych

Wnikliwe porównanie standardowych komunikatów ICMPv4 ze standardowymi komunikatami ICMPv6 pozwala dostrzec efekt starań zmierzających do usunięcia ze specyfikacji komunikatów nieprzydatnych i nieużywanych. Podobnie jak w przypadku ICMPv4, także komunikaty ICMPv6 wykorzystują pole *Kod*, głównie do różnicowania podtypów w ramach określonego *Typu*. W tabeli 8.4 zestawiamy znaczenie poszczególnych podtypów dla komunikatów *Destination Unreachable*, *Time Exceeded* i *Parameter Problem* — pozostałe typy komunikatów zawsze mają wartość 0 w polu *Kod*.

Oprócz pól *Typ* i *Kod* definiujących podstawowe znaczenie komunikatów ICMPv6, komunikatom tym towarzyszyć mogą rozmaite opcje (nie było ich w ICMPv4), niektóre z nich są obowiązkowe. Obecnie opcje takie zdefiniowane są jedynie dla komunikatów ND (typy 135 i 136) — ich formaty opisane są szczegółowo dokumencie [RFC4861], zajmiemy się nimi szczegółowo w podrozdziale 8.5.

Tabela 8.4. *Podtypy standardowych typów komunikatów ICMPv6*

Typ	Kod	Nazwa oficjalna	Znaczenie
1	0	*No Route to Destination*	Brak trasy
1	1	*Administratively Prohibited*	Zakaz administracyjny (realizowany np. przez firewall)
1	2	*Beyond Scope of Source Address*	Miejsce przeznaczenia poza zasięgiem adresu źródłowego
1	3	*Address Unreachable*	Sytuacja inna niż określona przez *Kod* 0, 1, lub 2
1	4	*Port Unreachable*	Brak nasłuchiwania na porcie transportowym
1	5	*Source Address Failed Policy*	Naruszenie zasad polityki dotyczącej pakietów przychodzących lub wychodzących
1	6	*Reject Route to Destination*	Odrzucenie trasy do miejsca przeznaczenia
3	0	*Hop Limit Exceeded in Transit*	Dekrementacja pola *Limit przeskoków* spowodowała jego wyzerowanie
3	1	*Reassembly Time Exceeded*	Nie ukończono reasemblacji w założonym interwale czasowym
4	0	*Erroneous Header Field Found*	Ogólny błąd przetwarzania nagłówka
4	1	*Unrecognized Next Header*	Nierozpoznana wartość w polu *Następny nagłówek*
4	2	*Unrecognized IPv6 Option*	Niepoprawna opcja „skok za skokiem" lub opcja docelowa

8.2.3. Przetwarzanie komunikatów ICMP

Przetwarzanie nadchodzących komunikatów ICMP jest zróżnicowane w różnych systemach. Ogólnie rzecz biorąc, komunikatami informacyjnymi automatycznie zarządza system operacyjny, natomiast komunikaty o błędach kierowane są do procesów użytkowników lub do protokołów warstwy wyższej, np. TCP ([RFC5461]) — sposób postępowania z otrzymanym komunikatem spoczywa całkowicie w gestii adresata. Wyjątkiem są jednak komunikaty *Redirect* oraz *Destination Unreachable — Fragmentation Required*: pierwszy z nich powoduje automatyczne uaktualnienie tablicy trasowania hosta, drugi natomiast wykorzystywany jest przez mechanizm rozpoznawania MTU ścieżki (PMTUD), implementowany zazwyczaj przez protokół transportowy (np. TCP). W ICMPv6 przetwarzanie komunikatów poddane zostało pewnym dodatkowym rygorom, zdefiniowanym w [RFC4443]. Oto one.

1. Komunikat ICMPv6 o nierozpoznanych wartościach w polach *Typ*/*Kod* musi zostać — o ile to możliwe — przekazany do procesu warstwy wyższej, który utworzył datagram będący przyczyną błędu.
2. Nierozpoznane komunikaty informacyjne ICMPv6 są ignorowane.
3. Komunikat ICMPv6 o błędzie zawiera kopię jak największej części przyczynowego datagramu — ograniczeniem jest jedynie maksymalna wielkość datagramu IP niosącego komunikat, równa 1280 bajtom, czyli minimalnej wartości MTU dla IPv6.

4. W czasie przetwarzania komunikatu ICMPv6 o błędzie informacja na temat protokołu warstwy wyższej (niezbędna do zidentyfikowania procesu docelowego) uzyskiwana jest na podstawie kopii przyczynowego datagramu, o której mowa w punkcie 3. Jeśli zidentyfikowanie protokołu warstwy wyższej okazuje się niemożliwe, komunikat ICMPv6 zostaje odrzucony (bez ostrzeżenia) przez warstwę IPv6.
5. Niektóre komunikaty o błędach przetwarzane są z uwzględnieniem dodatkowych wymogów, opisanych w podrozdziale 8.3.
6. Tempo generowania komunikatów ICMPv6 przez dany węzeł ograniczane jest przy użyciu rozmaitych mechanizmów, m.in. *zasobnika tokenów* opisywanego w podrozdziale 8.3.

8.3. Komunikaty ICMP o błędach

Rozróżnienie między komunikatami informacyjnymi ICMP a komunikatami o błędach ICMP jest istotne chociażby z tego względu, że generowanie komunikatów o błędach podlega pewnym ograniczeniom, określonym przez [RFC1812] dla ICMPv4 oraz przez [RFC4443] dla ICMPv6. Ogólnie mówiąc, komunikaty ICMP o błędach nie mogą być generowane w związku z następującymi przyczynami:

- innymi komunikatami ICMP,
- datagramami IP z błędnymi nagłówkami (np. niewłaściwą sumą kontrolną),
- datagramami rozgłoszeniowymi lub multicastingowymi w warstwie IP,
- datagramami enkapsulowanymi w ramkach rozgłoszeniowych lub multicastingowych warstwy łącza danych,
- datagramami z niepoprawnym lub zerowym adresem źródłowym,
- fragmentami datagramu różnymi od pierwszego fragmentu.

Powodem narzucenia powyższych ograniczeń jest zamiar uniknięcia tzw. burzy rozgłoszeniowej (*broadcast storm*) — sytuacji, w której niewielka liczba pierwotnych komunikatów powoduje kaskadowe namnażanie kolejnych (poprzez generowanie — w nieskończoność — komunikatów o błędnych komunikatach o błędach). Wspomniane ograniczenia skonkretyzowane zostały ma gruncie obu wersji ICMP — i tak w wersji ICMPv4 komunikaty o błędach nigdy nie są generowane w odniesieniu do:

- komunikatu ICMPv4 o błędzie (mogą być jednak generowane w odpowiedzi na informacyjne żądanie ICMPv4),
- datagramu IPv4 kierowanego na adres rozgłoszeniowy lub adres multicast (czyli adresy dawnej klasy D — patrz punkt 2.3.1),
- datagramu rozsyłanego jako rozgłoszenie w warstwie łącza danych,
- fragmentu innego niż pierwszy fragment datagramu,
- datagramu, którego adres źródłowy nie identyfikuje konkretnego hosta, czyli jest adresem zerowym, adresem pętli zwrotnej, adresem rozgłoszeniowym lub adresem multicast.

W ICMPv6 obowiązują podobne ograniczenia — komunikat ICMPv6 o błędzie nigdy nie jest generowany w odpowiedzi na:

- komunikat ICMPv6 o błędzie,
- komunikat IMCPV6 *Redirect*,
- pakiet kierowany na adres multicast IPv6, z dwoma wyjątkami, takimi jak:
  - komunikat *Packet Too Big* (PTB) (typ 2),
  - komunikat *Parameter Problem — Unrecognized IPv6 Option* (typ 4, kod 2),
- pakiet kierowany na adres multicast lub broadcast w warstwie łącza danych, z powyższymi wyjątkami,
- pakiet, którego adres źródłowy nie identyfikuje konkretnego węzła, czyli jest adresem nieokreślonym, adresem multicast IPv6 lub innym adresem używanym przez nadawcę w roli adresu anycast.

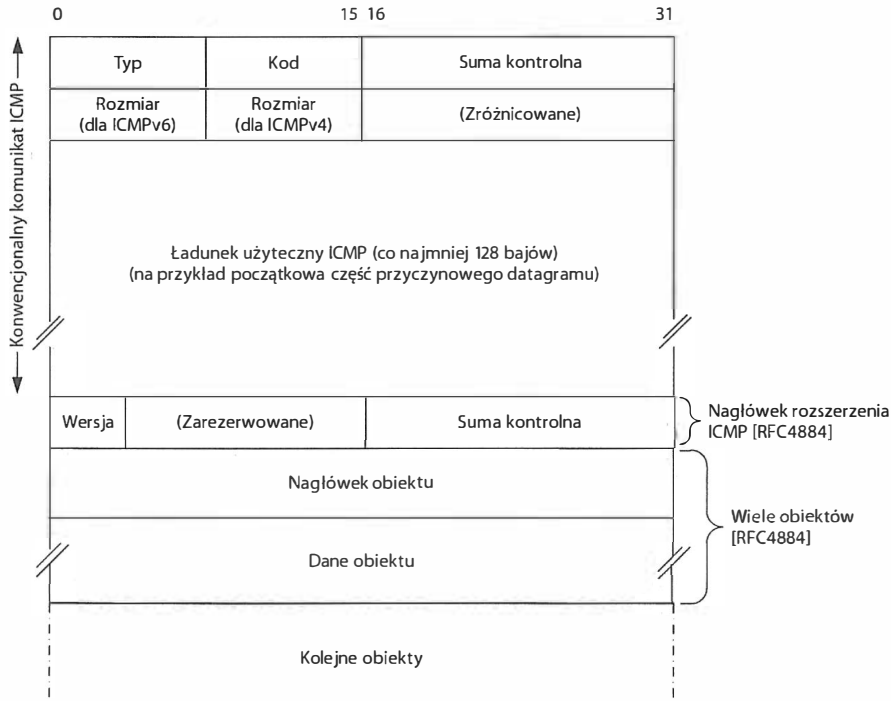
Jak wspominaliśmy w punkcie 6. listy z punktu 8.2.3, dodatkowym ograniczeniem narzuconym na funkcjonowanie protokołu ICMP jest ograniczenie tempa, w jakim dany węzeł może generować komunikaty ICMP — czyli ograniczenie wkładu, jaki pojedynczy nadawca wnosi do generowanego w sieci ruchu. W dokumencie [RFC4443] opisano jedną z realizacji tego zadania — koncepcję *zasobnika tokenów* (*token bucket*). Ów zasobnik zawiera zbiór tokenów, z których każdy reprezentuje zezwolenie na wysłanie N komunikatów ICMP, maksymalna liczba tokenów w zasobniku wynosi B. Wysyłanie komunikatów ICMP powoduje ubywanie tokenów (jednego na każde N komunikatów), jednocześnie zasobnik zasilany jest periodycznie nowymi tokenami. W dłuższym okresie czasu tempo wysyłania komunikatów ICMP przez węzeł limitowane jest szybkością zasilania zasobnika. Opisany mechanizm można więc scharakteryzować przez parę parametrów (B, N) — dla węzłów o małym i średnim natężeniu ruchu wspomniany dokument zaleca wartości (10, 10). Notabene mechanizm ten wykorzystywany jest przez wiele innych protokołów, choć wartości parametrów B i N wyrażają się niekiedy liczbą bajtów, a nie jednostek protokołu.

Komunikat ICMP o błędzie zawiera zawsze kopię nagłówka przyczynowego datagramu IP (wraz z ewentualnymi opcjami) oraz opcjonalnie wybrane fragmenty jego ładunku użytecznego; rozmiar datagramu IP enkapsulującego komunikat ICMP nie przekracza przy tym wartości gwarantującej przesłanie go bez fragmentowania, czyli 576 bajtów dla IPv4 i 1280 bajtów (minimalnego MTU) dla IPv6. Obecność pierwszych 8 bajtów początkowej części ładunku użytecznego przyczynowego datagramu umożliwia protokołowi ICMP powiązanie owego datagramu z konkretnym protokołem warstwy wyższej i odnośnym procesem użytkowym — przykładowo w pakietach TCP i UDP numer portu (identyfikujący wspomniany proces) znajduje się w pierwszych 8 bajtach (patrz rozdziały 10. i 12.). Zgodnie z oryginalną specyfikacją [RFC0792] do komunikatu ICMPv4 włączane jest zawsze pierwsze 8 bajtów ładunku użytecznego przyczynowego datagramu, co jednak okazuje się niewystarczające w przypadku bardziej skomplikowanego nakładania warstw (np. tunelowania datagramów IP w datagramach IP), dlatego specyfikacja [RFC1812] podwyższa tę wartość do poziomu dopuszczalnego przez wymienione wcześniej limity 576 i 1280 bajtów. Niezależnie od tego, w przypadku niektórych typów komunikatów dołączana jest dodatkowa informacja, mająca postać *danych*

rozszerzających (*extended data structure*). Rozpatrzymy pokrótce format i znaczenie tej informacji, po czym przeanalizujemy szczegółowo kilka najważniejszych typów komunikatów ICMP.

8.3.1. Rozszerzenia ICMP i komunikaty wieloczęściowe

W dokumencie [RFC4884] określono metodę zwiększania użyteczności komunikatów ICMP za pomocą dołączanych do nich *danych rozszerzających* (*extended data structure*). Dane te rozpoczynają się stosownym nagłówkiem, po którym następuje obiekt zawierający różnorodne informacje — co przedstawiamy na rysunku 8.3.



Rysunek 8.3. *Rozszerzony komunikat ICMPv4 lub ICMPv6 zawiera nagłówek rozszerzenia i ciąg (być może zerowy) załączonych obiektów. Każdy z obiektów rozpoczyna się nagłówkiem stałej długości, po którym następuje obszar danych zmiennej długości. Ze względu na kompatybilność z oryginalną specyfikacją ICMP, ładunek użyteczny komunikatu nie może być krótszy niż 128 bajtów*

W przedstawionym formacie został jawnie wskazany rozmiar ładunku użytecznego, czyli załączonej części przyczynowego datagramu: w ICMPv4 pole określające ów rozmiar znajduje się w szóstym bajcie i wyraża ten rozmiar w jednostkach 4-bajtowych, w ICMPv6 pole to znajduje się w piątym bajcie i wyraża wspomniany rozmiar w jednostkach 8-bajtowych (w oryginalnej definicji ICMP oba te bajty były wyzerowane i zarezerwowane do przyszłego użytku). Jeśli trzeba, wspomniany ładunek użyteczny dopełniany jest bajtami zerowymi do żądanego rozmiaru: rozmiar ten musi stanowić wielokrotność

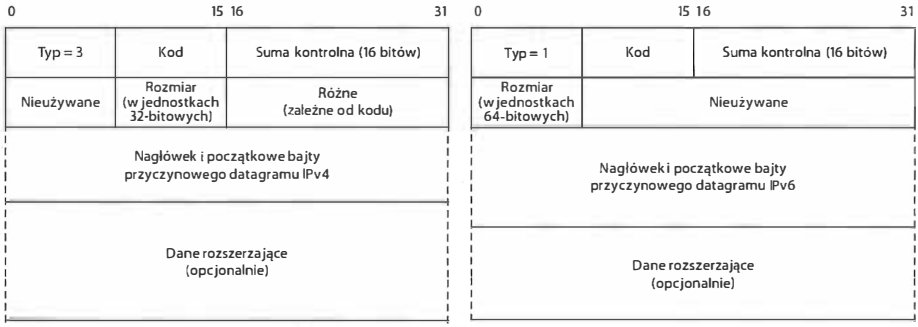
jednostki, w której jest wyrażany (czyli 4 bajtów w ICMPv4 i 8 bajtów w ICMPv6), ponadto jeśli wykorzystywane jest rozszerzenie komunikatu, rozmiar ten nie może być mniejszy niż 128 bajtów.

Dane rozszerzające mogą występować w komunikatach ICMPv4 *Destination Unreachable*, *Time Exceeded* i *Parameter Problem* oraz w komunikatach ICMPv6 *Destination Unreachable* i *Time Exceeded*. Przyjrzyjmy się dokładniej strukturze tych komunikatów.

8.3.2. Komunikat Destination Unreachable (typ 3 w ICMPv4, typ 1 w ICMPv6)

Komunikat typu *Destination Unreachable* wskazuje na niemożność dostarczenia datagramu do miejsca jego przeznaczenia bądź to z powodu problemów z jego transmisją, bądź też z braku adresata zainteresowanego jego odebraniem. W ICMPv4 jest to komunikat typu 3; mimo iż zdefiniowano 16 różnych jego podtypów, w praktycznym użyciu są tylko 4, oznaczające nieosiągalność hosta (kod 1), niedostępność portu (kod 3), konieczność pofragmentowania pakietu i jednocześnie brak zezwolenia na fragmentację (kod 4) oraz administracyjny zakaz komunikacji (kod 13). W ICMPv6 jest to komunikat typu 1 z 7 podtypami. W porównaniu z ICMPv4 sytuację koniecznej, acz zabronionej fragmentacji powierzono innemu komunikatowi (PTB — *Packet Too Big* — typ 2), który jednak omawiamy w tym miejscu ze względu na podobne znaczenie. Użyjemy przy tym uproszczonej terminologii, oznaczając skrótem PTB zarówno komunikat ICMPv4 o typie 3 i kodzie 4, jak i komunikat ICMPv6 o typie 2 i kodzie 0.

Formaty komunikatu *Destination Unreachable* w obu wersjach ICMP pokazano na rysunku 8.4. W polu *Typ* znajduje się wartość 3 w wersji ICMPv4 i 1 w wersji ICMPv6. Wartość pola *Kod* identyfikuje dokładniej przyczynę błędu lub obiekt kryjący się za tą przyczyną. Przeanalizujmy szczegółowo wybrane podtypy tego komunikatu.



Rysunek 8.4. *Komunikat Destination Unreachable w wersji ICMPv4 (z lewej) i ICMPv6 (z prawej). Pole Rozmiar, zgodnie z [RFC4884], wyraża rozmiar oryginalnego datagramu w słowach 32-bitowych (w IPv4) lub 64-bitowych (w IPv6). Pole Różne zawiera wartość MTU dla następnego przeskoku, gdy Kod = 4, i wykorzystywane jest przez procedurę PMTUD. Protokół ICMPv6 używa do tego celu innego komunikatu (Typ = 2)*

8.3.2.1. Komunikaty ICMPv4 Host Unreachable (kod 1) i ICMPv6 Address Unreachable (kod 3)

Komunikat tego podtypu generowany jest przez router lub host w sytuacji, gdy operacja dostarczania bezpośredniego (patrz punkt 5.4.3) nie może zostać wykonana z powodu nieosiągalności hosta docelowego. Może się tak zdarzyć np. wtedy, gdy żądanie ARP kierowane jest do hosta nieistniejącego lub wyłączonego (opisywaliśmy tę sytuację w rozdziale 4.) albo wykrycia „nieodpowiadających" hostów przez mechanizm odnajdywania sąsiadów (*Neighbor Discovery* — patrz podrozdział 8.5).

8.3.2.2. Komunikat ICMPv6 No Route to Destination (Kod 0)

Komunikat generowany jest wtedy, gdy przedmiotowy datagram IP uczestniczy w operacji dostarczania pośredniego, lecz nie jest możliwe wyznaczenie trasy dla tej operacji, czyli wskazanie routera stanowiącego następny przeskok. Jak wyjaśnialiśmy w rozdziale 5., każdy router IP musi w swych tablicach trasowania zawierać pozycje umożliwiające forwardowanie dowolnego poprawnego datagramu.

8.3.2.3. Komunikaty ICMPv4 Communication Administratively Prohibited (kod 13) i ICMPv6 Communication with Destination Administratively Prohibited (kod 1)

Ten typ komunikatu (w obu wersjach ICMP) wskazuje na niemożność dostarczenia datagramu IP w konsekwencji *decyzji administracyjnych*, urzeczywistnionych np. w konfiguracji firewalla (patrz rozdział 7.) jawnie odrzucającego pakiety spełniające określone kryteria. Odrzucanie takie niekoniecznie jednak musi być kwitowane w taki sposób — możliwe jest skonfigurowanie firewalla tak, by odrzucał wspomniane pakiety bez ostrzeżenia bądź generował komunikat ICMP innego typu.

8.3.2.4. Komunikaty ICMPv4 Port Unreachable (kod 3) i ICMPv6 Port Unreachable (kod 4)

Komunikat stanowi konsekwencję otrzymania przez host datagramu IP, wskazującego w nagłówku port docelowy nieprzypisany do żadnej aplikacji realizowanej przez host (lub przypisany do aplikacji nieprzygotowanej aktualnie na przyjęcie datagramu). Sytuacja taka występuje zwykle w kontekście protokołu UDP (patrz rozdział 10.), którego datagram posiada w nagłówku numer portu niepowiązany z żadnym z procesów aktualnie realizowanych przez host.

W charakterze ilustracji działania protokołu ICMP sprowokujemy wygenerowanie komunikatu ICMPv4 *Port Unreachable* (typ 3, kod 3) za pomocą protokołu TFTP (*Trivial File Transfer Protocol* — prosty protokół transferu plików, patrz [RFC1350]), standardowo używającego „dobrze znanego" portu 69. Tak się składa, że mimo dostępności tego protokołu w każdym niemal systemie operacyjnym, w większości systemów nie jest uruchamiany proces realizujący jego usługę. W przykładzie prezentowanym na listingu 8.1 host zarządzany systemem Windows wysyła żądanie pobrania pliku (oczywiście, za pośrednictwem TFTP) z linuksowego hosta, na którym *nie uruchomiono serwera TFTP*

i na którym działa program tcpdump rejestrujący zdarzenia zachodzące w kontekście protokołów TCP/IP. Opcja -s programu tcpdump powoduje formowanie pakietów o wskazanym rozmiarze (1500 bajtów), opcja -i wskazuje interfejs, którego dotyczyć ma śledzenie (eth1), zaś użycie opcji -vv skutkować ma generowaniem komunikatów w bardziej opisowej formie. Fraza icmp or port tftp służy ukierunkowaniu śledzenia zarówno na port TFTP (69), jak i na generowane komunikaty ICMP.

Listing 8.1. *Ilustracja niedostępności portu i limitowania tempa generowania komunikatów ICMP*

```
C:\> tftp 10.0.0.1 get /foo        próba pobrania pliku "/foo" z serwera 10.0.0.1
Timeout occurred                   przeterminowanie po 9 sekundach

Linux# tcpdump -s 1500 -i eth1 -vv icmp or port tftp

1 09:45:48.974812 IP (tos 0x0, ttl 128, id 9914, offset 0,
                flags [none], length: 44)

                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

2 09:45:48.974812 IP (tos 0xc0, ttl 255, id 43734, offset 0, flags
                [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9914, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii

3 09:45:49.014812 IP (tos 0x0, ttl 128, id 9915, offset 0,
                flags [none], length: 44)

                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

4 09:45:49.014812 IP (tos 0xc0, ttl 255, id 43735, offset 0, flags
                [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9915, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii

5 09:45:49.014812 IP (tos 0x0, ttl 128, id 9916, offset 0,
                flags [none], length: 44)

                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

6 09:45:49.014812 IP (tos 0xc0, ttl 255, id 43736, offset 0, flags
                [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9916, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii
```

```
7 09:45:49.024812 IP (tos 0x0, ttl 128, id 9917, offset 0,
                flags [none], length: 44)

                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

8 09:45:49.024812 IP (tos 0xc0, ttl 255, id 43737, offset 0,
                flags [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9917, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii

9 09:45:49.024812 IP (tos 0x0, ttl 128, id 9918, offset 0,
                flags [none], length: 44)

                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

10 09:45:49.024812 IP (tos 0xc0, ttl 255, id 43738, offset 0,
                flags [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9918, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii

11 09:45:49.034812 IP (tos 0x0, ttl 128, id 9919, offset 0,
                flags [none], length: 44)
                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

12 09:45:49.034812 IP (tos 0xc0, ttl 255, id 43739, offset 0,
                flags [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
                  10.0.0.1 udp port tftp unreachable
                  for IP (tos 0x0, ttl 128, id 9919, offset 0,
                  flags [none], length: 44)
                     10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                     RRQ "/foo" netascii

13 09:45:49.034812 IP (tos 0x0, ttl 128, id 9920, offset 0,
                flags [none], length: 44)
                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

14 09:45:57.054812 IP (tos 0x0, ttl 128, id 22856, offset 0,
                flags [none], length: 44)
                10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                RRQ "/foo" netascii

15 09:45:57.054812 IP (tos 0xc0, ttl 255, id 43740, offset 0,
                flags [none], length: 72)
                10.0.0.1 > 10.0.0.54: icmp 52:
```

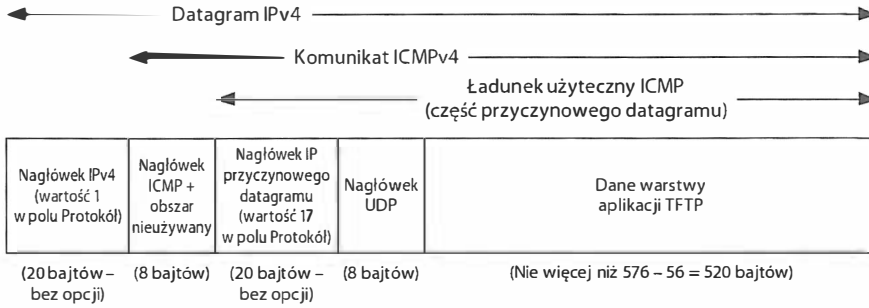
```
                    10.0.0.1 udp port tftp unreachable
                    for IP (tos 0x0, ttl 128, id 22856, offset 0,
                    flags [none], length: 44)
                       10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok] 16
                       RRQ "/foo" netascii

16 09:45:57.064812 IP (tos 0x0, ttl 128, id 22906, offset 0,
                  flags [none], length: 51)
                  10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok]
                       23 ERROR EUNDEF timeout on receive"

17 09:45:57.064812 IP (tos 0xc0, ttl 255, id 43741, offset 0,
                  flags [none], length: 79)
                  10.0.0.1 > 10.0.0.54: icmp 59:
                    10.0.0.1 udp port tftp unreachable
                    for IP (tos 0x0, ttl 128, id 22906, offset 0,
                    flags [none], length: 51)
                       10.0.0.54.3871 > 10.0.0.1.tftp: [udp sum ok]
                            23 ERROR EUNDEF timeout on receive"
```

Na powyższym listingu widzimy efekt kilku bezskutecznych prób pobrania (przez klienta) pliku `/foo` z serwera TFTP. Klient formułuje pierwsze żądanie i prawie natychmiast otrzymuje komunikat ICMPv4 *Port Unreachable*; niezrażony tym faktem sześciokrotnie ponawia próbę pobrania pliku, za każdym razem z takim samym skutkiem. Po szóstej próbie odczekuje 8 sekund (za chwilę wyjaśnimy, dlaczego), po czym ponawia próbę jeszcze raz i po otrzymaniu komunikatu ICMP ostatecznie się poddaje.

Zauważmy, że komunikaty ICMP wysyłane są bez wskazywania jakiegokolwiek numeru portu, natomiast każdy 16-bajtowy pakiet żądania klienta ma swe źródło w porcie 3871 i kierowany jest do portu TFTP 69. Wartość 16 na końcu każdego wiersza RRQ (od *Read Request* — odczytane żądanie) stanowi sumaryczną długość pakietu, na którą składają się 2 bajty kodu operacji, 4-znakowa nazwa `/foo`, zerowy ogranicznik tej nazwy, 8-znakowy łańcuch `netascii` i jego zerowy ogranicznik. Format kompletnego komunikatu ICMPv4 *Destination Unreachable* widoczny jest na rysunku 8.5; komunikat ten składa się z 52 bajtów (nie licząc 4-bajtowego nagłówka IPv4): 4 początkowe bajty zajmuje podstawowy nagłówek ICMPv4, 4 następne bajty są niewykorzystywane (zgodnie z [RFC4884]), kolejne 20 stanowi kopię nagłówka przyczynowego datagramu IPv4, kolejne 8 bajtów to nagłówek UDP, całość kończy 16-bajtowa kopia oryginalnego żądania TFTP (4 + 4 + 20 + 8 + 16 = 52).

Jak wcześniej wyjaśnialiśmy, na podstawie kopii nagłówka przyczynowego datagramu IP protokół ICMP określa sposób interpretacji ładunku użytecznego enkapsulowanego w tym datagramie — w naszym przykładzie wartość 17 w polu *Protokół* oznacza protokół UDP. Z nagłówka UDP protokół ICMP odczytuje numery portów źródłowego (69) i docelowego (3871); ten ostatni pomocny jest, dla hosta będącego adresatem komunikatu ICMP, w skojarzeniu tego komunikatu z konkretnym procesem, choć — prawdę mówiąc — w naszym przypadku proces ten niezbyt się przejmuje otrzymywanymi komunikatami.

Czytelnik studiujący listing 8.1 z pewnością zauważy zagadkowy brak komunikatu ICMP po pakiecie 13. zawierającym kolejne żądanie klienta — właśnie brak rzeczonego komunikatu ICMP, czyli *brak reakcji ze strony serwera*, powoduje, że klient odczekuje



Rysunek 8.5. *Komunikat ICMPv4 Destination Unreachable — Port Unreachable zawiera kopię części przyczynowego datagramu IP; ograniczenie na rozmiar tej kopii wynika bezpośrednio z ograniczenia rozmiaru wynikowego datagramu ICMP/IP (576 bajtów). W tym przypadku rozmiar żądania TFTP jest na tyle mały, że mieści się ono w całości w ramach wspomnianego ograniczenia*

8 sekund przed ponowieniem próby pobrania pliku. To nie przypadek a konsekwencja wspomnianego wcześniej ograniczania tempa generowania komunikatów ICMP przez dany węzeł, w wersji linuksowej nazywanego oficjalnie *rate limiting*. Gdy porównamy znaczniki czasowe ograniczające przedział czasu, w którym wysłano 6 takich komunikatów (48,974812 dla pakietu 2. i 49,034812 dla pakietu 12.), otrzymamy tempo 6 komunikatów na 0,06 sekundy, czyli 100 komunikatów na sekundę. Nasze przypuszczenia możemy zweryfikować za pomocą stosownego polecenia:

```
Linux% sysctl -a | grep icmp_rate
net.ipv4.icmp_ratemask = 6168
net.ipv4.icmp_ratelimit = 100
```

(program `grep` wykorzystywany jest tu do wybrania interesujących wierszy z obszernego raportu generowanego przez polecenie `sysctl -a`). Zmienna `icmp_ratemask` wskazuje podzbiór typów komunikatów, których dotyczy opisywane limitowanie: wartość `6168` (dziesiętnie), równoważna szesnastkowemu `0x1818`, czyli binarnemu `0001100000011000`, oznacza zbiór typów {3, 4, 11 i 12} (skrajny prawostronny bit maski reprezentuje typ 0). Gdybyśmy w tej masce wyzerowali bit reprezentujący typ 3 (czyli nadali jej postać *xxxxxxxxxxxx0xxx*) bądź ustalili „ratę" limitowania na 0 (oznaczające brak limitowania), komunikaty ICMP byłyby w naszym przykładzie konsekwentnie generowane w odpowiedzi na *każde* żądanie klienta, wskutek czego klient błyskawicznie — bez 8-sekundowej zwłoki — wykonałby serię ośmiu bezskutecznych prób. Zachowanie takie można by zaobserwować, używając serwera `tftp` w systemie Windows XP niestosującym limitowania tempa generacji komunikatów ICMP.

Można by w tym miejscu zapytać, w jakim celu klient podejmuje kolejne próby pobrania pliku, skoro już pierwszy komunikat ICMP powinien przekonać go o bezskuteczności takiego postępowania? Kwestia ta odkrywa jedną z osobliwości programowania aplikacji sieciowych — większość systemów nie informuje procesów opartych na protokole UDP o nadchodzeniu komunikatów ICMP adresowanych do tych procesów, chyba że te zażyczą sobie otrzymywania takiej informacji, wywołując specjalną funkcję (np. funkcję `connect` gniazda UDP). Większość klientów TFTP nie wywołuje takiej funkcji, co notabene obnaża w naszym przykładzie słabość mechanizmu (jałowych) żądań i odpowiedzi: proces klienta, nie dysponując żadną informacją o losie wysyłanych przez siebie pakietów, wciąż ponawia próby wykonania operacji. I chociaż TFTP wyposażono w rozszerzenia niwelujące ten

mankament (patrz [RFC2349]), to — jak zobaczymy w rozdziale 16. — znacznie lepszym wyjściem jest wykorzystywanie aplikacji bazujących na bardziej zaawansowanych protokołach transportowych, np. na protokole TCP.

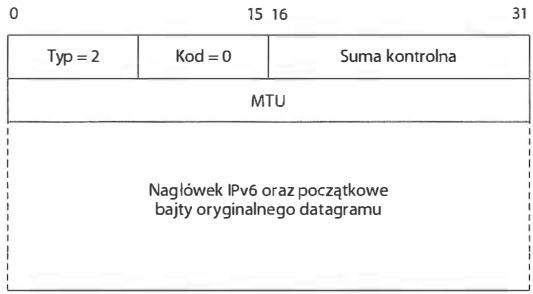
8.3.2.5. Komunikat ICMPv4 PTB (typ 0, kod 4)

Jeśli router IPv4 otrzymuje datagram przeznaczony do forwardowania i rozmiar tego datagramu przekracza wartość MTU dla łącza skojarzonego z interfejsem, przez które datagram ów ma zostać przekazany do następnego przeskoku, konieczne jest podzielenie tego datagramu na kilka powiązanych ze sobą *fragmentów* (patrz rozdział 10.). Jeżeli jednak w nagłówku rzeczonego datagramu ustawiony jest bit DF, co oznacza zakaz fragmentowania, router nie ma innego wyjścia jak odrzucenie datagramu, z jednoczesnym wysłaniem komunikatu ICMPv4 *Destination Unreachable (PTB)* (typ 3, kod 4). Ponieważ routerowi znana jest wspomniana wartość MTU, dołącza ją do generowanego komunikatu.

Oryginalnie komunikat ten zaprojektowany został na potrzeby diagnostyki sieci, lecz równie dobrze może być wykorzystywany do rozpoznawania MTU dla ścieżki prowadzącej do określonego hosta (w ramach procesu PMTUD — *Path MTU Discovery*), co niezbędne jest w sytuacji, gdy pożądane jest unikanie fragmentowania przesyłanych pakietów. Mechanizm ten wykorzystywany jest głównie przez protokół TCP, czym zajmiemy się szczegółowo w rozdziale 14.

8.3.2.6. Komunikat ICMPv6 PTB (typ 2, kod 0)

W wersji ICMPv6, w sytuacji gdy pakiet jest zbyt duży w stosunku do MTU łącza prowadzącego do następnego przeskoku, tworzony jest specjalny komunikat *Packet Too Big (PTB)* (typ 2, kod 0), o formacie przedstawionym na rysunku 8.6.



Rysunek 8.6. *Komunikat ICMPv6 Packet Too Big (typ 2) podobny jest do komunikatu ICMPv4 Destination Unreachable, zawiera jednak dodatkowo 32-bitowe pole określające MTU dla następnego przeskoku*

Jak pamiętamy, w protokole IPv6 fragmentację datagramu wykonywać może jedynie jego nadawca, który tym samym musi korzystać z mechanizmu PMTUD w celu określenia maksymalnej dopuszczalnej wartości MTU na całej trasie datagramu. To właśnie jest powodem oddzielenia opisywanego komunikatu od komunikatów typu *Destination Unreachable*, bo stanowi on podstawowy środek realizacji wspomnianego PMTUD. Mimo wszystko, może się jednak zdarzyć (choć zdarza się raczej rzadko), że już po wysłaniu datagramu zmienią się warunki i rozpoznana wartość PMTU okaże się zbyt duża,

i wędrówkę pakietu trzeba będzie przerwać. Przerwanie to jest wówczas sygnalizowane opisywanym komunikatem ICMPv6, zawierającym sugerowaną wartość MTU — podobnie jak w nowszych implementacjach ICMPv4.

8.3.2.7. Komunikat ICMPv6 Beyond Scope of Source Address (kod 2)

Zgodnie z opisem z rozdziału 2., adresy IPv6 mają zróżnicowane zakresy, możliwe jest więc skonstruowanie datagramu IP, w którym adresy źródłowy i docelowy różnią się zakresami, ponadto możliwe jest, że adres docelowy będzie nieosiągalny w zakresie adresu źródłowego — przykładowo datagram z adresem źródłowym lokalnym dla łącza może być kierowany do docelowego adresu globalnego, co wymagać będzie przejścia tego datagramu przez jeden lub więcej routerów. I jeśli wówczas adres źródłowy będzie mieć zakres zbyt wąski z perspektywy któregoś routera, datagram zostanie odrzucony przez router, z jednoczesnym wysłaniem do nadawcy opisywanego komunikatu ICMP.

8.3.2.8. Komunikat ICMPv6 Source Address Failed Ingress/ Egress Policy (kod 5)

Komunikat ICMPv4 identyfikowany przez typ 1 i kod 5 jest bardziej specjalizowaną odmianą komunikatu typu 1 o kodzie 1; generowany jest w sytuacji, gdy określone zasady filtrowania wejściowego lub wyjściowego stają na przeszkodzie pomyślnemu dostarczeniu datagramu do miejsca przeznaczenia. Może się tak zdarzyć np. w sytuacji, gdy host próbuje wysłać datagram IP z adresem źródłowym o nieoczekiwanym prefiksie (patrz [RFC3704]).

8.3.2.9. Komunikat ICMPv6 Reject Route to Destination (kod 6)

Niekiedy w tablicach trasowania routerów celowo umieszcza się pozycje, które powodują odrzucanie pewnej grupy pakietów; postępowanie takie podyktowane jest chęcią zapobieżenia wyciekowi „wrażliwych" informacji do określonych miejsc przeznaczenia. Taka „odrzucająca" pozycja może być skonstruowana przy użyciu „bezsensownego" adresu w roli następnego przeskoku, przy czym do wyboru są dwie możliwości: ten „bezsensowny" adres może być adresem nietrasowalnym w globalnym Internecie — takie pozycje w tabelach trasowania nazywamy *marsjańskimi* — albo adresem teoretycznie poprawnym na globalnym poziomie, lecz jeszcze nieużywanym, bo nieprzydzielonym do użytku — tego rodzaju pozycje nazywane są popularnie *bogonami*, przez analogię do fizyki cząstek elementarnych[1]. Odrzucenie pakietu „pasującego" do takiej pozycji

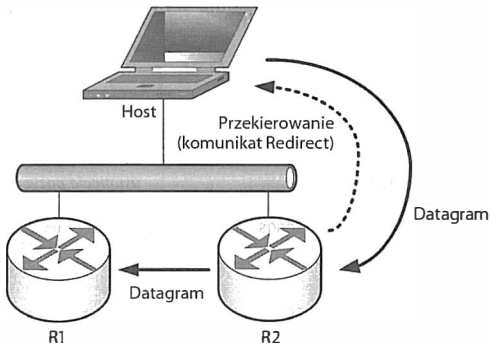
[1] Zgodnie z jedną z fundamentalnych zasad mechaniki kwantowej — zasadą de Broglie'a dualizmu korpuskularno-falowego — każdy ruch falowy można interpretować także jako strumień poruszających się cząstek. Gdy rozwinąć tę ideę w kierunku metafory (choć już niekonieczne na bazie faktów fizycznych), każdemu zjawisku, idei, pomysłowi, koncepcji itp. przyporządkować można pewną (hipotetyczną) cząstkę elemntamą; zgodnie z tą zasadą, wszelkiego rodzaju niedorzeczności, absurdy czy niemożliwości utożsamiane są z cząstką o nazwie *bogon*, wywodzącą się od angielskiego *bogosity*. W przełożeniu na komunikację sieciową *bogonem* może być bezsensowny adres, lecz także nieprawidłowo uformowany pakiet czy błędna fizycznie ramka ethernetowa emitowana przez uszkodzone urządzenie. Jednak to tylko przysłowiowy wierzchołek góry lodowej: niespożyta inwencja programistów, hakerów i internautów w ogóle zaowocowała rozwinięciem oryginalnej idei w gałąź — jeśli nie wiedzy, to zapewne interesującej kreatywności — o nazwie *bogodynamika kwantowa*, z dziwacznymi cluonami, clutronami, psytonami i futonami. Zainteresowanych tematem czytelników odsyłam do stosowanego hasła w wikipedii: *http://pl.wikipedia.org/wiki/Quantum_bogodynamics — przyp.tłum.*

powoduje jednoczesne wysłanie do nadawcy komunikatu ICMPv6 *Reject Route to Destination* (typ 1, kod 6). (Analogiczne pozycje, powodujące odrzucanie pakietów, lecz bez generowania komunikatów ICMP, nazywane są popularnie *czarnymi dziurami* — znowu przez analogię do fizyki).

8.3.3. Komunikat Redirect (typ 5 w ICMPv4, typ 137 w ICMPv6)

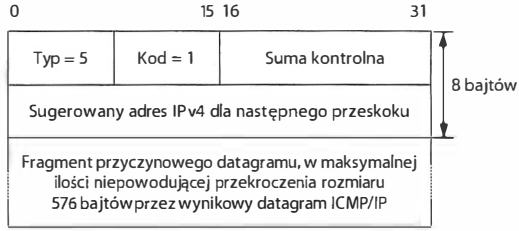
Zdarza się, iż router stwierdza, że to nie on powinien być adresatem otrzymanego właśnie pakietu; przekierowuje wówczas ten pakiet do właściwego (jego zdaniem) routera, powiadamiając jednocześnie o tym fakcie hosta nadawcę, za pomocą komunikatu ICMP *Redirect* (w ICMPv4 typ 5, w ICMPv6 typ 137). W ten oto prosty sposób funkcja forwardująca może korygować swe działanie (opisane z detalami w rozdziale 5.).

Na rysunku 8.7 widzimy segment sieci i dwa routery — R_1 i R_2. Gdy host niepoprawnie prześle datagram do routera R_2, ten przekieruje otrzymany datagram do R_1, wysyłając do hosta odpowiedni komunikat. Prawdopodobną reakcją hosta będzie skorygowanie swych tabel trasowania; teoretycznie podobną korekcję mogłyby wykonywać także routery niewłaściwie adresujące wysyłane pakiety, co jednak jest odradzane z tej prostej przyczyny, że kiedy używają dynamicznych protokołów trasowania — zawsze znają optymalną trasę do każdego (osiągalnego) miejsca przeznaczenia.



Rysunek 8.7. *Host niepoprawnie kieruje datagram do routera R2 (zamiast R1); R2 wykrywa pomyłkę, przekierowuje datagram do R1, a do hosta wysyła komunikat Redirect. Host prawdopodobnie wykorzysta komunikat do zaktualizowania swych tablic trasowania — tak by w przyszłości datagramy adresowane do tego samego przeznaczenia, co obecnie, kierowane były bezpośrednio do routera R1*

Komunikat *Redirect* (przedstawiony na rysunkach 8.8 i 8.9 w formacie dla — odpowiednio — ICMPv4 i ICMPv6) zawiera adres IP routera (lub hosta docelowego, jeśli ten osiągalny jest za pomocą dostarczania bezpośredniego) sugerowanego jako następny przeskok. Oryginalnie specyfikacja ICMPv4 odróżniała przypadek korygowania *adresu IP* (typ 5, kod 1) od korygowania *prefiksu IP* (typ 5, kod 0), jednak z chwilą pojawienia się adresowania bezklasowego i CIDR (patrz rozdział 2.) przyjęto zasadę, że korekcja tabel trasowania, wykonywana przez host w wyniku otrzymania komunikatu *Redirect*, powinna obejmować konkretny adres, nie cały prefiks; w rezultacie host konsekwentnie posługujący się błędnym prefiksem może otrzymać od domyślnego routera całą serię komunikatów



Rysunek 8.8. *Komunikat ICMPv4 Redirect zawiera adres IPv4 właściwego routera, który powinien zostać użyty jako następny przeskok dla pakietu, którego kopia (być może częściowa) stanowi ładunek użyteczny. Host zwykle sprawdza adres źródłowy w nagłówku IPv4 nadchodzącego pakietu enkapsulującego komunikat Redirect w celu zweryfikowania, czy komunikat ten wysłany został przez aktualnie domyślny router*

Redirect dotyczących różnych węzłów docelowych zlokalizowanych poza jego własną podsiecią (i sumarycznie korekcja wielu pojedynczych adresów da efekt taki sam, jaki dałaby korekcja błędnego prefiksu).

Operację przekierowania możemy zaobserwować, celowo fałszując w tablicy trasowania hosta adres domyślnego routera przez wskazanie innego routera w tej samej podsieci, a następnie próbując skontaktować się ze zdalnym serwerem. System bierze sfałszowaną zawartość tablicy za dobrą monetę i próbuje wysyłać pakiet za pośrednictwem niewłaściwego hosta:

```
C:\> netstat -rn
Network Dest      Netmask     Gateway          Interface     Metric
0.0.0.0           0.0.0.0     10.212.2.1       10.212.2.88   1


C:\> route delete 0.0.0.0                          usunięcie pozycji domyślnego trasowania
C:\> route add 0.0.0.0 mask 0.0.0.0 10.212.2.112   dodanie nowej pozycji
C:\> ping ds1.eecs.berkeley.edu sends thru 10.212.2.112   wysłanie pakietu do serwera
                                                          10.212.2.112


Badanie ds1.eecs.berkeley.edu [169.229.60.105] z 32 bajtami danych:
Odpowiedź z 169.229.60.105: bajtów=32 czas=1ms TTL=250
Odpowiedź z 169.229.60.105: bajtów=32 czas=5ms TTL=250
Odpowiedź z 169.229.60.105: bajtów=32 czas=1ms TTL=250
Odpowiedź z 169.229.60.105: bajtów=32 czas=1ms TTL=250
Statystyka badania ping dla 169.229.60.105:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0
             (0% straty),
Szacunkowy czas błądzenia pakietów w milisekundach:
    Minimum = 1ms, Maksimum = 5ms, Czas średni = 2ms
```

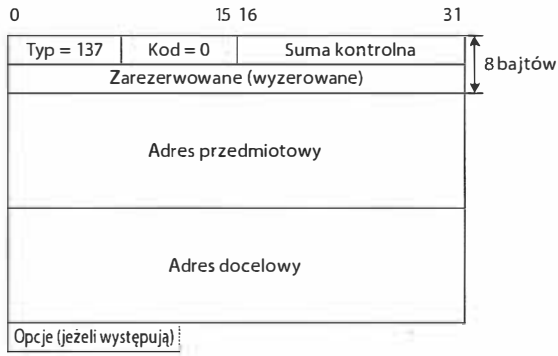
Podobnie jak poprzednio, zobaczmy, co dzieje się na linuksowym serwerze monitorowanym za pomocą programu `tcpdump` (nieistotna część raportu została pominięta):

```
Linux# tcpdump host 10.212.2.88

1 20:27:00.759340 IP 10.212.2.88 > ds1.eecs.berkeley.edu: icmp 40:
                  echo request seq 15616
2 20:27:00.759445 IP 10.212.2.112 > 10.212.2.88: icmp 68:
                  redirect ds1.eecs.berkeley.edu to host 10.212.2.1
```

```
3 20:27:00.759468 IP 10.212.2.88 > ds1.eecs.berkeley.edu: icmp 40:
                  echo request seq 15616
...
```

Jak widzimy, program ping wysyła pakiet *Echo Request* do węzła identyfikowanego nazwą `ds1.eecs.berkeley.edu`, która zostaje zidentyfikowana przez usługę DNS (patrz rozdział 11.) jako równoważna adresowi IP `169.229.60.105`. Ze względu na spreparowanie tablic trasowania, następnym przeskokiem dla wysyłanego pakietu staje się router o adresie `10.212.2.112` zamiast właściwego domyślnego routera `10.212.2.1`. Ponieważ router `10.212.2.112` jest poprawnie skonfigurowany, zauważa zaistniałą nieprawidłowość i przekierowuje otrzymany pakiet do routera `10.212.2.1`, a do hosta nadawcy wysyła komunikat ICMPv4 *Redirect* sugerujący, że w przyszłości wszelki ruch kierowany do lokalizacji `ds1.eecs.berkeley.edu` powinien przechodzić przez domyślny router o adresie `10.212.2.1`.

W ICMPv6 komunikat *Redirect* (typ 137, kod 0 — patrz rysunek 8.9) zawiera adres przedmiotowy i adres docelowy, oba definiowane w kontekście procesu odnajdywania sąsiadów (ND, patrz podrozdział 8.5). *Adres przedmiotowy* to sugerowany dla następnego przeskoku adres IPv6 lokalny dla łącza, zaś *Adres docelowy* to kopia adresu docelowego z nagłówka przyczynowego datagramu IPv6. W szczególnej sytuacji, gdy węzeł docelowy jest sąsiadem (na łączu) hosta odbierającego komunikat *Redirect*, oba wspomniane adresy są identyczne. Stanowi to sposób poinformowania hosta o tym, że inny host znajduje się na tym samym łączu, nawet jeśli używa innego prefiksu adresu (patrz [RFC5942]).

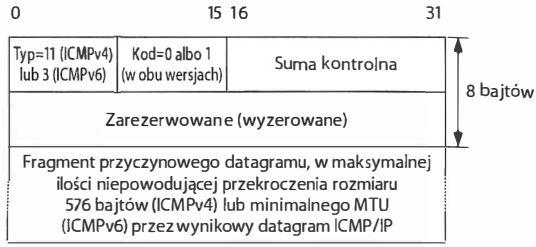


Rysunek 8.9. *Komunikat ICMPv6 Redirect. W polu Adres przedmiotowy znajduje się sugerowany adres „lepszego" routera w roli następnego przeskoku dla danego Adresu docelowego. Komunikat ten może być także wykorzystywany do wskazania, że adres docelowy identyfikuje sąsiada odbiorcy — w takiej sytuacji Adres przedmiotowy jest identyczny z Adresem docelowym*

Podobnie jak inne komunikaty ICMPv6 grupy ND, komunikat *Redirect* może zawierać opcje, szczególnie *Target Link-Layer Address* i *Redirected Header*. Pierwsza z nich wymagana jest w sytuacji, gdy komunikat *Redirect* używany jest w sieci nieobsługującej rozgłaszania (NBMA — *non-broadcast multiple access*), ponieważ w takim przypadku host otrzymujący wspomniany komunikat nie ma innej możliwości efektywnego określenia adresu lokalnego dla łącza dla następnego przeskoku. Opcja *Redirected Header* zawiera natomiast fragment kopii przyczynowego datagramu. Szczegółowym formatem obu wymienionych opcji zajmiemy się w podrozdziale 8.5 poświęconym procedurze ND.

8.3.4. Komunikat ICMP Time Exceeded (typ 11 w ICMPv4, typ 3 w ICMPv6)

W celu uniknięcia możliwości nieskończonego krążenia datagramów IP w „zapętlonych" trasach ogranicza się czas ich istnienia. W wersji IPv4 służy do tego pole *Czas życia*, w wersji IPv6 pole *Limit przeskoków*. Według oryginalnej koncepcji pole *Czas życia* miało być konsekwentnie dekrementowane co sekundę, przy czym wymagano jego zmniejszenia przynajmniej o 1 na każdym przeskoku (biorąc pod uwagę oczywisty fakt, że czas przebywania pakietu w danym przeskoku mógł być krótszy niż sekunda). Efektywnie wymóg ten dało się zrealizować w ten sposób, że wspomniane pole dekrementowane było natychmiast przy wejściu pakietu do przeskoku, a kolejne dekrementacje następowały z upływem kolejnych sekund przebywania pakietu w tymże przeskoku. A ponieważ typowy pakiet przebywał w każdym przeskoku jedynie przez drobny ułamek sekundy, pole, które w zamierzeniu miało być (tykającym wstecz) zegarem, stało się de facto licznikiem przeskoków. Metamorfoza ta urzeczywistniona została w nowej nazwie pola w wersji IPv6 oraz ograniczeniu jego dekrementowania do jednorazowego aktu przy wejściu datagramu na przeskok. W obu wersjach IP wyzerowanie pola *Czas życia* lub *Limit przeskoków* przy kolejnej dekrementacji jest traktowane jako zakończenie żywota datagramu, a rolę swoistego nekrologu wysyłanego do nadawcy pełni wówczas komunikat *Time Exceed* (w ICMPv4 typ 11, w ICMPv6 typ 3, w obu wersjach kod 0); pakiet przychodzący do przeskoku z zerową wartością wspomnianego pola jest — oczywiście — natychmiast odrzucany, bez dekrementacji. Opisany mechanizm, oprócz niezbędnego ograniczania czasu istnienia datagramów, spełnia także istotną rolę w funkcjonowaniu programu śledzenia trasy `traceroute` (w Linuksie) i `tracert` (w Windows). Format komunikatu *Time Exceed* przedstawiono na rysunku 8.10.

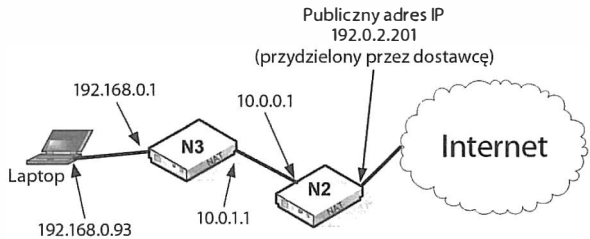


Rysunek 8.10. *Komunikat ICMP Time Exceed w formacie dla ICMPv4 i ICMPv6. Kod równy 0 oznacza wyczerpanie Czasu życia (Limitu przeskoków), Kod równy 1 oznacza wyczerpanie czasu przeznaczonego na defragmentację datagramu*

Komunikat *Time Exceed* generowany jest także w innej sytuacji — po przekroczeniu czasu przeznaczonego na reasemblację pofragmentowanego datagramu. Ponieważ łatwo można by doprowadzić do przepełnienia buforów routera przez zasypanie go niekończącym się strumieniem „pośrednich" fragmentów datagramu (które muszą być pamiętane aż do momentu nadejścia ostatniego fragmentu, inaczej nie można by przeprowadzić defragmentacji), ogranicza się czas, w ciągu którego nadejść muszą do routera *wszystkie* fragmenty danego datagramu — gdy w tym czasie nie nadejdą, cały datagram jest odrzucany, a do nadawcy wysyłany jest komunikat *Time Exceed* z wartością 1 w polu *Kod*.

8.3.4.1. Przykład: program traceroute

Opisany mechanizm limitowania liczby przeskoków na trasie datagramu można wykorzystać do zidentyfikowania kolejnych routerów na tej trasie — co wyjaśnimy na przykładzie wersji IPv4. Wysyłając datagram z wartością 1 w polu *Czas życia*, spowodujemy zakończenie jego istnienia już w najbliższym routerze, a nadesłany w związku z tym komunikat ICMP *Time Excced* będzie źródłem informacji na temat adresu IP rzeczonego routera. Kiedy wyślemy datagram z wartością 2 w polu *Czas życia*, poznamy w taki sam sposób adres drugiego routera na trasie itp. — sukcesywnie zwiększając wartość pola *Czas życia* w kolejno wysyłanych pakietach, możemy rozpoznać całą trasę. Aby uzyskać bardziej wiarygodną informację i bardziej kompletną informację, dla danej wartości *Czasu życia* wysyłać można nie jeden, lecz kilka pakietów (domyślnie wysyłane są trzy). Adresem źródłowym wszystkich wysyłanych datagramów jest adres IPv4 interfejsu routera, do którego przyłączony jest host wysyłający te pakiety. Ideę tę przedstawiono na rysunku 8.11.



Rysunek 8.11. *Program traceroute umożliwia zidentyfikowanie trasy wiodącej do określonego miejsca przeznaczenia, przy założeniu, że trasa ta nie zmienia się zbyt często. Każdy węzeł na trasie identyfikowany jest zwykle adresem IP interfejsu prowadzącego do następnego przeskoku*

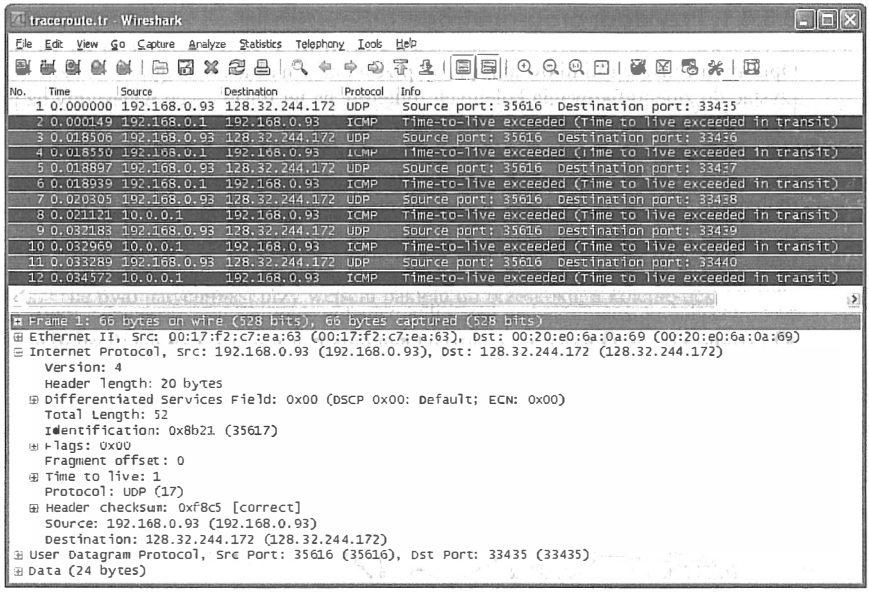
W konfiguracji przedstawionej na rysunku program `traceroute`, uruchomiony w laptopie, wysyła datagramy UDP (patrz rozdział 10.), zgodnie z opisaną strategią, do docelowego hosta `www.eecs.berkeley.edu` (nieuwidocznionego na rysunku). Program uruchomiony został za pomocą polecenia

```
Linux% traceroute -m 2 www.cs.berkeley.edu
```

a sporządzony przezeń raport prezentuje się następująco:

```
traceroute to web2.eecs.berkeley.edu (128.32.244.172), 2 hops max,
52 byte packets
 1 gw (192.168.0.1) 3.213 ms 0.839 ms 0.920 ms
 2 10.0.0.1 (10.0.0.1) 1.524 ms 1.221 ms 9.176 ms
```

Parametr `-m` specyfikuje wykonanie śledzenia tylko dwóch początkowych hostów na trasie — w polu *Czas życia* umieszczane są kolejno wartości 1 i 2. Wyniki śledzenia widoczne są w dwóch kolejnych wierszach raportu, przykładowo w pierwszym wierszu widzimy adres pierwszego routera `192.168.0.1` i wyniki trzech niezależnych pomiarów czasu wędrówki pakietu (3,213 milisekund, 0,839 milisekundy i 0,92 milisekundy); wyraźnie dłuższy czas wędrówki pierwszego pakietu wynika z konieczności jednorazowego wykonania pewnej dodatkowej pracy, związanej m.in. z transakcją ARP. Na rysunkach 8.12 i 8.13 widoczne są (w oknie programu Wireshark) szczegóły wykonywanych operacji oraz struktura poszczególnych komunikatów.

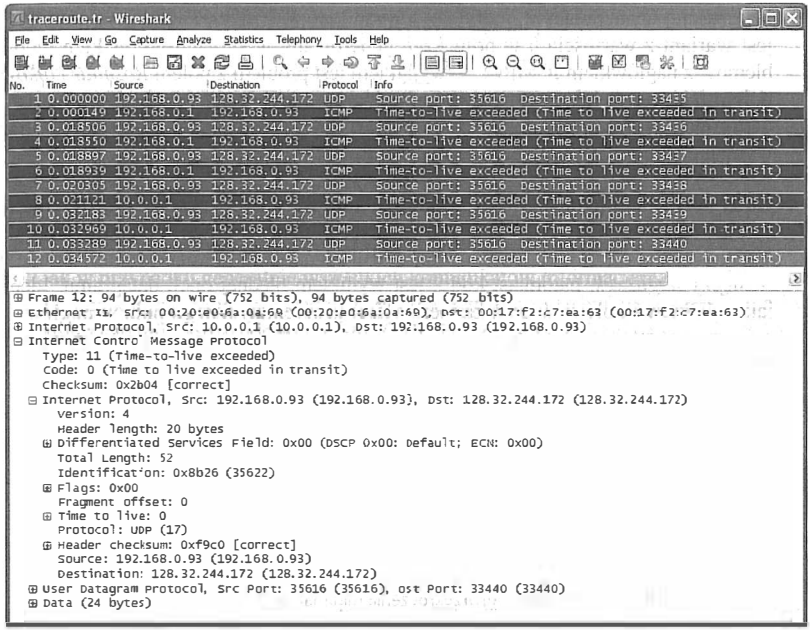


Rysunek 8.12. *Działanie programu traceroute w ramach IPv4 rozpoczyna się od wysłania datagramu UDP/IPv4 z wartością 1 w polu Czas życia, do portu docelowego 33435. Po wysłaniu trzech datagramów z tą samą wartością w polu Czas życia pole to jest inkrementowane i wysyłane są trzy następne pakiety. Każda seria datagramów z określoną wartością w tym polu kończy swój żywot na ściśle określonym węźle, a generowany w związku z tym komunikat ICMP Time Exceed niesie dla nadawcy informację o adresie IP tegoż węzła*

I tak na rysunku 8.12 widzimy wysyłanie przez `traceroute` sześciu datagramów, z sukcesywnie zwiększanymi numerami portu docelowego, począwszy od 33435; pierwsze trzy datagramy wysyłane są z wartością 1 w polu *Czas życia*, trzy następne z wartością 2. W rozwiniętej części raportu widzimy wiersz 1. pakietu oraz szczegóły komunikatu ICMPv4 (typ 11, kod 0) generowanego wskutek wyczerpania czasu jego życia. Takie komunikaty ICMP generowane są sześć razy, po jednym dla każdego wysłanego datagramu; nadawcą pierwszych trzech jest router **N3** o adresie `192.168.0.1`, trzy następne generowane są przez router **N2** o adresie `10.0.0.1`. Na rysunku 8.13 widoczne są szczegóły ostatniego z wysłanych komunikatów ICMP. Zawiera on kopię oryginalnego datagramu (pakietu o numerze 11) w takiej postaci, w jakiej został odebrany przez router **N2**, czyli z wartością 1 w polu *Czas życia*; dekrementacja tego pola spowodowała jego wyzerowanie, wskutek czego datagram nie został już forwardowany do adresu `128.32.244.172`, natomiast router **N2** wysłał stosowny komunikat do nadawcy datagramu.

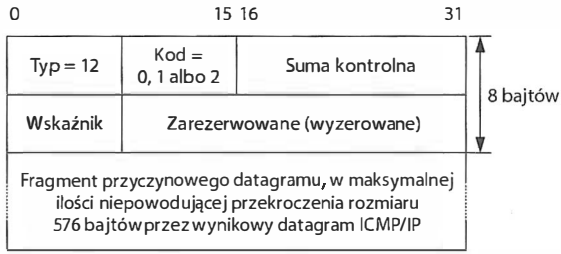
8.3.5. Komunikat Parameter Problem (typ 12 w ICMPv4, typ 4 w ICMPv6)

Komunikat generowany jest przez host lub router odbierający datagram IP z nieodwracalnie uszkodzonym nagłówkiem. Gdy wspomniany datagram nie może być żadną miarą poprawnie przetworzony i żaden inny komunikat ICMP nie opisuje w adekwatny sposób przyczyny tego stanu rzeczy, komunikat *Parameter Problem* stanowi rodzaj zastępczej



Rysunek 8.13. *Ostatni z komunikatów ICMPv4 Time Exceed wysyłany jest przez węzeł N2 o adresie IP 10.0.0.1. Komunikat ten zawiera kopię przyczynowego datagramu, pole Czas życia w wewnętrznym nagłówku IPv4 jest wyzerowane w rezultacie ostatniej dekrementacji*

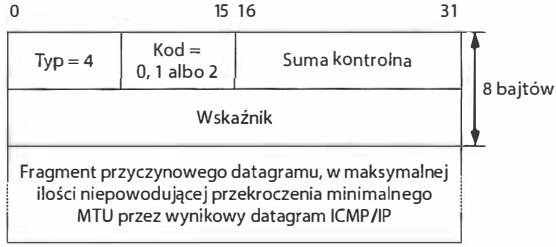
sygnalizacji bliżej niesprecyzowanego problemu. W obu wersjach protokołu IP, jeśli niepoprawne jest któreś z pól nagłówka otrzymanego datagramu — bo np. wartość tego pola wykracza poza dopuszczalny zakres — w polu *Wskaźnik* komunikatu ICMP (patrz rysunek 8.14) znajduje się offset tegoż pola w bajtach, liczony od początku nagłówka przyczynowego datagramu. Przykładowo w ICMPv4 wartość 1 pola *Wskaźnik* identyfikuje pole *DS/ECN* nagłówka IPv4 (patrz rozdział 5.).



Rysunek 8.14. *Komunikat ICMPv4 Parameter Problem generowany jest w sytuacji, gdy żaden inny komunikat ICMPv4 nie identyfikuje zaistniałego błędu w lepszy sposób. W polu Wskaźnik znajduje się offset (w bajtach) przyczynowego pola, liczony względem początku nagłówka IPv4 przyczynowego datagramu. Najczęściej w polu Kod znajduje się wartość 0; wartość 2 oznacza błędną wartość w polu IHL lub Rozmiar całkowity. Dawniej Kod=1 identyfikował brak obowiązkowej opcji, obecnie jednak wartość ta ma wyłącznie znaczenie historyczne*

Najczęściej występującym podtypem komunikatu *Parameter problem* w wersji ICMPv4 jest wariant z wartością 0 w polu *Kod*, generowany w sytuacji wszystkich niemal problemów z nagłówkiem ICMPv4 — wyjątkiem jest problem z polem *Rozmiar całkowity* lub polem *IHL*, powodujący generowanie podtypu z wartością 2 w polu *Kod*. Wariant z kodem równym 1 ma obecnie znaczenie historyczne, niegdyś sygnalizował brakujące, acz wymagane opcje, np. etykiety bezpieczeństwa.

Na rysunku 8.15 pokazano format komunikatu *Parameter problem* w wersji ICMPv6. W porównaniu z ICMPv4 doprecyzowane zostało znaczenie poszczególnych podtypów: kod 0 oznacza ogólny błąd nagłówka, kod 1 sygnalizuje nierozpoznaną wartość w polu *Następny nagłówek*, natomiast kod 2 jest wynikiem błędnej opcji datagramu. Identycznie jak w ICMPv4 pole *Wskaźnik* zawiera offset problematycznego pola w nagłówku przyczynowego datagramu IPv6, przykładowo wartość 40 identyfikuje problem z pierwszym nagłówkiem rozszerzenia.



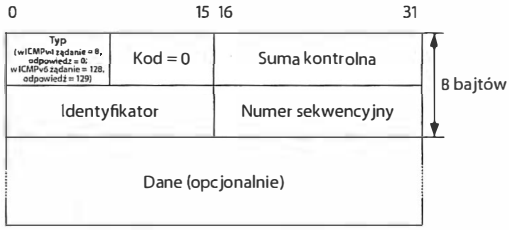
Rysunek 8.15. *Komunikat ICMPv6 Parameter Problem. Pole Wskaźnik zawiera offset (w bajtach) przyczynowego pola względem początku przyczynowego datagramu. Kod równy 0 oznacza błędne pole w nagłówku, Kod 1 nierozpoznaną wartość w polu Następny nagłówek, a Kod 2 nierozpoznaną opcję IPv6*

8.4. Komunikaty informacyjne ICMP

Mimo iż w specyfikacji protokołu ICMP zdefiniowano wiele różnych komunikatów informacyjnych, funkcjonujących w parach żądanie-odpowiedź, m.in. *Address Mask* (typy 17 i 18), *Timestamp* (typy 13 i 14) oraz *Information* (typy 15 i 16), funkcje spełniane przez te komunikaty powierzone zostały z biegiem czasu odrębnym protokołom, dedykowanym wykonywaniu specjalizowanych zadań — m.in. protokołowi DHCP opisywanemu w rozdziale 6. W powszechnym użyciu nadal jednak pozostają dwie grupy komunikatów tej kategorii: *Echo* (typy 8 i 0) stanowiące podstawę działania popularnego programu `ping` oraz komunikaty związane z mechanizmem *odnajdywania routerów* (*Router Discovery* — RD). I chociaż ta druga grupa nie jest szeroko wykorzystywana przez IPv4, to już jej odpowiednik na gruncie ICMPv6, stanowiący element mechanizmu *odnajdywania sąsiadów* (*Neighbor Discovery* — ND), ma dla protokołu IPv6 znaczenie fundamentalne. ICMPv6 został także rozbudowany o funkcje związane z mobilnym IPv6 oraz mechanizmy zarządzania multicastingiem (MLD — *Multicast Listener Discovery*, patrz rozdziały 6. i 9.). Szczegółami mechanizmu ND zajmiemy się w podrozdziale 8.5.

8.4.1. Komunikaty Echo Request/Reply (ping) (typy 0/8 w ICMPv4, typy 129/128 w ICMPv6)

Najpopularniejszym bodaj komunikatem informacyjnym ICMP jest *Echo*, w ICMPv4 identyfikowany wartością typu 8 (żądanie) i 0 (odpowiedź); w ICMPv6 wartości pola *Typ* równe są (odpowiednio) 128 i 129. Komunikaty te mogą mieć dowolny rozmiar, w praktyce limitowany jedynie maksymalną dopuszczalną wielkością wynikowego datagramu ICMP/IP. Od implementacji ICMP wymaga się, by wszystkie dane dołączone do żądania *Echo* zwrócone zostały w niezmienionej postaci w komunikacie niosącym odpowiedź. Podobnie jak w przypadku generalnie wszystkich komunikatów informacyjnych ICMP, wartości pól *Identyfikator* i *Numer sekwencyjny* w odpowiedzi muszą być identyczne z tymi w odnośnym żądaniu. Format komunikatu ICMP *Echo* przedstawiono na rysunku 8.16.



Rysunek 8.16. *Format komunikatów Echo Request i Echo Reply w wersjach ICMPv4 i ICMPv6. Ewentualne dane załączone do komunikatu Echo Request muszą być wiernie zwrócone w Echo Reply. Identyfikator wykorzystywany jest przez NAT (patrz rozdział 7.) do kojarzenia żądań z odpowiedziami*

Komunikat *Echo* stanowi podstawę działania programu `ping`[2], powszechnie używanego do badania dostępności zdalnego hosta. Otrzymanie odpowiedzi ze zdalnego hosta można było kiedyś uważać za gwarancję tego, że ze wspomnianym zdalnym hostem możliwe jest nawiązanie połączenia także za pomocą innych mechanizmów (np. zdanego logowania); obecnie, gdy niemal każda sieć chroniona jest przez jakąś odmianę firewalla, założenie takie jest wysoce nieuzasadnione.

Nazwa `ping` stanowi analogię do sygnału wysyłanego przez aktywny sonar, emitowanego w celu określenia lokalizacji obiektu względem obserwatora. Program `ping` został przez Mike'a Muussa (1958–2000) w grudniu 1983 roku i zaimplementowany po raz pierwszy w jądrze Uniksa BSD 4.x, a historię jego rozwoju poznać można, czytając stronę [PING] zawierającą również odsyłacz do kodu źródłowego (w języku C).

Analogią do numerów portów w warstwie transportowej, różnicujących poszczególne żądania wychodzące z tego samego adresu IP, są na gruncie komunikatów informacyjnych ICMP *identyfikatory żądań*, które host wysyłający żądania wykorzystuje później do demultipleksowania nadchodzących odpowiedzi (i które stanowią także jeden z parametrów konfigurowania firewalli, o czym pisaliśmy w rozdziale 7.). W systemach uniksowych do pola *Identyfikator* komunikatu *Echo* wpisywany jest zwykle identyfikator procesu realizującego instancję programu `ping`, co umożliwia różnicowanie żądań i odpowiedzi związanych z poszczególnymi instancjami.

[2] Pomysłowi użytkownicy szybko skonstatowali, że adekwatną nazwą dla odpowiedzi na żądanie `ping` jest (oczywiście) `pong` — i tak oto folklor komputerowy wzbogacił się o kolejny element — *przyp. tłum.*

Gdy uruchomiona zostaje nowa instancja programu `ping`, *Numer sekwencyjny* pierwszego komunikatu-żądania *Echo* równy jest 0 i zwiększany jest o 1 w każdym następnym żądaniu. Numer ten wyświetlany jest w raporcie produkowanym przez pogram, co umożliwia użytkownikowi identyfikowanie pakietów zagubionych, zdublowanych lub odpowiedzi przychodzących w kolejności różnej niż kolejność wysyłania odnośnych żądań. Nie zapominajmy, że ICMP opiera się na ruchu niegwarantowanym (*best-effort delivery*), każda z tych okoliczności ma więc realną szansę wystąpienia. ICMP weryfikuje natomiast integralność swych pakietów za pomocą własnej sumy kontrolnej.

Zazwyczaj w żądaniu *Echo* generowanym przez program `ping` umieszczany jest także znacznik czasowy odzwierciedlający moment wysłania pakietu; znacznik ten, wraz z innymi danymi zawartymi opcjonalnie w żądaniu, zwracany jest w komunikacie odpowiedzi. Odejmując wartość owego znacznika od bieżącego wskazania czasu, otrzymujemy natychmiast czas wędrówki pakietu w obu kierunkach (RTT — *Round-Trip Time*); zauważmy, że obliczenia te odbywają się wyłącznie w kontekście lokalnego zegara hosta nadawcy, nie ma zatem znaczenia ewentualna rozbieżność wskazań zegarów między różnymi komputerami. Na podobnej zasadzie dokonywany jest pomiar RTT w opisywanym wcześniej programie `traceroute`.

W pierwszych wersjach programu `ping` co sekundę wysyłane było jedno żądanie *Echo* i wyświetlane były wszystkie nadchodzące odpowiedzi. W nowszych implementacjach zróżnicowano zarówno sposób zachowania programu, jak i formaty wyświetlanych raportów. Przykładowo w Windows wysyłane są domyślnie cztery pakiety w odstępach jednosekundowych, wyświetlane są odpowiedzi i program kończy pracę; użytkownik może jednak wywołać program `ping` z parametrem `-t`, w rezultacie czego program będzie wysyłał żądania i wyświetlał nadchodzące odpowiedzi, aż do jawnego przerwania kombinacją klawiszy *Ctrl+C*. W Linuksie takie „zapętlone" działanie aż do przerwania jest opcją domyślną. Z biegiem lat pojawiły się nowe wersje programu, udostępniające wiele opcji umożliwiających m.in. manipulowanie zawartościami niektórych pól wynikowego datagramu ICMP/IP. Niektóre wersje umożliwiają włączanie do komunikatów *Echo* dodatkowych danych zgodnych z pewnymi predefiniowanymi wzorcami, co znakomicie pomaga w identyfikowaniu błędów sprzętowych zależnych od konkretnej postaci przesyłanych danych.

W poniższym przykładzie żądania *Echo* wysyłane są na adres rozgłoszeniowy podsieci; w systemie Linux użytkownik w takiej sytuacji musi użyć parametru wywołania `-b` w celu zasygnalizowania, iż faktycznie jest to jego intencją — jak łatwo się domyślić, pojedyncze żądanie *Echo* wysłane na adres rozgłoszeniowy skutkuje dużą liczbą odpowiedzi, system upewnia się więc dodatkowo, że użytkownik istotnie wie, co robi. Niezależnie od użycia parametru `-b` i tak wyświetlane jest stosowne ostrzeżenie.

```
Linux% ping -b 10.0.0.127
WARNING: pinging broadcast address
PING 10.0.0.127 (10.0.0.127) from 10.0.0.1 : 56(84) bytes of data.
64 bytes from 10.0.0.1: icmp_seq=0 ttl=255 time=1.290 msec
64 bytes from 10.0.0.6: icmp_seq=0 ttl=64 time=1.853 msec (DUP!)
64 bytes from 10.0.0.47: icmp_seq=0 ttl=64 time=2.311 msec (DUP!)
64 bytes from 10.0.0.1: icmp_seq=1 ttl=255 time=382 usec
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=1.587 msec (DUP!)
64 bytes from 10.0.0.47: icmp_seq=1 ttl=64 time=2.406 msec (DUP!)
64 bytes from 10.0.0.1: icmp_seq=2 ttl=255 time=380 usec
```

```
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=1.573 msec (DUP!)
64 bytes from 10.0.0.47: icmp_seq=2 ttl=64 time=2.394 msec (DUP!)
64 bytes from 10.0.0.1: icmp_seq=3 ttl=255 time=389 usec
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=1.583 msec (DUP!)
64 bytes from 10.0.0.47: icmp_seq=3 ttl=64 time=2.403 msec (DUP!)

--- 10.0.0.127 ping statistics ---
4 packets transmitted, 4 packets received,
+8 duplicates, 0% packet loss
round-trip min/avg/max/mdev = 0.380/1.545/2.406/0.765 ms
```

Program `ping` wysyła cztery (domyślnie) żądania *Echo* o numerach sekwencyjnych (kolejno) 0, 1, 2 i 3, co skutkuje nadejściem 12 odpowiedzi widocznych w raporcie — na każde z żądań odpowiadają trzy komputery o adresach `10.0.0.1`, `10.0.0.6` i `10.0.0.47`. Wszystkie trzy odpowiedzi na to samo żądanie mają — oczywiście — ten sam numer sekwencyjny, co czujny program opatruje ostrzeżeniem `(DUP!)`. Zauważmy także różne wartości `ttl` świadczące o zróżnicowanym „oddaleniu" komputerów.

Przedstawione „pingowanie" ukierunkowane na adres rozgłoszeniowy jest wygodnym sposobem wypełniania lokalnych tablic ARP (o których pisaliśmy w rozdziale 4.). Mimo iż żądania *Echo* wysyłane są na adres rozgłoszeniowy, generowane w reakcji na nie odpowiedzi ukierunkowane są na nadawcę tych żądań. Ponieważ oba hosty — wysyłający żądanie i odpowiadający — znajdują się w tej samej podsieci, wymieniane są między nimi również komunikaty ARP, w efekcie czego w tablicy ARP nadawcy żądań znajdą się aktualne pozycje związane z adresami `10.0.0.1`, `10.0.0.6` i `10.0.0.47`, nawet jeśli nie było ich tam przed uruchomieniem programu `ping`. Warto jednak w tym miejscu przestrzec, że *nie jest obowiązkowe* generowanie odpowiedzi na żądania *Echo* wysyłane jako rozgłoszenia: w systemie Linux domyślnie odpowiedzi takie są generowane, w ale w Windows XP nie.

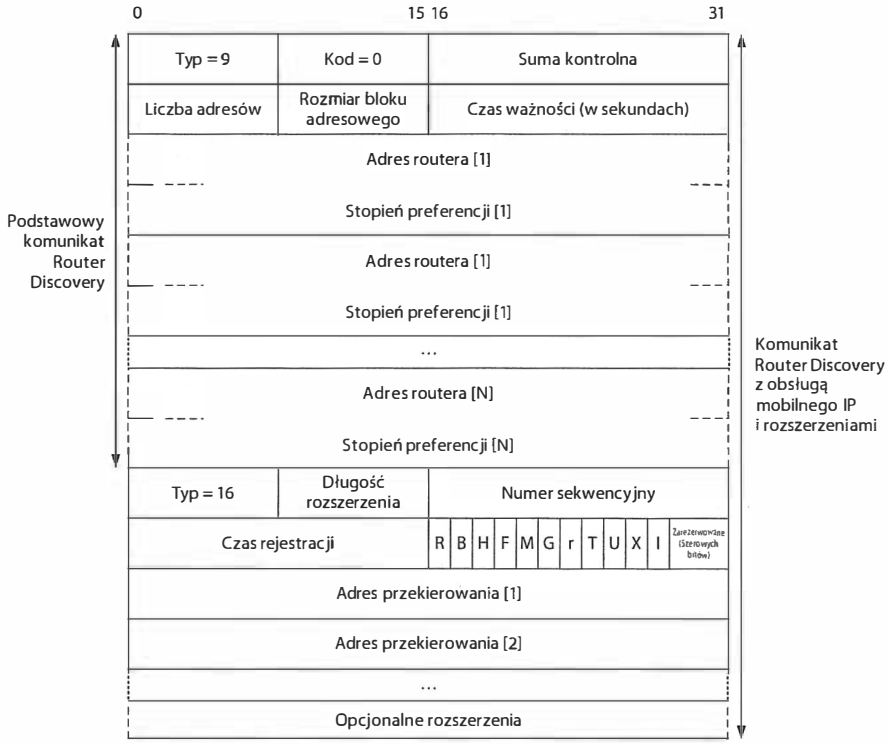
8.4.2. Odnajdywanie routerów: komunikaty Router Solicitation i Router Advertisement (typy 9 i 10 w ICMPv4)

W rozdziale 6. pokazywaliśmy, jak za pomocą protokołu DHCP host uzyskuje adres IP oraz rozpoznaje obecność pobliskich routerów. Wspominaliśmy wówczas, że istnieje alternatywny sposób identyfikowania routerów, nazywany odnajdywaniem routerów (*Router Discovery* — RD). Mimo iż zaprojektowany został na potrzeby zarówno hostów implementujących IPv4, jak i tych implementujących IPv6, na gruncie IPv4 nie jest powszechnie używany, bo ustępuje miejsca bardziej popularnemu DHCP; ostatnio jednak tendencja ta wydaje się zmieniać w związku z mobilnym IP, opiszemy więc pokrótce funkcjonowanie RD w kontekście IPv4. W wersji IPv6 mechanizm RD jest częścią funkcji SLAAC (patrz rozdział 6.) i logicznym elementem mechanizmu *odnajdywania sąsiadów* (*Neighbor Discovery* — ND), której poświęcamy podrozdział 8.5.

W wersji IPv4 mechanizm *Router Discovery* realizowany jest za pomocą pary komunikatów informacyjnych *Router Solicitation* (RS, typ 10) i *Router Advertisement* (RA, typ 9). W ramach RA routery mogą wysyłać swe ogłoszenia (*advertisements*) w dwojaki sposób: kierując periodycznie komunikaty na adres grupowy wszystkich hostów (*All Hosts* — `224.0.0.1`) bądź do konkretnego hosta, w odpowiedzi na wysłane przez niego żądanie RS. Żądania RS wysyłane są na adres grupowy wszystkich routerów (*All Routers* —

224.0.0.2). (O multicastingu i adresach grupowych w sieci lokalnej piszemy w rozdziale 9.). Podstawowym zadaniem mechanizmu *Router Discovery* jest udostępnianie hostowi informacji o wszystkich routerach w jego lokalnej podsieci, dzięki czemu może on wybrać domyślną trasę między nimi; ponadto mechanizm ten umożliwia wykrywanie routerów skłonnych pełnić funkcję agenta domowego w ramach mobilnego IP.

Na rysunku 8.17 przedstawiono format komunikatu *Router Advertisement* w wersji IPv4. Zasadniczą jego częścią jest lista adresów IPv4 wszystkich tych węzłów, które mogą być użyte przez host w charakterze domyślnego routera. W polu *Liczba adresów* znajduje się liczba bloków adresowych zawartych w komunikacie. Każdy blok adresowy składa się z adresu IPv4 oraz *Stopnia preferencji* przypisanego temu adresowi. Stopień ten jest 32-bitową liczbą całkowitą ze znakiem, zapisywaną w uzupełnieniu do 2; wartość 0 oznacza preferencję domyślną, wartość minimalna — 0x80000000 — wyraża przeciwwskazanie do używania danego adresu w charakterze domyślnego routera.

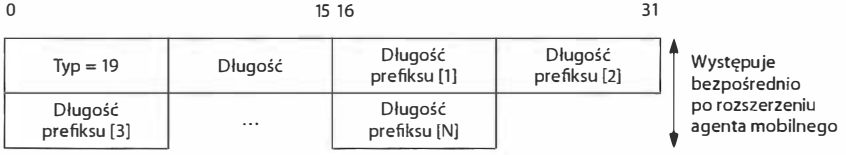


Rysunek 8.17. *Komunikat ICMPv4 Router Advertisement (RA) zawiera listę adresów IPv4 routerów, które mogą być użyte przez host w roli następnego przeskoku. Stopień preferencji umożliwia operatorom sieci różnicowanie przydatności poszczególnych routerów — wyższy stopień preferencji oznacza większe prawdopodobieństwo wyboru routera przez host. W mobilnym IPv4 (patrz [RFC5944]) komunikat RA rozszerzony jest o ogłoszenie adresów potencjalnych agentów mobilnych i (lub) specyfikację długości prefiksów w proponowanych adresach routerów*

Rozmiar bloku adresowego określony jest w 4-bajtowych słowach i w tym przypadku ma wartość 2. Pole *Czas ważności* określa (w sekundach) czas, przez jaki lista adresowa może być uważana przez host za wiążącą ofertę.

Komunikaty RA wykorzystywane są również przez węzeł mobilnego IP (patrz [RFC5944]) do lokalizowania agenta mobilności (czyli agenta domowego lub agenta zdalnego). Na rysunku 8.17 po tradycyjnej części komunikatu RA następuje rozszerzenie *ogłoszenia agenta mobilności* (*Mobility Agent Advertisement*). Rozszerzenie to identyfikowane jest wartością 16 w bajcie typu, natomiast w polu *Długość rozszerzenia* znajduje się liczba bajtów zajmowanych przez rozszerzenie, czyli sumaryczna długość pól *Numer sekwencyjny* i *Czas rejestracji*, pola znaczników i listy adresów przekierowania — czyli wartość $6 + 4K$, gdzie K jest liczbą wspomnianych adresów przekierowania. Pole *Numer sekwencyjny* wypełniane jest przez agenta przy tworzeniu komunikatu, zaś *Czas rejestracji* to liczba sekund, przez które agent zamierza oczekiwać na rejestrację węzłów (wartość `0xffff` oznacza oczekiwanie bez ograniczenia). W polu znaczników zdefiniowane są bity: `R` (1 oznacza wymaganą rejestrację dla usług MIP), `B` (agent zbyt zajęty, by przyjmować nowe rejestracje), `H` (agent może pełnić funkcję agenta domowego), `F` (agent może pełnić funkcję agenta zdalnego) `M` (obsługiwany jest format minimalnej enkapsulacji, definiowany w [RFC2004]), `G` (agent obsługuje tunelowanie GRE w enkapsulowanych datagramach), `T` (obsługiwane jest tunelowanie odwrotne opisywane w [RFC3024]), `U` (obsługiwane jest tunelowanie UDP opisywane w [RFC3519]), `X` (dopuszczalne jest anulowanie rejestracji opisywane w [RFC3543]) oraz `I` (zdalny agent obsługuje rejestrację regionalną, patrz [RFC4857]). Bit `r` jest zarezerwowany i musi mieć wartość `0`, podobnie jak pięć niewykorzystywanych obecnie końcowych bitów.

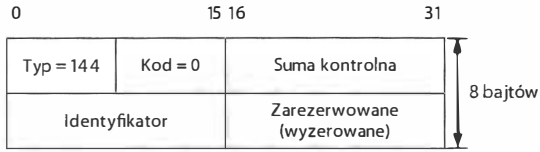
Inne rozszerzenie, jakie towarzyszyć może opisanej konfiguracji, związane jest z wyodrębnieniem prefiksów z adresów routerów (w części bazowej komunikatu). Wyodrębnienie takie może być wykorzystywane przez węzeł mobilny do wykrywania zmiany sieci, zgodnie z algorytmem 2 z dokumentu [RFC5944]. Węzeł zapamiętuje w cache listę prefiksów dostępnych na konkretnym łączu; zmiana tej listy świadczy o przemieszczeniu się do innej sieci. Format opisywanego rozszerzenia przedstawiono na rysunku 8.18. W polu *Typ* znajduje się wartość 19, pole długość ma wartość tożsamą z polem *Liczba adresów* w podstawowej części komunikatu, dalej następują bajty określające długości poszczególnych prefiksów.



Rysunek 8.18. *Opcjonalne rozszerzenie komunikatu ICMPv4 Router Advertisement specyfikujące długości prefiksów proponowanych adresów routerów w części podstawowej, czyli liczbę najbardziej znaczących bitów składających się na prefiksy poszczególnych adresów. Rozszerzenie to wykorzystywane jest przez mobilny IP*

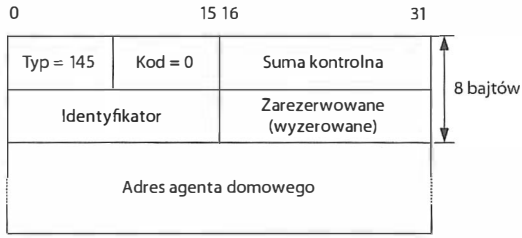
8.4.3. Komunikaty Home Agent Address Discovery Request/Reply (typy 144/145 w ICMPv6)

W dokumencie [RFC6275] zdefiniowano cztery komunikaty związane z mobilnym IPv6 (MIPv6). Dwa z nich przeznaczone są do dynamicznego odnajdywania adresu agenta domowego, dwa pozostałe wykorzystywane są do konfigurowania węzłów mobilnych i ich „przenumerowywania". Żądanie *Home Agent Address Discovery*, o formacie przedstawionym na rysunku 8.19, wysyłane jest przez węzeł mobilny przy wchodzeniu do nowej sieci w celu dynamicznego odnalezienia agenta domowego. Węzeł wysyła to żądanie na adres anycast agentów domowych MIPv6, używając swego prefiksu domowego; adresem źródłowym jest zwykle adres przekierowania węzła, czyli adres, który węzeł nabywa przy wchodzeniu do nowej sieci (patrz rozdział 5).



Rysunek 8.19. *Żądanie MIPv6 Home Agent Address Discovery zawiera Identyfikator, który skopiowany zostaje do odpowiedzi. Komunikat wysyłany jest na adres anycast agentów domowych MIPv6*

Odpowiedź *Home Agent Address Discovery*, o formacie widocznym na rysunku 8.20, udzielana jest przez węzeł, który zdecyduje się pełnić rolę agenta domowego dla określonego węzła mobilnego o ustalonym prefiksie domowym. Odpowiedź ta, zawierająca adres agenta, wysyłana jest bezpośrednio na adres unicast węzła mobilnego, czyli najprawdopodobniej na jego adres przekierowania.

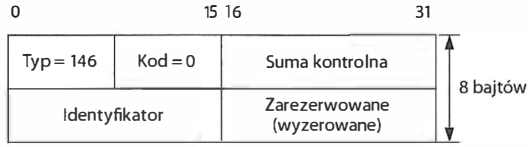


Rysunek 8.20. *Odpowiedź MIPv6 Home Agent Address Discovery zawiera Identyfikator skopiowany z odnośnego żądania oraz listę adresów węzłów skłonnych pełnić rolę agenta domowego*

Po ustanowieniu skojarzenia z nowym agentem domowym węzeł mobilny inicjuje zwykle procedurę aktualizacji wiązań (*binding updates*), o której pisaliśmy w rozdziale 5.

8.4.4. Komunikaty Mobile Prefix Solicitation/Advertisement (typy 146/147 w ICMPv6)

Widoczny na rysunku 8.21 komunikat *Mobile Prefix Solicitation* wysyłany jest do agenta domowego przez węzeł mobilny w sytuacji, gdy adres domowy tego węzła bliski jest wygaśnięcia. Komunikat zawiera opcję *Home Address* (spośród opcji docelowych

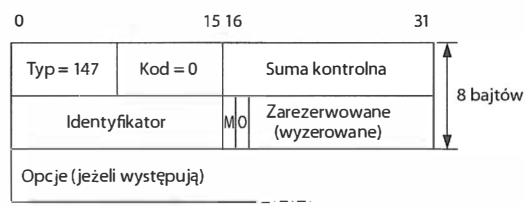


Rysunek 8.21. *Komunikat MIPv6 Mobile Prefix Solicitation wysyłany jest przez węzeł mobilny opuszczający sieć w celu wyzwolenia komunikatów Mobile Prefix Advertisement*

IPv6 — patrz rozdział 5.), a jego integralność chroniona jest za pomocą mechanizmu IPsec (o którym piszemy w rozdziale 18.).

W polu *Identyfikator* znajduje się losowo wybrana wartość, niezbędna do skojarzenia żądania z odpowiedzią. Komunikat ten podobny jest do komunikatu Router Solicitation (RS) z tą różnicą, że wysyłany jest do agenta domowego, a nie do routera lokalnej podsieci.

Datagram enkapsulujący odpowiedź na komunikat (*Mobile Prefix Advertisement*), w formacie przedstawionym na rysunku 8.22, musi zawierać nagłówek rozszerzenia *Trasowanie* w wersji RH2 (patrz punkt 5.3.2). W polu *Identyfikator* znajduje się kopia wartości identyfikatora odnośnego żądania. Ustawienie na 1 bitu *M* (*Managed Address*) oznacza, że host powinien unikać autokonfiguracji, wykonując „stanową" wersję konfigurowania adresów; ustawienie na 1 bitu *O* (*Other*) oznacza natomiast, że „stanowe" konfigurowanie dostarcza także informacji dodatkowych niezwiązanych z adresami. Na końcu komunikatu mogą występować opcje *Prefix Information* (jedna lub więcej).



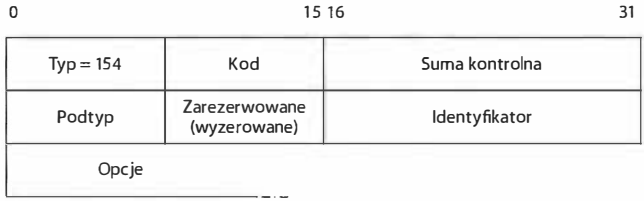
Rysunek 8.22. *Komunikat MIPv6 Mobile Prefix Advertisement. Pole Identyfikator skopiowane jest z odnośnego żądania Mobile Prefix Solicitation. Ustawiony bit M (Managed Address) oznacza, że host powinien unikać autokonfiguracji, wykonując „stanową" wersję konfigurowania adresów; ustawiony bit O (Other) oznacza dostępność dodatkowych informacji w ramach konfiguracji stanowej*

Zadaniem komunikatu *Mobile Prefix Advertisement* jest poinformowanie węzła mobilnego o zmianie jego prefiksu domowego. Integralność tego komunikatu zapewnia się zwykle przy użyciu IPsec (patrz rozdział 18.), co chronić ma węzeł mobilny przed sfabrykowanymi komunikatami tego typu. Opcja *Prefix Information*, o formacie opisanym w sekcji 4.6.2 dokumentu [RFC4861], zawiera prefiks, którego węzeł mobilny może użyć do konfigurowania swego adresu domowego; komunikat może zawierać wiele takich opcji.

8.4.5. Komunikaty szybkiego przełączania w mobilnych IPv6 (typ 154 w ICMPv6)

W dokumencie [RFC5568] zdefiniowano mechanizm *szybkiego przełączania połączeń* (*fast handovers*) dla mobilnego IPv6, w skrócie FMIPv6. Jego istotą jest zmniejszenie opóźnienia związanego z przełączaniem węzła między punktami dostępowymi (AP),

dzięki swoistemu „przewidywaniu" przez węzeł informacji o adresach, jakie będą przez niego używane w nowych warunkach, czyli w kontekście nowego punktu dostępowego. Cel ten realizuje się przy użyciu tzw. *routerów proxy*, zachowujących się w przybliżeniu tak, jak zachowywać się będą rzeczywiste routery obsługujące węzeł po przełączeniu do nowej sieci. Z mechanizmem tym związane są dwa komunikaty ICMPv6: *Proxy Router Solicitation* (w skrócie *RtSolPr*) i *Proxy Router Advertisement* (w skrócie *PrRtAdv*). Podstawowy format obu tych komunikatów przedstawiono na rysunku 8.23.

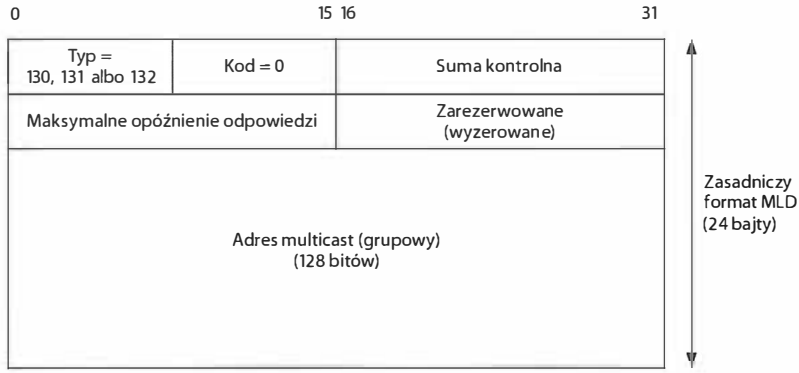


Rysunek 8.23. *Podstawowy format komunikatów ICMPv6 używanych przez FMIPv6. Poszczególne komunikaty różnicowane są za pomocą pól Kod i Podtyp. Komunikaty indagujące (solicitation — Kod=0, Podtyp=2) mogą (w ramach opcji) zawierać adres łącza danych nadawcy i adres (jeśli jest znany) łącza danych preferowanego następnego punktu dostępowego. Komunikaty ogłaszające używają Kodu 0 – 5 oraz Podtypu 3; różne wartości Kodu rozróżniają odmienne warunki, m.in. tryb generowania komunikatu (jako odpowiedź na indagowanie albo z inicjatywy nadawcy), zmianę prefiksu lub informacji o routerze oraz dostępność DHCP. Różne warunki związane są z obecnością odmiennych opcji w komunikacie*

Węzeł mobilny może zawczasu dysponować informacjami o adresach lub identyfikatorach związanych z punktem dostępowym, na który wkrótce się przełączy (informacje te zdobyć może np. przez „skanowanie" sieci 802.11). Komunikat *RtSolPr* zawiera wartość 0 w polu *Kod* i wartość 2 w polu *Podtyp*; w komunikacie tym musi wystąpić przynajmniej jedna opcja — *New Access Point Link-Layer Address* — wskazująca adres AP, od którego węzeł mobilny żąda niezbędnych informacji. Komunikat *RtSolPr* może także zawierać opcję *Link-Layer Address*, reprezentującą adres źródłowy (jeśli jest znany); jej format jest ściśle związany z mechanizmem odnajdywania sąsiadów (ND), odkładamy więc jej omówienie do podrozdziału 8.5.

8.4.6. Komunikaty Multicast Listener Query/Report/Done (typy 130/131/132 w ICMPv6)

Zadaniem komunikatów grupy *Multicast Listener Discovery* (MLD), definiowanych w dokumentach [RFC2710] i [RFC3590], jest zarządzanie adresami multicast na łączach wykorzystujących IPv6. Komunikaty te stanowią odpowiednik protokołu IGMP wykorzystywanego przez IPv4, opisywanego w rozdziale 9., w którym szczegółowo zajmiemy się także komunikatami wymienionymi w tytule, tu ograniczymy się jedynie do opisania ich formatu w wersji 1., wspólnego dla wszystkich trzech typów: *MLD Query* (130), *MLD Report* (131) i *MLD Done* (132). Format ten przedstawiamy na rysunku 8.24; wymienione komunikaty zawierają opcję *Router Alert* (z grupy opcji „skok po skoku"), a pole *Limit przeskoków* w nagłówku enkapsulującego je datagramu IPv6 ma wartość 1.



Rysunek 8.24. *Format komunikatów MLDv1. Żądania (Typ=130) mogą mieć charakter generalny (w celu rozpoznania ogółu używanych adresów multicast — pole Adres multicast ma wówczas wartość 0) lub być ukierunkowane na konkretny adres multicast (w celu określenia, czy jest nadal używany). Maksymalne opóźnienie odpowiedzi określa czas, przez jaki router wysyłający komunikat oczekuje na rejestrowanie się zainteresowanych hostów pod wybranym adresem multicast. W komunikatach raportach pole Adres Multicast zawiera adres grupy multicast, w której rejestruje się nadawca, natomiast w komunikatach Done nadawca określa w ten sposób adres grupy, którą opuszcza*

Głównym celem komunikatów MLD jest umożliwienie routerom rozpoznania, które hosty przynależne są do grupy identyfikowanej przez określony adres multicast. Wersja MLDv2, którą opisujemy w następnym punkcie, rozszerza tę funkcjonalność o możliwość ograniczenia przez host zbioru adresów źródłowych, z których życzy on sobie (albo nie życzy) otrzymywać komunikaty multicast. Nadawcami żądań *MLD Query* są routery, odpowiedziami na te żądania są komunikaty (raporty) *MLD Report*, wysyłane przez hosty; host może także wysłać komunikat *MLD Report* z własnej inicjatywy, jako sygnalizację dołączenia do określonej grupy multicast.

Pole *Maksymalne opóźnienie odpowiedzi*, niezerowe tylko w komunikatach *MLD Query*, specyfikuje maksymalną dopuszczalną dla węzła zwłokę w udzieleniu odpowiedzi, wyrażoną w milisekundach. Hosty mogą celowo opóźniać swe raporty o losowy interwał czasu, dając tym szansę dotarcia wielu komunikatom *MLD Query*; często bywa tak, że tylko jedno z otrzymanych żądań kwitowane jest odpowiedzią, pozostałe stają się nieaktualne po upływie wspomnianego interwału. Należy w tym miejscu zwrócić uwagę na fakt, że dla routera wysyłającego zapytanie *MLD Query* nie jest istotna *liczba hostów* należących do grupy multicast, lecz rozpoznanie, czy do grupy tej należy *przynajmniej jeden host*. Gdy router forwarduje pakiet z adresem docelowym multicast, wysyła *tylko jedną instancję* tego pakietu na wspomniany adres, niezależnie od rzeczywistej liczby hostów należących do tej grupy (byleby tylko była to liczba niezerowa), bo powielenie instancji pakietu dla poszczególnych hostów wykonywane jest dopiero przez mechanizmy multicast warstwy łącza danych.

Komunikaty *MLD Query* wysyłane są w dwóch odmianach: *generalnej*, mającej na celu ogólne rozpoznanie przynależności poszczególnych hostów do poszczególnych grup multicast — pole *Adres Multicast* ma wówczas wartość 0 — oraz *specyficznej*, w ramach której router rozpoznaje przynależność hostów do *konkretnej* grupy multicast, identyfikowanej przez adres we wspomnianym polu.

W komunikacie *MLD Report* (typ 131) pole *Adres Multicast* zawiera adres grupy multicast, do której w ten sposób zgłasza akces host wysyłający komunikat; analogicznie w komunikacie *MLD Done* (typ 132) pole to zawiera adres grupy multicast, z uczestnictwa w której host nadawca właśnie rezygnuje.

8.4.7. Wersja 2 komunikatu Multicast Listener Discovery (MLDv2) (typ 143 w ICMPv6)

W dokumencie [RFC3810] definiowane jest rozszerzenie oryginalnej specyfikacji MLD z [RFC2710], głównie o możliwość zawężenia zbioru adresów źródłowych, z których dany host skłonny jest akceptować albo odrzucać przychodzące komunikaty *multicast*. Funkcjonalność ta wykorzystywana jest przez mechanizm multicastingu specyficznego dla źródła (SSM — *Source-Specific Multicast*), opisywany w rozdziale 9. i dokumentach [RFC4604] oraz [RFC4607]. Mechanizm ten jest de facto rezultatem przeniesienia funkcjonalności protokołu IGMPv3 na grunt IPv6, gdzie zarządzanie adresami multicast odbywa się głównie przy udziale protokołu ICMPv6. W tym punkcie ograniczymy się do omówienia formatu komunikatów MLDv2, odkładając do rozdziału 9. opis dynamiki ich operacji.

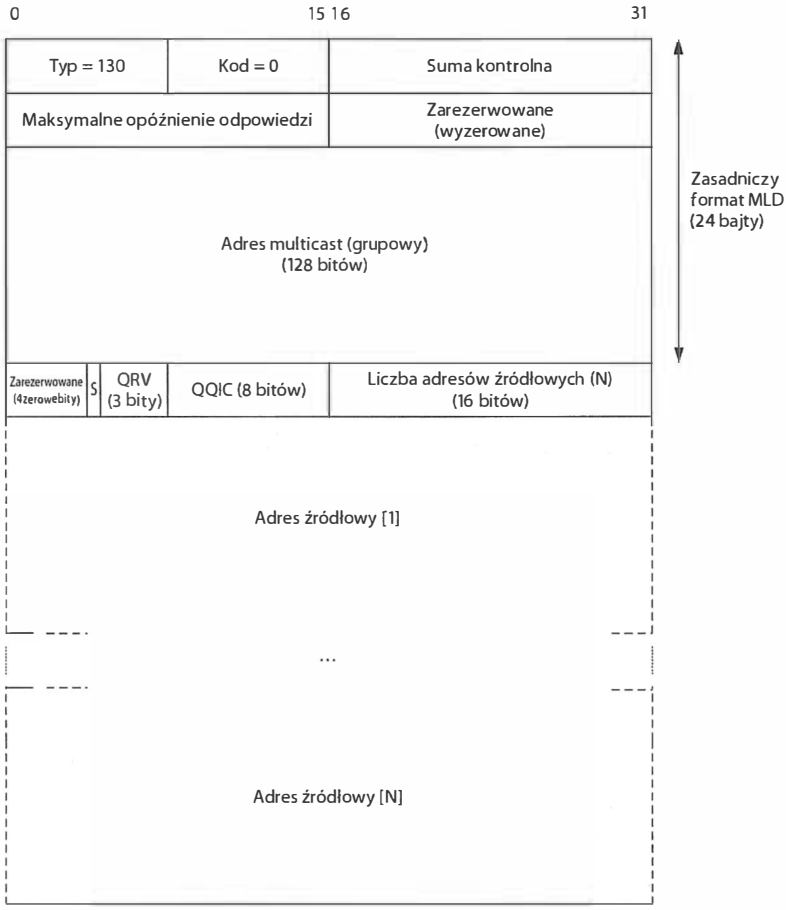
Komunikat *MLDv2 Query*, widoczny na rysunku 8.25, jest w pierwszych 24 bajtach niemal identyczny ze swym odpowiednikiem z wersji 1., jednak z dwiema różnicami: oczywistą różnicą wartości w polu *Typ* oraz znaczeniem pola *Kod maksymalnej zwłoki*. W MLDv1 w polu tym znajdowała się liczba całkowita z zakresu 0 – 65535, co ograniczało wielkość zwłoki czasowej do niecałych 65 sekund; zmiana wprowadzona w wersji MLDv2 pozwala na specyfikowanie większych interwałów, z jednoczesnym zachowaniem jak największej zgodności z poprzednikiem. Otóż jeśli najbardziej znaczący bit pola równy jest 0, pozostałe 15 bitów traktowane jest jako liczba całkowita wyrażająca (w milisekundach) wartość maksymalnej zwłoki. Jeśli bit ten ma wartość 1, zawartość pozostałych 15 bitów traktowana jest jak liczba zmiennopozycyjna[3] (patrz rysunek 8.26): na bitach od 1. do 3. kodowany jest *wykładnik*, na bitach od 4. do 15. *mantysa*, a całość reprezentuje liczbę o wartości

$$(mantysa + 4096) \cdot 2^{(wykładnik + 3)}$$

co przy ustawieniu wszystkich bitów na 1 daje maksymalną wartość interwału (8191 + 4096) * 2^{10} = 12 581 888 milisekund. Interwały nie dłuższe niż 32 767 sekund mogą być zatem kodowane w sposób tradycyjny, kompatybilny z MLDv1.

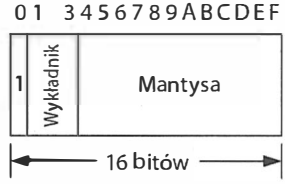
Podobnie jak w MLDv1, pole *Adres Multicast* ma wartość zerową w żądaniach generalnych i zawiera konkretny adres multicast w żądaniach specyficznych. Ustawienie bitu *S* oznacza modyfikację przetwarzania pakietu po stronie routera, w sytuacji gdy router

[3] Zwracam uwagę czytelników na dwie istotne różnice opisywanej reprezentacji w stosunku do łudząco podobnego odpowiednika IEEE-754: mantysa jest w tym przypadku *liczbą całkowitą*, a nie ułamkiem, choć domyślne jej zwiększanie o 4096 przypomina „niejawną jedynkę" poprzedzającą kropkę dziesiętną w niektórych formatach IEEE-754. Ponadto, w przeciwieństwie do IEEE-754, gdzie wykładnik jest symetrycznie polaryzowany, w tym przypadku jest on sztucznie zwiększany. Opisane różnice wynikają bezpośrednio z różnych celów, jakim służą obie reprezentacje: liczby zmiennopozycyjne IEEE-754 mają reprezentować podzbiór liczb rzeczywistych, natomiast w reprezentacji widocznej na rysunku kodowane są dodatnie liczby całkowite — *przyp. tłum.*



Rysunek 8.25. *Format komunikatu MLDv2; początkowy obszar zgodny jest z formatem MLDv1. W stosunku do wersji MLDv1 dodano możliwość selekcji dopuszczalnych adresów źródłowych*

Rysunek 8.26. *Zmiennopozycyjna reprezentacja pola Kod maksymalnej zwłoki dla wartości przekraczających 32 767 milisekund*



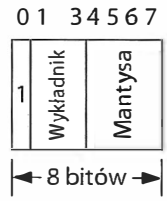
przepytywacz współdziała z routerami zapasowymi; pojawiają się wówczas subtelne kwestie związane z uaktualnieniem wartości interwału retransmisji przez poszczególne routery. Omawiamy tę kwestię szczegółowo w punkcie 9.4.5.

Pole QRV (*Querier Robustness Variable* — zmienna [odzwierciedlająca] solidność [w ocenie] przepytywacza) związane jest z zalecaną częstotliwością generowania komunikatów uaktualniających MLD, wynikającą z oczekiwanego współczynnika gubienia pakietów w podsieci. Rozmiar pola pozwala na wpisanie doń wartości nie większej niż 7 — jeśli wewnętrzna zmienna QRV nadawcy przekracza tę wartość, do pola QRV wpisywane jest 0. Zmiennymi QRV zajmiemy się szczegółowo w rozdziale 9.

Wartość w polu QQIC (*Querier's Query Interval Code*) to zakodowany interwał czasowy QQI (w sekundach) między kolejno generowanymi żądaniami MLD. Metoda kodowania, przedstawiona na rysunku 8.27, stanowi 8-bitową analogię 16-bitowego kodowania przedstawionego na rysunku 8.26: jeżeli najbardziej znaczący bit ma wartość 0, siedem pozostałych bitów tworzy liczbę całkowitą bezpośrednio wyrażająca rzeczony interwał; gdy najbardziej znaczący bit ma wartość 1, trzy kolejne bity formują wykładnik, cztery następne mantysę, a reprezentowana wartość równa jest

$$(mantysa + 16) \cdot 2^{(wykładnik + 3)}$$

co pozwala na kodowanie interwałów o wartości maksymalnej $(15 + 16) * 2^{10} = 31\,744$ sekund.

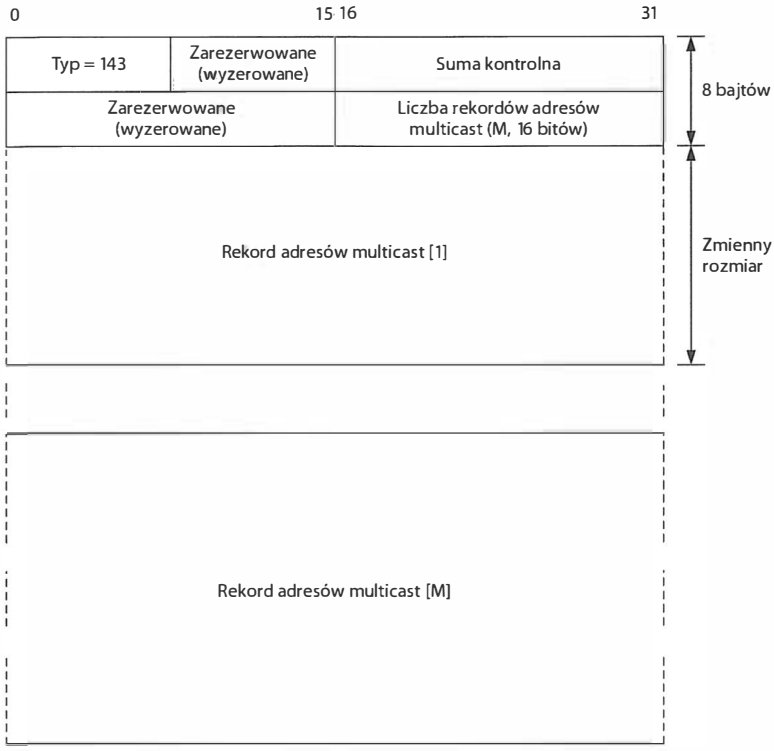


Rysunek 8.27. *Zmiennopozycyjna reprezentacja pola QQIC dla wartości przekraczających 127 sekund*

Lista *Adresów źródłowych* poprzedzona jest bezpośrednio 16-bitowym polem wskazującym ich *Liczbę*. Liczba ta równa jest zero dla żądań *MLDv2 Query* generalnych i specyficznych dla konkretnego adresu, wskazuje natomiast rzeczywistą ilość specyfikowanych adresów źródłowych w żądaniach specyficznych dla źródła (źródeł).

Rekordy adresów multicast, wykorzystywane w komunikatach *MLDv2 Reports* (patrz rysunki 8.28 i 8.29), zawierają modyfikatory filtrowania adresów źródłowych, wykonywanego przez zainteresowany węzeł IPv6 w związku z selekcją adresów źródłowych, od których tolerować będzie otrzymywanie komunikatów multicast (szczegółami tego filtrowania zajmiemy się w rozdziale 9.).

Zdefiniowane typy rekordów z rysunku 8.29 dzielą się na trzy główne kategorie, związane z *bieżącym stanem*, *zmianą trybu filtrowania* oraz *zmianą listy adresów źródłowych*. Do pierwszej kategorii należą typy `MODE_IS_INCLUDE (IS_IN)` i `MODE_IS_EXCLUDE (IS_EX)` reprezentujące (odpowiednio) włączenie i wyłączenie załączonych adresów źródłowych z zestawu adresów tolerowanych (na liście adresów musi występować przynajmniej jedna pozycja). Należące do drugiej kategorii typy `CHANGE_TO_INCLUDE (TO_IN)` i `CHANGE_TO_EXCLUDE (TO_EX)` mają podobne znaczenie, reprezentują jednak generalną zmianę trybu filtrowania (włączanie/wyłączanie) bez zmiany listy adresów. Dla odmiany sytuację, gdy zmienia się lista źródłowa, a niezmienny pozostaje tryb filtrowania, reprezentują typy `ALLOW_NEW_SOURCES (ALLOW)` i `BLOCK_OLD_SOURCES (BLOCK)` należące do trzeciej kategorii.

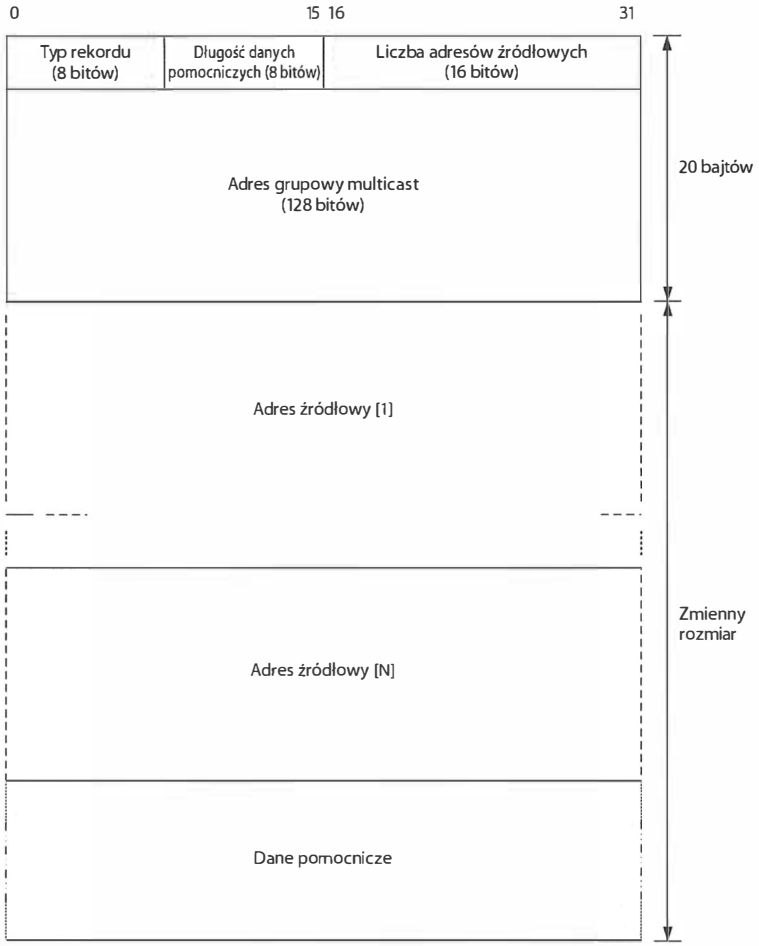


Rysunek 8.28. *Raport MLDv2 zawiera wektor rekordów reprezentujących poszczególne adresy multicast*

Specyfikacja [RFC5790], zwana „lekkim" (*lightweight*) MLDv2, w skrócie LW-MLDv2 (na gruncie IPv4 LW-IGMPv3), upraszcza radykalnie powyższą sytuację, likwidując rzadko wykorzystywany tryb wyłączania (`EXCLUDE`), co przyczynia się do upraszczania implementacji ICMPv6 i IGMPv3, dzięki rezygnacji z utrzymywania dodatkowego stanu w maszynie stanowej routera multicast.

8.4.8. Komunikaty Multicast Router Discovery (MRD) (typy 48/49/50 w IGMP, typy 151/152/153 w ICMPv6)

W dokumencie [RFC4286] opisano mechanizm *odnajdywania routerów multicast* (*Multicast Router Discovery*, w skrócie MRD) — metodę definiującą specjalne komunikaty, które używane w połączeniu z ICMPv6 lub IGMP umożliwiają odnajdywanie routerów zdolnych do forwardowania pakietów multicast oraz rozpoznawanie niektórych parametrów konfiguracji tych routerów. Funkcjonalność ta w zamierzeniu projektantów miała wspomagać tzw. podsłuchiwanie ICMP/MLD — mechanizm, dzięki któremu systemy inne niż hosty i routery, np. przełączniki warstwy 2., uzyskiwać mogą informację o lokalizacji routerów multicast i zainteresowanych nimi hostów; szczegółowy opis tego mechanizmu pozostawiamy do rozdziału 9.



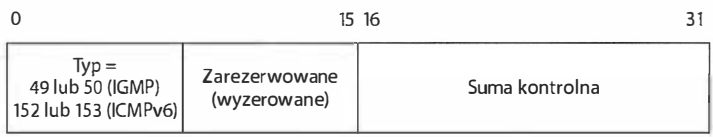
Rysunek 8.29. *Rekord adresu multicast z raportu MLDv2. Pole Typ rekordu może zawierać jedną z wartości MODE_IS_INCLUDE, MODE_IS_EXCLUDE, CHANGE_TO_INCLUDE, CHANGE_TO_EXCLUDE, ALLOW_NEW_SOURCES lub BLOCK_OLD_SOURCES; niektóre z tych wartości zostały wycofane w wersji LW-MLDv2. Pole Długość danych pomocniczych określa rozmiar tych danych liczony w 32-bitowych słowach; w wersji LW-MLDv2 pole to musi mieć wartość 0 (ponieważ pomocnicze dane nie występują)*

Komunikaty MRD zawsze wysyłane są z limitem przeskoków (czasem życia) równym 1, zawierają opcję *Router Alert* i należą do jednego z trzech typów: ogłaszanie (*Advertisement* — typ 151), indagowanie (*Solicitation* — typ 152) lub zakończenie (*Termination* — typ 153). Ogłoszenia wysyłane są periodycznie, w skonfigurowanym odstępie czasowym, w celu ujawniania obecności routerów zdolnych do forwardowania ruchu multicast; komunikaty indagujące stanowią dla routerów bodźce do generowania ogłoszeń, zaś za pomocą komunikatów typu „zakończenie" routery sygnalizują rezygnację z pełnienia ogłaszanej funkcji. Format komunikatu ogłoszenia przedstawiono na rysunku 8.30.

Rysunek 8.30. *Komunikat ogłoszenie MRD (w ICMPv6 Typ=151, w IGMP Typ=48). Pole Interwał ogłaszania określa częstotliwość wysyłania ogłoszeń niestanowiących reakcji na żądania, analogicznie pole QQI oznacza częstotliwość wysyłania żądań MLD. Parametr QRV oznacza wskaźnik niezawodności zdefiniowany w konfiguracji MLD. Adres IP routera nadawcy sygnalizuje gotowość określonego routera do forwardowania ruchu multicast. Komunikat wysyłany jest na adres grupowy podsłuchiwania (224.0.0.106 w IPv4, ff02::6a w IPv6)*

Komunikat ogłoszenie wysyłany jest z adresu IP routera (w IPv6 — z adresu lokalnego dla łącza) na adres grupowy podsłuchiwania (*All Snoopers* — `224.0.0.106`) lub na lokalny dla łącza adres multicast `ff02::6a`. Odbiorca komunikatu zapamiętuje ogłaszane przez router parametry QQI i QRV (opisujemy je dokładnie w rozdziale 9.); zauważmy, że tym razem QQI kodowane jest w postaci regularnej 16-bitowej liczby całkowitej, odmiennie niż na rysunku 8.27.

Komunikaty „indagowanie" i „zakończenie" mają format niemal identyczny, z oczywistą różnicą w polu *Typ*: w IPv4 są to (odpowiednio) typy `49` i `50`, w IPv6 typy `152` i `153` (rysunek 8.31). Komunikat indagujący stanowi żądanie wygenerowania przez router komunikatu ogłoszenia; wysyłany jest na adres grupowy wszystkich routerów (*All Routers* — `224.0.0.2`) albo na adres lokalny dla łącza `ff02::2`. Komunikat „zakończenie" wysyłany jest na wspomniany już adres podsłuchiwania (*All Snoopers* — `224.0.0.106`) lub `ff02::6a`.



Rysunek 8.31. *Format komunikatów ICMPv6 MRD Solicitation (w ICMPv6 Typ=152, w IGMP Typ=49) i MRD Termination (w ICMPv6 Typ=153, w IGMP Typ=50). Komunikaty te wysyłane są z wartością 1 pola Czas życia/Limit przeskoków i zawierają opcję Router Alert. Komunikaty MRD Solicitation wysyłane są na adres multicast All Routers (224.0.0.2 w IPv4, ff02::2 w IPv6)*

8.5. Odnajdywanie sąsiadów w IPv6

Protokół odnajdywania sąsiadów (*Neighbor Discovery Protocol*, w skrócie ND lub NDP) definiowany w dokumencie [RFC4861] łączy na gruncie IPv6 funkcjonalność trzech elementów specyficznych dla IPv4: komunikatów ICMPv4 *Router Discovery* i *Redirect* oraz protokołu ARP. Jest także nieodłącznym elementem obsługi mobilnego IPv6. W przeciwieństwie jednak do ARP i IPv4, generalnie używających broadcastingu (z wyjątkiem RD), ICMPv6 powszechnie wykorzystuje multicasting (w IPv6 nie istnieją zresztą adresy broadcast, o czym pisaliśmy w rozdziałach 2. i 5.). ND zaprojektowano z myślą o zapewnieniu węzłom (routerom i hostom), działającym na tym samym łączu lub w tym samym segmencie sieci, wzajemnego odnajdywania się, wykrywania stanów nieoperatywności oraz badania dwukierunkowości łączy. ND oferuje także funkcję autokonfiguracji

bezstanowej (o której pisaliśmy w rozdziale 6.). Cała funkcjonalność ND realizowana jest na bazie ICMPv6, na poziomie warstwy sieciowej lub powyżej niej, co czyni ND wysoce niezależnym od konkretnej technologii użytej na poziomie warstwy łącza danych; należy jednak zauważyć, że ND chętnie korzysta z możliwości multicastingu łącza danych (patrz rozdział 9.), z tego względu jego działanie może być nieco inne w na łączach NBMA (*Non-Broadcast Multiple Access*) nieobsługujących adresów broadcast i multicast.

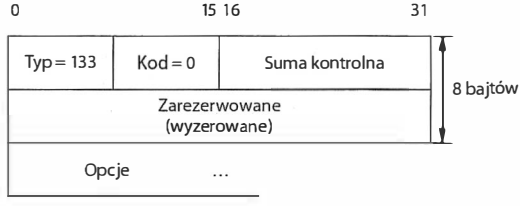
Podstawowymi filarami ND są komunikaty *indagowania sąsiadów* (*Neighbor Solicitation* — NS) i *ogłoszenia sąsiedzkie* (*Neighbor Advertisement* — NA) oferujące mapowanie adresów sieciowych na adresy łącza danych, podobnie do ARP, oraz komunikaty *indagowania routerów* (*Router Solicitation* — RS) i *ogłoszenia routerów* (*Router Advertisement* — RA) oferujące funkcjonalność odnajdywania routerów, odnajdywania agentów mobilnego IP i przekierowań oraz ograniczone wsparcie dla autokonfiguracji. W dokumencie [RFC3971] definiowana jest bezpieczna wersja ND o nazwie SEND (*SEcure ND*), rozszerzająca wersję podstawową o uwierzytelnianie i specjalne formy adresowania, głównie w związku z dodatkowymi opcjami.

Komunikaty ND to komunikaty ICMPv6 z maksymalną (255) wartością w polu *Limit przeskoków* nagłówka zewnętrznego datagramu IPv6; ewentualne „podrabiane" komunikaty ICMPv6 przychodzące spoza łącza z konieczności mają w tym polu wartość mniejszą, dzięki czemu odbiorca bezbłędnie odróżnia je od legalnych.

Rozpoczniemy od przeglądu podstawowych komunikatów ND, po czym zajmiemy się opcjami ND, które w bogatym zestawie mogą przenosić rozmaite informacje dodatkowe załączane do wspomnianych komunikatów.

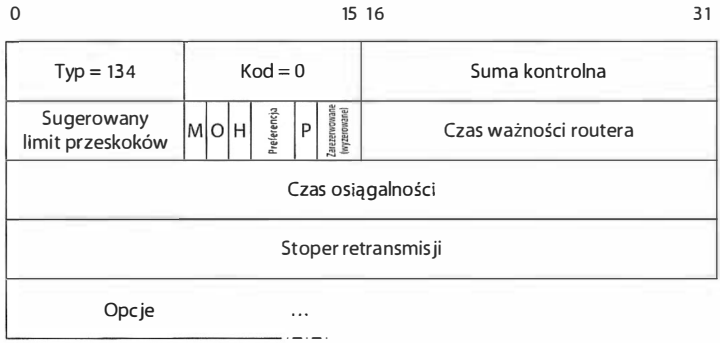
8.5.1. Komunikaty ICMPv6 Router Solicitation i Router Advertisement (typy 133 i 134)

Komunikaty *Router Advertisement* (RA) oznajmiają obecność i możliwości pobliskich routerów. Są one wysyłane periodycznie z inicjatywy routerów, lecz także w odpowiedzi na komunikaty *Router Solicitation* (RS). Komunikat RS, o formacie przedstawionym na rysunku 8.32, wysyłany jest na adres grupowy wszystkich routerów (All Routers — `ff02::2`). Jeśli nadawca komunikatu używa adresu IPv6 innego niż nieokreślony (używany przy autokonfiguracji), do komunikatu załączona jest także opcja *Source Link-Layer Address* — jedyna dopuszczalna (oprócz rzadko występującej opcji MTU) opcja w tym komunikacie (zgodnie z [RFC4861]).



Rysunek 8.32. *Komunikat ICMPv6 Router Solicitation jest bardzo prosty, lecz zwykle zawiera także opcję Source Link-Layer Address (w przeciwieństwie do swego odpowiednika z ICMPv4). Może także zawierać opcję MTU, jeśli łącze charakteryzuje się nietypową wartością MTU*

Komunikat *Router Advertisement* (RA), o formacie przedstawionym na rysunku 8.33, wysyłany jest bądź to na adres grupowy wszystkich węzłów (*All Nodes* — `ff02::1`), jeśli generowany jest z inicjatywy routera, bądź na adres unicast hosta, który uprzednio nadesłał komunikat RS. Komunikaty RA informują lokalne hosty i inne routery o szczegółach konfiguracji lokalnego łącza.



Rysunek 8.33. *Komunikat ICMPv6 Router Advertisement wysyłany jest na adres grupowy wszystkich węzłów (All Nodes — ff02::1) lub adres unicast zapytującego hosta. Odbiorca komunikatu upewnia się o jego autentyczności, sprawdzając, czy pole Limit przeskoków ma wartość 255 — co świadczy o tym, że pakiet nie był forwardowany przez router. Komunikat RA zawiera trzy znaczniki: M (Managed address configuration), O (Other stateful configuration) i H (Home Agent)*

Pole *Sugerowany limit przeskoków* zawiera zalecaną hostowi przez router wartość pola *Limit przeskoków* dla datagramów IPv6 generowanych przez ten host; wartość 0 oznacza brak sugestii. W następnym bajcie obecne są znaczniki reprezentujące dostępność „stanowej" konfiguracji adresów (patrz rozdział 6.) i zalecenie unikania autokonfiguracji (*M*), dostępność dodatkowych informacji w ramach konfiguracji stanowej (*O*) i zdolność routera do pełnienia funkcja agenta domowego (*H*). Pole *Preferencja* określa stopień preferencji w odniesieniu do pełnienia przez router roli domyślnego routera: 01 oznacza podwyższoną preferencję, 00 preferencję domyślną, 11 preferencją obniżoną; wartość 10 jest zarezerwowana i nie powinna być używana. Szczegółowy opis wymienionych znaczników znajduje się w dokumencie [RFC4191]. Bit *P* (*Proxy*) związany jest z eksperymentalnym mechanizmem IPv6 *ND Proxy* (patrz [RFC4389]), stanowiącym odpowiednik opisywanego w rozdziale 6. mechanizmu *ARP-proxy*.

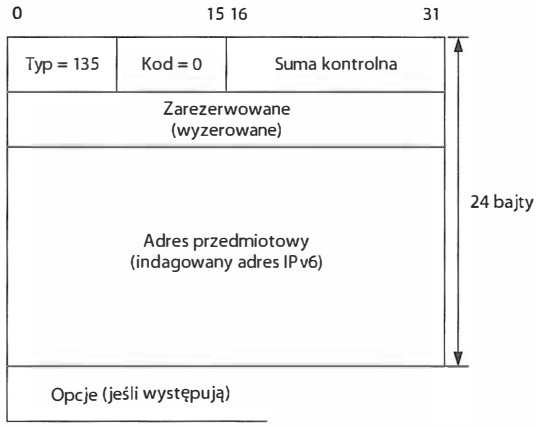
Pole *Czas ważności routera* związane jest z funkcją routera domyślnego jako następnego przeskoku i stanowi ze strony routera nadawcy deklarację czasu (w sekundach), przez który skłonny jest on tę funkcję pełnić; wartość 0 oznacza, że router takiej gotowości w ogóle nie sygnalizuje. Pole to dotyczy wyłącznie wspomnianej funkcji i nie ma związku z innymi opcjami danego komunikatu.

Wartość w polu *Czas osiągalności* określa (w milisekundach) czas, przez jaki węzeł komunikujący się z innym węzłem ma prawo milcząco zakładać jego osiągalność. Pole to wykorzystywane jest przez mechanizm wykrywania nieosiągalności sąsiada, który opisujemy w punkcie 8.5.4. Pole *Stoper retransmisji* określa interwał czasowy (w milisekundach), jaki powinien upłynąć między wysyłaniem przez host kolejnych komunikatów ND.

Komunikat RA może również zawierać opcję *Source Link-Layer*, a także opcję *MTU*, jeśli na łączu obowiązuje zmienna wartość MTU. Router powinien umieszczać w komunikacie także opcję *Prefix Information* wskazującą prefiksy wykorzystywane na lokalnym łączu. Przykład wykorzystywania komunikatów RS i RA widzieliśmy już w rozdziale 6., na rysunkach 6.24 i 6.25.

8.5.2. Komunikaty ICMPv6 Neighbor Solicitation i Neighbor Advertisement (typy 135 i 136)

Komunikat ICMPv6 *Neighbor Solicitation* (NS) jest odpowiednikiem żądań ARP w IPv4 — jego głównym zadaniem jest konwersja adresów IPv6 na adresy warstwy łącza danych, lecz wykorzystywany jest także do badania osiągalności pobliskich węzłów oraz możliwości nawiązywania z nimi komunikacji dwukierunkowej. Format komunikatu przedstawiamy na rysunku 8.34.

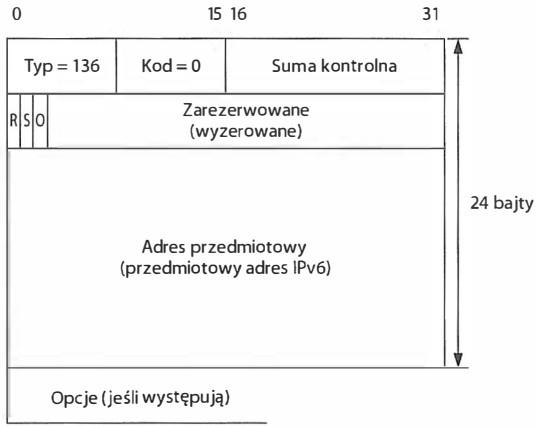


Rysunek 8.34. *Komunikat ICMPv6 Neighbor Solicitation podobny jest do komunikatu RS, zawiera jednak dodatkowo przedmiotowy adres IPv6. Wysyłany jest na adres multicast o prefiksie ff02::1:ff/104 w przypadku mapowania adresów analogicznie do ARP lub na adres unicast konkretnego węzła w celu zbadania jego osiągalności. Zawiera opcję Source Link-Layer Address*

W celu dokonania mapowania adresów komunikat NS wysyłany jest na adres multicast odpowiadający *Adresowi przedmiotowemu*, stanowiący kombinację prefiksu `ff02::1:ff/104` oraz 24 najmniej znaczących bitów adresu IPv6 indagowanego węzła (o szczegółach piszemy w rozdziale 9.). Komunikat NS wykorzystywany jest także do badania osiągalności konkretnych węzłów, wysyłany jest wtedy na adres unicast konkretnego węzła.

Centralną częścią komunikatu NS jest adres IPv6, dla którego nadawca chce uzyskać odpowiadający adres warstwy łącza danych. Komunikat może zawierać opcję *Source Link-Layer Address* — opcja ta jest konieczna, gdy w sieci wykorzystującej adresowanie warstwy łącza danych pakiet ND wysyłany jest na adres multicast; opcja powinna pojawiać się także w pakietach ND wysyłanych na adres unicast. Jeśli nadawca komunikatu specyfikuje adres nieokreślony w charakterze adresu źródłowego (np. w procedurze weryfikowania unikatowości adresu), opcja ta nie jest wymagana.

Komunikat ICMPv6 *Neighbor Advertisement* (NA), widoczny na rysunku 8.35, stanowi na gruncie IPv6 ekwiwalent odpowiedzi protokołu ARP z IPv4, spełnia ponadto pomocniczą rolę w procesie wykrywania nieosiągalności sąsiadów (o czym piszemy w punkcie 8.5.4). Może stanowić odpowiedź na komunikat NS, może też zostać wysłany asynchronicznie z inicjatywy węzła zmieniającego swój adres IPv6. Wysyłany jest bądź to na adres unicast indagującego węzła, bądź na adres grupowy wszystkich węzłów (*All Nodes* — `ff02::1`), jeśli indagujący węzeł podaje adres nieokreślony jako źródłowy.



Rysunek 8.35. *Komunikat ICMPv6 Neighbor Advertisement zawiera następujące znaczniki: R oznaczający, że nadawcą jest router, S oznaczający, że komunikat jest odpowiedzią na indagowanie, oraz O oznaczający, że informacja zawarta w komunikacie powinna zastąpić ewentualnie istniejące w cache odwzorowania dla odnośnych adresów. Adres przedmiotowy jest adresem IPv6 nadawcy komunikatu (zwykle adresem indagującego węzła, który uprzednio wysłał komunikat NS). Załączona do komunikatu opcja Target Link-Layer Address realizuje w IPv6 funkcję analogiczną do protokołu ARP*

Ustawiony bit *R* (*Router*) w polu znaczników wskazuje, że nadawca komunikatu jest routerem; wyzerowanie tego bitu jest sygnałem, że nadawca zrzekł się właśnie funkcji routera i stał się zwykłym hostem. Ustawienie bitu *S* (*Solicited*) oznacza, że komunikat jest odpowiedzią na wcześniejszy komunikat NS, co świadczy o możliwości nawiązywania łączności dwukierunkowej między węzłami. Ustawienie znacznika *O* (*override*) nakazuje usunięcie poprzednio uzyskanej i cache'owanej informacji dotyczącej adresów zawartych w komunikacie; bit *O* powinien być wyzerowany w komunikatach NA stanowiących odpowiedzi na komunikaty NS, komunikatach dotyczących adresów anycast oraz w ogłoszeniach routera pełniącego funkcję proxy dla innych routerów (w RFC 2461 komunikaty takie nazywane są *solicited proxy advertisements*); powinien być natomiast ustawiony we wszystkich innych komunikatach NA.

Gdy komunikat NA jest odpowiedzią na żądanie NS, *Adres przedmiotowy* jest adresem, dla którego poszukiwany jest odpowiadający adres warstwy łącza danych; gdy komunikat wysyłany jest z inicjatywy węzła zmieniającego adres IPv6, *Adres przedmiotowy* jest nowym adresem IPv6 tegoż węzła. Jeśli komunikat wysyłany jest na adres multicast w sieciach obsługujących rozgłaszanie i multicasting, musi zawierać opcję *Target Link-Layer Address*.

8.5.2.1. Przykład

Teraz na prostym przykładzie zaprezentujemy funkcjonowanie komunikatów ICMPv6 *Echo*, w połączeniu z mechanizmem NDP. Rolę nadawcy pełni komputer z systemem Windows XP i włączoną implementacją IPv6, zaś w linuksowym serwerze będącym adresatem wysyłanych żądań włączona została funkcja śledzenia pakietów; w załączonym raporcie pominęliśmy niektóre fragmenty dla lepszej czytelności.

```
C:\> ping6 -s fe80::210:18ff:fe00:100b fe80::211:11ff:fe6f:c603
Badanie fe80::211:11ff:fe6f:c603 z fe80::210:18ff:fe00:100b z 32 bajtami danych:

Odpowiedź z fe80::211:11ff:fe6f:c603: bajtów=32 czas<1ms
Odpowiedź z fe80::211:11ff:fe6f:c603: bajtów=32 czas<1ms
Odpowiedź z fe80::211:11ff:fe6f:c603: bajtów=32 czas<1ms
Odpowiedź z fe80::211:11ff:fe6f:c603: bajtów=32 czas<1ms

Statystyka badania ping dla fe80::211:11ff:fe6f:c603:
    Pakiety: Wysłane = 4, Odebrane = 4, Utracone = 0
             (0% straty),
Szacunkowy czas błądzenia pakietów w milisekundach:
    Minimum = 0ms, Maksimum = 0ms, Czas średni = 0ms

Linux# tcpdump -i eth0 -s1500 -vv -p ip6
tcpdump: listening on eth0,
         link-type EN10MB (Ethernet), capture size 1500 bytes

1 21:22:01.389656 fe80::211:11ff:fe6f:c603 > ff02::1:ff00:100b:
                [icmp6 sum ok] icmp6: neighbor sol: who has
                                      fe80::210:18ff:fe00:100b
                                      (src lladdr: 00:11:11:6f:c6:03)
                                      (len 32, hlim 255)
2 21:22:01.389845 fe80::210:18ff:fe00:100b > fe80::211:11ff:fe6f:c603:
                [icmp6 sum ok] icmp6: neighbor adv: tgt is
                                      fe80::210:18ff:fe00:100b(SO)
                                      (tgt lladdr: 00:10:18:00:10:0b)
                                      (len 32, hlim 255)
3 21:22:02.390713 fe80::210:18ff:fe00:100b > fe80::211:11ff:fe6f:c603:
                [icmp6 sum ok] icmp6: echo request seq 18
                                      (len 40, hlim 128)
4 21:22:02.390780 fe80::211:11ff:fe6f:c603 > fe80::210:18ff:fe00:100b:
                [icmp6 sum ok] icmp6: echo reply seq 18
                                      (len 40, hlim 64)

...
```

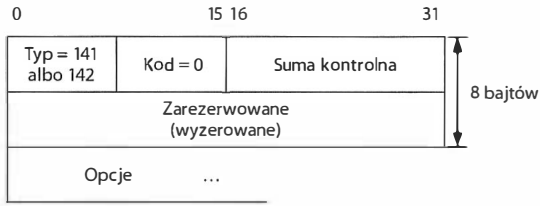
Program `ping6` dostępny jest zarówno w Windows XP, jak i w Linuksie; w nowszych wersjach Windows włączono do programu `ping` funkcjonalność IPv6. Parametr wywołania `-s` wskazuje adres źródłowy umieszczany w generowanych pakietach *Echo* — jak pamiętamy, hostowi można przydzielić wiele adresów IPv6, użytkownik uruchamiający program `ping` może więc wybrać jeden z nich; na potrzeby naszego przykładu wybraliśmy adres `fe80::211:11ff:fe6f:c603` lokalny dla łącza. Działający na serwerze program `tcpdump` uwidacznia wymianę komunikatów NS/NA oraz par żądanie-odpowiedź komunikatów *Echo*. Zauważmy, że każdy z komunikatów ND ma w polu *Limit przeskoków* (`hlim`) wartość 255, co wcześniej wyjaśnialiśmy; komunikaty *Echo* mają w tym polu wartości 64 i 128.

Komunikat NS (pakiet 1.) wysyłany jest na adres `ff02::1:ff00:100b`, który jest adresem multicast dla indagowania adresu `fe80::210:18ff:fe00:100b`, utworzonym jako konkatenacja prefiksu `ff02::1:ff/104` i 24 najmłodszych bitów indagowanego adresu (`00:100b`). Załączona opcja *Source Link-Layer Address* niesie ze sobą adres warstwy łącza danych nadawcy `00:11:11:6f:c6:03`.

Komunikat odpowiedź NA wysyłany jest do indagującego węzła przy użyciu adresu łącza danych i adresu IP — obu w trybie unicast. W polu *Adres przedmiotowy* znajduje się kopia adresu z żądania NS (`fe80::210:18ff:fe00:100b`), widzimy ponadto wyzerowany bit *R* (odpowiadający węzeł nie jest routerem) i ustawiony bit *O* (co daje pierwszeństwo informacji niesionej przez komunikat przed innymi, posiadanymi przez indagujący węzeł, informacjami dotyczącymi tego samego adresu IPv6). Ustawiony bit *S* sygnalizuje, że komunikat jest odpowiedzią na wcześniejsze żądanie NS, a nie asynchroniczną inicjatywą węzła nadawcy. W ramach opcji *Target Link-Layer Address* znajduje się najważniejsza informacja, o którą indagowany był węzeł nadawca — adres warstwy łącza danych `00:10:18:00:10:0b`.

8.5.3. Komunikaty ICMPv6 Inverse Neighbor Discovery Solicitation/Advertisement (typy 141 i 142)

Opisywany w [RFC3122] mechanizm odwrotnego odnajdywania sąsiadów (*Inverse Neighbor Discovery* — IND) realizuje mapowanie odwrotne w stosunku do ND — odnajdywanie adresu IPv6 węzła o podanym adresie warstwy łącza danych. Swą genealogię wywodzi od podobnego mechanizmu IPv4 o nazwie *Odwrotny ARP*, wykorzystywanego głównie do konfigurowania bezdyskowych stacji roboczych. Na rysunku 8.36 widoczny jest format obu komunikatów IND — żądania (*IND Solicitation* — typ 141) i odpowiedzi (*IND Advertisement* — typ 142).

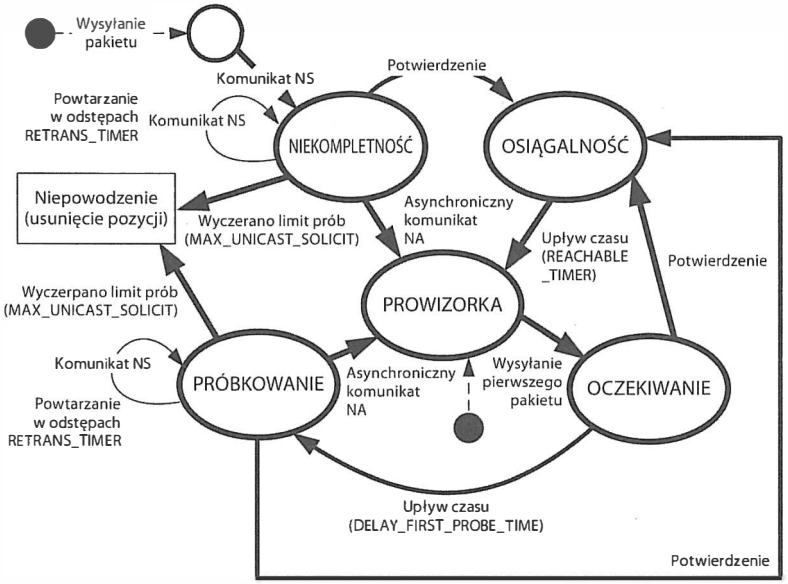


Rysunek 8.36. *Format komunikatów IND Solicitation (typ 141) i IND Advertisement (typ 142), związanych z mapowaniem adresu warstwy łącza danych na adres IPv6*

Komunikat *IND Solicitation* wysyłany jest na adres grupowy wszystkich węzłów (*All Nodes* — `ff02::1`), lecz opatrzony tym adresem pakiet warstwy sieciowej enkapsulowany jest w ramce łącza danych opatrzonej adresem unicast konkretnego węzła. Komunikat ten musi zawierać opcje *Source Link-Layer Address* i *Destination Link-Layer Address*, zawierać może także opcje *Source/Target Address List* i *MTU*.

8.5.4. Wykrywanie nieosiągalności sąsiadów (NUD)

Jednym z istotnych elementów ND jest wykrywanie przypadków utraty wzajemnej osiągalności dwóch systemów na tym samym łączu — utraty całkowitej lub tylko w jednym kierunku (asymetrycznej). Odpowiedzialny za to algorytm nosi nazwę *wykrywania nieosiągalności sąsiadów* (*Neighbor Unreachability Detection* — NUD) i opiera się na utrzymywanej w każdym węźle *pamięci podręcznej sąsiadów* (*neighbor cache*), stanowiącej analogię cache protokołu ARP (któremu poświęciliśmy rozdział 4.). Ta pamięć podręczna stanowi (przynajmniej pod względem koncepcyjnym) strukturę odwzorowującą pewien zbiór adresów IPv6 na odpowiadające im adresy warstwy łącza danych, wraz z informacją o stanie poszczególnych odwzorowań. Odwzorowanie to jest niezbędne do realizacji bezpośredniego dostarczania (*direct delivery* — patrz rozdział 5.) datagramów IPv6 między węzłami operującymi na tym samym łączu. Na rysunku 8.37 widoczne jest funkcjonowanie algorytmu NUD w ujęciu maszyny stanów.



Rysunek 8.37. *Maszyna stanów algorytmu wykrywania nieosiągalności sąsiadów (NUD). Każde odwzorowanie może znajdować się w danej chwili w jednym z pięciu stanów. Potwierdzenie osiągalności węzła następuje bądź to za pomocą komunikatów NA i RA (asynchronicznych lub stanowiących odpowiedź na komunikat NS), bądź też za pomocą informacji pochodzącej od protokołów warstw wyższych*

Każde z indywidualnych odwzorowań, dotyczących poszczególnych adresów, może znajdować się w jednym z pięciu stanów: NIEKOMPLETNOŚĆ, OSIĄGALNOŚĆ, PROWIZORKA, OCZEKIWANIE lub PRÓBKOWANIE. Jak wynika z rysunku 8.37, stanem początkowym może być NIEKOMPLETNOŚĆ lub PROWIZORKA. Gdy węzeł IPv6 staje przed zadaniem wysłania datagramu na adres unicast, algorytm wchodzi w stan NIEKOMPLETNOŚĆ. Gdy w pamięci podręcznej znalezione zostanie odwzorowanie dotyczące wspomnianego adresu, algorytm przechodzi w stan OSIĄGALNOŚĆ i datagram zostaje natychmiast wysłany za pomocą dostar-

czenia bezpośredniego. Gdy odwzorowania tego brak, lecz docelowy adres wydaje się być poprawny w kontekście bieżącego łącza, węzeł wysyła komunikat NS i po otrzymaniu potwierdzającego komunikatu NA algorytm przechodzi do stanu OSIĄGALNOŚĆ, a węzeł wysyła przedmiotowy datagram. Gdy jednak komunikat NA nie nadejdzie w żądanym czasie, odwzorowanie dla danego adresu IPv6 uważane jest za tymczasowo niewykonalne; algorytm kilkakrotnie ponawia próbę uzyskania wspomnianego komunikatu i w przypadku niepowodzenia ostatecznie się poddaje.

Jak pamiętamy, komunikaty NA mogą być generowane nie tylko w odpowiedzi na żądania NS, lecz także asynchronicznie, z inicjatywy nadawców. Gdy któryś węzeł odbierze taki „niezamówiony" komunikat, wykorzystuje go do odświeżenia zawartości swej pamięci podręcznej odwzorowań: pozycja dotycząca adresu IPv6 (lub adresu łącza danych) zawartego w komunikacie zostaje uaktualniona i (o ile nie znajduje się w stanie OSIĄGALNOŚĆ) zostaje wprowadzona w stan PROWIZORKA (być może po uprzednim utworzeniu, jeśli nie było jej w pamięci podręcznej). Generalnie stan PROWIZORKA reprezentuje sytuację, gdy prawdopodobnie poprawne odwzorowanie nie zostało uprawomocnione stosownym potwierdzeniem. Określone odwzorowanie może przejść w ten stan nie tylko wskutek opisanego nadejścia asynchronicznego komunikatu NA, lecz również w konsekwencji wyczerpania się limitu czasowego REACHABLE_TIMER przebywania tej pozycji w stanie OSIĄGALNOŚĆ.

Gdy ma zostać użyte odwzorowanie adresu znajdujące się aktualnie w stanie PROWIZORKA, algorytm przechodzi do tymczasowego stanu OCZEKIWANIE, w którym przebywa przez czas określony w parametrze DELAY_FIRST_PROBE_TIME; w tym czasie protokoły warstw wyższych mają szansę na dostarczenie własnych dowodów osiągalności węzła docelowego. Otrzymanie takiego dowodu przez algorytm powoduje awansowanie odwzorowania do stanu OSIĄGALNOŚĆ, nieotrzymanie powoduje jego przejście w stan PRÓBKOWANIE. W stanie PRÓBKOWANIE algorytm periodycznie wysyła komunikaty NS, w odstępach określonych przez parametr RETRANS_TIMER; jeśli nie uzyska odpowiedzi, mimo wyczerpania limitu prób (określonego przez parametr MAX_UNICAST_SOLICIT), algorytm poddaje się, a odnośna pozycja odwzorowania zostaje usunięta z pamięci podręcznej. Ze stanu PRÓBKOWANIE algorytm może ponadto wyjść na dwa inne sposoby: otrzymanie wspomnianej odpowiedzi wprowadza odwzorowanie w stan OSIĄGALNOŚĆ, natomiast nadejście asynchronicznego komunikatu NA wprowadza odwzorowanie w stan PROWIZORKA.

8.5.5. Bezpieczne odnajdywanie sąsiadów (SEND)

SEND to definiowany w [RFC3971] zbiór rozszerzeń ND wprowadzających zabezpieczenia do jego komunikatów, w celu zabezpieczenia tych komunikatów przed próbami oszustw wynikających z podszywania się intruzywnych hostów lub routerów pod działające legalnie i wysyłanie podrabianych komunikatów NA w imieniu tychże (piszemy na ten temat w podrozdziale 8.6 i rozdziale 18., tematyce tej poświęcony jest także dokument [RFC3756]). SEND nie wykorzystuje funkcji IPsec (opisywanych w rozdziale 18.), posługując się w zamian własnym, specyficznym mechanizmem bezpieczeństwa używanym również do ochrony przełączania ruchu w ramach FMIPv6 (patrz [RFC5269]).

SEND funkcjonuje w określonym środowisku, rozumianym jako spełnienie kilku podstawowych założeń. Po pierwsze, każdemu routerowi zaangażowanemu w realizację SEND musi być przyporządkowany *certyfikat*, czyli forma kryptograficznego uwierzytelniania,

za pomocą której router ten udowadnia swą autentyczność wobec hosta, z którym nawiązuje połączenie. Po drugie, host musi być, rzecz jasna, wyposażony w mechanizmy konfiguracyjne zdolne do weryfikacji rzeczonego certyfikatu — ogół takich informacji nosi fachową nazwę *kotwicy zaufania* (*trust anchor*). Po trzecie wreszcie, każdy węzeł, w procesie konfigurowania używanych adresów IPv6, konfiguruje jednocześnie parę komplementarnych kluczy prywatny-publiczny. Szczegółami wymienionych elementów — certyfikatami, kotwicami zaufania, kluczami kryptograficznymi — i innymi kryptograficznymi elementami bezpieczeństwa zajmiemy się szczegółowo w rozdziale 18. Generowanie wspomnianych kluczy odbywa się całkowicie w sposób autonomiczny, bez angażowania infrastruktury kluczy publicznych (PKI) lub innego niezależnego podmiotu zaufania.

8.5.5.1. Adresy generowane kryptograficznie (CGA)

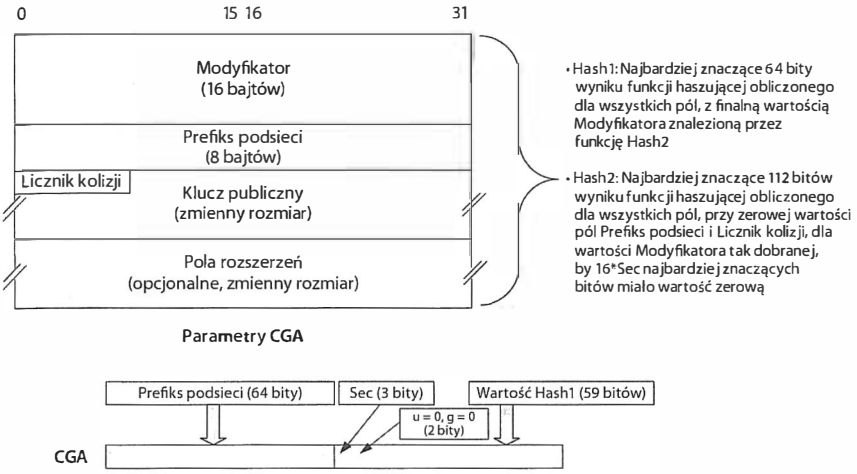
Najbardziej bodaj interesującą cechą SEND jest specyfika adresów IPv6, jakich mechanizm ten używa do swych celów. Adresy te, ze względu na swą genealogię nazywane *adresami generowanymi kryptograficznie* (*cryptographically generated addresses*, w skrócie CGA), opisywane szczegółowo w dokumentach [RFC3972], [RFC4581] i [RFC4982], wywodzą swą postać od kluczy publicznych odnośnych węzłów. W związku z tym informacja uwierzytelniająca dany węzeł zakodowana zostaje wprost w jego adresie, dzięki czemu węzeł ów może udowodnić legalność posługiwania się swym adresem CGA, używając do tego celu swego klucza prywatnego. W adresie CGA zakodowany jest także prefiks podsieci, do której przynależy wspomniany węzeł, adres ten nie jest więc przenaszalny w prosty sposób między podsieciami. Kryptograficzne generowanie adresów IPv6 jest procesem istotnie różnym od konfigurowania stanowego i bezstanowego, które opisywaliśmy w rozdziale 6.

Adres CGA powstaje przez skonkatenowanie 64-bitowego prefiksu podsieci ze specjalnie skonstruowanym identyfikatorem interfejsu — właśnie identyfikator jest głównym obiektem operacji kryptograficznych CGA, polegających na zastosowaniu bezpiecznych funkcji haszujących (*secure hash function*); funkcja haszująca uważana jest za bezpieczną, jeśli jej odwracanie[1] nie jest wykonalne obliczeniowo przy użyciu rozsądnych środków. Używane są dwie funkcje haszujące — które nazwiemy umownie *Hash1* i *Hash2* — co rozszerza bezpieczeństwo haszowania i sprawia, że jest m.in. bardziej odporne na ataki intruza próbującego dla danego x znaleźć wartość x' spełniającą równość $H(x) = H(x')$ (patrz [A03] i [RFC6273]). Informację wejściową dla haszowania stanowi klucz publiczny węzła wraz z innymi parametrami, tworzącymi strukturę przedstawioną na rysunku 8.38. Opcjonalne *Pole rozszerzające* nie pełni obecnie żadnej roli i przeznaczone jest do ewentualnego wykorzystania w przyszłości (patrz [RFC4581]).

Zależnie od konkretnych potrzeb można różnicować uzyskiwany stopień bezpieczeństwa, za cenę zróżnicowanej złożoności obliczeniowej. Wybór jednego z kilku (obecnie trzech) wariantów reprezentowany jest przez 3-bitowy parametr nazwany *Sec*. Jego wartości definiowane są centralnie przez IANA, ich znaczenie wyjaśnione jest w dokumencie [RFC4581].

---

[1] Funkcja haszująca z definicji nie jest funkcją różnowartościową, nie jest więc odwracalna w ścisłym matematycznym sensie. W sensie potocznym przez odwracanie funkcji H rozumiemy znajdowanie *jakiejkolwiek* wartości x, dla której zachodzi równość $H(x) = y$ dla ustalonego y. Trudność obliczeniowa takiego zadania — czyli jego niewykonalność przy użyciu czasu i zasobów w zakresie uzasadnionym opłacalnością przedsięwzięcia — stanowi właśnie jeden z filarów (choć nie jedyny) zabezpieczeń opartych na funkcjach haszujących — *przyp. tłum.*



Rysunek 8.38. *Adres CGA i towarzysząca mu struktura parametryczna. Wartości pól tej struktury używane są jako dane wejściowe dla dwóch kryptograficznych funkcji haszujących Hash1 i Hash2. Wynik funkcji Hash2 musi mieć wyzerowane 16*Sec początkowych bitów (Sec jest 3-bitową liczbą całkowitą) — obliczenia powtarzane są dla kolejno zwiększanych o 1 wartości Modyfikatora, aż do spełnienia tego warunku. Znaleziona wartość Modyfikatora używana jest do obliczenia wyniku funkcji Hash1, który w połączeniu z parametrem Sec i prefiksem podsieci daje finalny adres CGA*

Obliczenia rozpoczynają się od wpisania losowej wartości do pola *Modyfikator* i wyzerowania pól *Licznik kolizji* oraz *Prefiks podsieci*. Obliczana jest funkcja *Hash2*, po czym sprawdza się, czy najbardziej znaczące bity wyniku w liczbie 16**Sec* są wszystkie zerowe; jeśli nie, obliczenia są powtarzane dla kolejno zwiększanych wartości *Modyfikatora* — aż do skutku, czyli spełnienia rzeczonego warunku. Złożoność tych obliczeń kształtuje się na poziomie $O(2^{16*Sec})$, lecz wykonywane są one tylko raz dla danego adresu.

Po znalezieniu odpowiedniego *Modyfikatora* pole *Prefiks podsieci* wypełniane jest właściwą wartością i obliczana jest funkcja *Hash1*. 59-bitowy wynik zostaje użyty w roli 59 najmniej znaczących bitów identyfikatora interfejsu. Trzy najbardziej znaczące ustawiane są na wartość *Sec*, dwa pozostałe zostają natomiast wyzerowane, zgodnie ze znaczeniem bitów *u* i *g* wyjaśnionym w punkcie 2.3.6 tej książki. 64 najbardziej znaczące bity adresu CGA tożsame są z prefiksem podsieci.

Może się zdarzyć, że wygenerowany adres już istnieje w sieci (co niezawodnie sprawdzić można za pomocą procedury DAD opisywanej w rozdziale 6.). W takiej sytuacji pole *Licznik kolizji* zwiększone zostaje do wartości 1 i funkcja *Hash1* obliczana jest ponownie. Gdy i tym razem zdarzy się kolizja adresu, pole *Licznik kolizji* zwiększone zostaje do wartości 2 i znowu funkcja *Hash1* obliczana jest ponownie. Wystąpienie kolizji i tym razem świadczy najprawdopodobniej o błędzie konfiguracji lub trwającym ataku, bo wykorzystywanie algorytmu SHA-1 do obliczania funkcji *Hash1* sprawia, że wysoce nieprawdopodobne jest uzyskanie trzech kolizji z rzędu.

Zauważmy, że gdy zmienia się prefiks podsieci, trzeba ponownie obliczyć tylko funkcję *Hash1*. Nie jest konieczne ponowne obliczanie funkcji *Hash2*, ponieważ jest ona obliczana przy zerowej wartości pola *Prefiks podsieci*; raz znaleziony właściwy *Modyfikator* zachowuje więc swoją aktualność.

Przy okazji proponujemy zainteresowanym czytelnikom lekturę dokumentu [RFC5535], opisującego adresy oparte na haszowaniu (*Hash-Based Addresses*, w skrócie HBA). Są one wykorzystywane przez hosty multihomed, wykorzystujące wiele prefiksów w zróżnicowanych kontekstach. W porównaniu z adresami CGA wymagają mniejszych nakładów obliczeniowych. We wspomnianym dokumencie omawiana jest także interesująca kwestia kompatybilności CGA i HBA.

Zobaczmy teraz, na czym polega bezpieczeństwo uzyskiwane w wyniku zastosowania opisanej procedury generowania adresów CGA. Weryfikacja posiadanego przez węzeł adresu CGA obejmuje dwa elementy: sprawdzenie poprawności samego adresu CGA oraz uwiarygodnienie legalności posługiwania się tym adresem przez węzeł. Weryfikatorowi znany jest zarówno adres CGA, jak i struktura parametryczna z rysunku 8.38. Adres zostaje uznany za poprawny, jeżeli (wraz ze wspomnianą strukturą) spełnia łącznie następujące warunki:

- pole *Licznik kolizji* ma wartość nie większą niż 2,
- prefiks podsieci zawarty w adresie jest identyczny z polem *Prefiks podsieci*,
- wynik funkcji *Hash1* obliczony dla bieżącej postaci struktury z rysunku 8.38 jest identyczny z 59 najmniej znaczącymi bitami adresu,
- w wyniku funkcji *Hash2* obliczonym dla wspomnianej struktury, po uprzednim wyzerowaniu pól *Licznik kolizji* i *Prefiks podsieci*, najbardziej znaczące 16**Sec* bitów ma wartość 0 (parametr *Sec* odczytany zostaje bezpośrednio z adresu).

Weryfikacja poprawności adresu CGA wymaga więc jedynie dwukrotnego obliczania funkcji haszującej, zwykle przebiega zatem szybciej niż generowanie wspomnianego adresu. Zauważmy, że nie podlegają weryfikacji zawarta w adresie wartość *Sec* oraz ustawienie bitów *g* i *u*.

W celu udowodnienia legalności posługiwania się danym adresem CGA węzeł będący właścicielem tegoż adresu wysyła komunikat zawierający m.in. sygnaturę (podpis cyfrowy) uzależnioną od swego klucza prywatnego, komplementarnego z kluczem publicznym użytym do wygenerowania rzeczonego adresu. Weryfikator tworzy strukturę kontrolną, poprzedzając otrzymany komunikat specjalnym 128-bitowym znacznikiem, po czym weryfikuje komunikat w roli świadectwa legalności, sporządzając podpis cyfrowy RSA (RSASSA-PKCS1-v1_5, patrz [RFC3447]) na podstawie klucza publicznego (odczytanego ze struktury parametrycznej), bloku danych zawartego we wspomnianym komunikacie i parametrów algorytmu podpisywania.

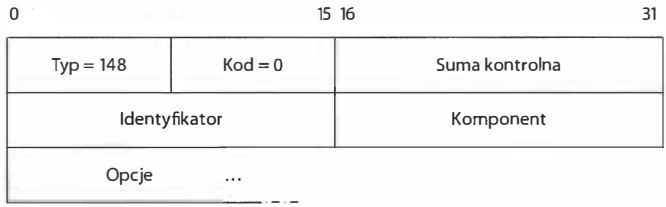
Generalnie, adres CGA używany przez węzeł zaakceptowany zostaje przez weryfikator po pozytywnym rezultacie obu opisanych testów.

Zarządzanie adresami CGA i ich weryfikowanie odbywa się przy użyciu dwóch komunikatów ICMPv6 i sześciu opcji zdefiniowanych w dokumencie [RFC3971]). Dokument ten definiuje także (kontrolowane przez IANA) wartości pól *Name Type* w opcji *Trust Anchor* oraz *Cert Type* w opcji *Certificate* (patrz podpunkt 8.5.6.13). W dokumencie [RFC3972] definiowany jest natomiast *Typ komunikatu* CGA, ze 128-bitową wartością `0x086FCA5E10B200C99C8CE00164277C08` reprezentującą SEND (obok czterech innych, dedykowanych innym zastosowaniom). Dla parametru *Sec* zdefiniowano obecnie trzy wartości

— 0, 1 i 2 — odpowiadające (kolejno) brakowi wymogu zerowych bitów na początku hasza, wymogowi 16 zerowych bitów i wymogowi 32 zerowych bitów; dla wszystkich przypadków algorytmem haszującym jest SHA-1. W sekcji 2. dokumentu [RFC4581] definiowany jest format rozszerzeń, w postaci „typ-rozmiar-wartość"; obecnie wykorzystywane jest tyko jedno z tych rozszerzeń, związane z kompatybilnością CGA i HBA, zdefiniowane w [RFC5535]. Opiszemy szczegółowo dwa wspomniane komunikaty, zaś wspomnianymi opcjami zajmiemy się przy ogólnym omawianiu opcji ND w następnym podpunkcie.

8.5.5.2. Komunikaty ICMPv6 Certification Path Solicitation/Advertisement (typy 148 i 149)

W celu ułatwienia hostom uzyskiwania informacji składających się na ścieżkę certyfikacji SEND definiuje komunikat indagujący (*Certification Path Solicitation* — typ ICMPv6 148) i ogłaszający (*Certification Path Advertisement* — typ ICMPv6 149). Format komunikatu ogłaszającego przedstawiamy na rysunku 8.39.

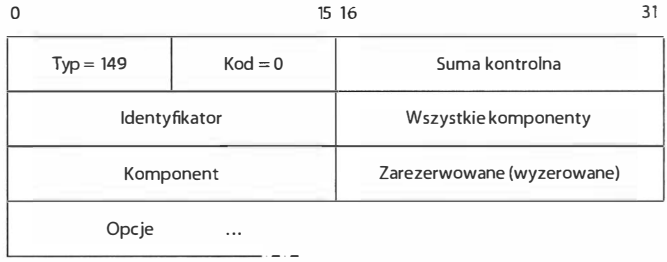


Rysunek 8.39. *Komunikat Certification Path Solicitation. W polu Komponent zawarty jest indeks określający pozycję żądanego komponentu w ścieżce certyfikacji; wartość 65535 oznacza żądanie wszystkich certyfikatów ścieżki zakotwiczonej w tożsamości identyfikowanej przez załączoną opcję Trust Anchor*

W polu *Identyfikator* znajduje się losowo wybrana wartość służącą do kojarzenia żądań z odpowiedziami. Wartość w polu *Komponent* jest indeksem wskazującym punkt na ścieżce certyfikacji, którego dotyczy żądanie; wartość 65535 (wszystkie bity ustawione na 1) oznacza żądanie dotyczące kompletnej ścieżki. Komunikat może zawierać opcję *Trust Anchor* (opisujemy ją w podpunkcie 8.5.6.12). Certyfikaty i ścieżki certyfikacji omawiamy szczegółowo w rozdziale 18.

Widoczny na rysunku 8.40 komunikat *Certification Path Advertisement* niesie ze sobą informacje dotyczące jednego komponentu (certyfikatu) ścieżki certyfikacji. Komunikat taki może stanowić odpowiedź na żądanie *Certification Path Solicitation*, wówczas *Identyfikator* jest kopią identyfikatora zawartego w żądaniu; komunikat ten może być także generowany periodycznie z inicjatywy routerów implementujących mechanizm SEND — wówczas wysyłany jest na adres grupowy wszystkich węzłów (*All Nodes* — `ff02::1`), a w polu *Identyfikator* znajduje się wartość 0.

Pole *Wszystkie komponenty* zawiera liczbę komponentów tworzących ścieżkę certyfikacji, łącznie z kotwicą zaufania. Ponieważ generalnie zalecane jest unikanie fragmentacji datagramów niosących komunikaty ND, komunikat *Certification Path Advertisement* zawiera informacje dotyczące tylko jednego komponentu (certyfikatu), identyfikowanego w polu *Komponent*, obecnego w ramach opcji *Certificate*. W odpowiedzi na żądanie

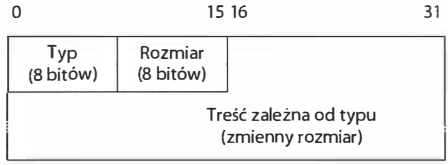


Rysunek 8.40. *Komunikat Certification Path Advertisement. Pole Identyfikator jest kopią identyfikatora odnośnego żądania, w polu Wszystkie komponenty znajduje się liczba wszystkich komponentów tworzących ścieżkę. Odpowiedź dotyczy tylko jednego komponentu, identyfikowanego przez pole Komponent, więc na żądanie dotyczące całej ścieżki może być generowana seria komunikatów odpowiedzi*

dotyczące całej ścieżki generowana jest więc seria takich komunikatów, po jednym dla każdego komponentu, przy czym zalecane jest wysyłanie poszczególnych komunikatów w kolejności malejących numerów komponentów — *N*-1, *N*-2, ..., 0. Komponent o najwyższym numerze (*N*) nie musi być wysyłany, bo jest już zawarty w opcji *Trust Anchor*.

8.5.6. Opcje komunikatów odnajdywania sąsiadów

Podobnie jak wiele protokołów rodziny IPv6, także protokół *Neighbor Discovery* definiuje zbiór standardowych nagłówków, a jego komunikaty zawierać mogą rozmaite opcje (niektóre z opcji występować mogą wielokrotnie). W niektórych komunikatach obowiązkowe wręcz jest wystąpienie określonych opcji. Ogólny format opcji ND przedstawiony jest na rysunku 8.41.



Rysunek 8.41. *Opcje ND są zmiennej długości; mają wspólny format w postaci „typ-rozmiar-treść", przy czym „rozmiar" obejmuje całość obszaru opcji, nie tylko jej „treść", i wyrażony jest w jednostkach 8-bajtowych. Rozmiar opcji musi więc być wielokrotnością 8 bajtów, w razie potrzeby stosuje się dopełnianie*

Pierwszy bajt każdej opcji identyfikuje jej typ, w drugim bajcie zawarta jest natomiast informacja o jej rozmiarze: rozmiar ten wyrażony jest w jednostkach 8-bajtowych i obejmuje cały obszar opcji, łącznie z polami *Typ* i *Długość*. Oczywiście, w tych warunkach rozmiar opcji musi być wielokrotnością 8 bajtów, więc „niewymiarowe" opcje dopełniane są do tej granicy. W tabeli 8.5 zestawiono 28 opcji standardowych i dwie eksperymentalne opcje zdefiniowane przez IANA (stan na 16 stycznia 2013 roku).

8.5.6.1. Opcje Source/Target Link-Layer Address (typy 1 i 2)

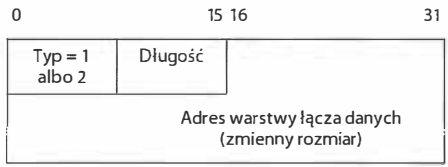
Format wspólny dla obu tytułowych opcji przedstawiliśmy na rysunku 8.42: po standardowych polach określający typ i rozmiar opcji następuje adres warstwy łącza danych.

Tabela 8.5. *Zdefiniowane przez IANA opcje protokołu ND*

Typ	Nazwa oficjalna	Dokument	Znaczenie
1	*Source Link-Layer Address*	[RFC4861]	Adres warstwy łącza danych nadawcy; opcja wykorzystywana w komunikatach NS, RS i RA
2	*Target Link-Layer Address*	[RFC4861]	Przedmiotowy adres warstwy łącza danych; opcja wykorzystywana w komunikatach NA i *Redirect*
3	*Prefix Information*	[RFC4861][RFC6275]	Prefiks lub adres IPv6; opcja wykorzystywana w komunikatach RA
4	*Redirected Header*	[RFC4861]	Część oryginalnego datagramu IPv6; opcja wykorzystywana w komunikatach *Redirect*
5	*MTU*	[RFC4861]	Zalecana wartość MTU; opcja wykorzystywana w komunikatach *IND Advertisement*
6	*NMBA Shortcut Limit*	[RFC2491]	Limit przeskoków dla tzw. prób na skróty (*shortcut attempt*) w sieci NMBA; opcja wykorzystywana w komunikatach NS
7	*Advertisement Interval*	[RFC6275]	Interwał czasowy rozdzielający kolejne komunikaty RA wysyłane z inicjatywy routera. Opcja wykorzystywana w komunikatach RA
8	*Home Agent Information*	[RFC6275]	Wskaźnik preferencji i okresu pełnienia funkcji agenta domowego przez router; opcja wykorzystywana w komunikatach RA z ustawionym bitem *H*
9	*Source Address List*	[RFC3122]	Adresy hostów; opcja wykorzystywana w komunikatach IND
10	*Target Address List*	[RFC3122]	Adresy przedmiotowe; opcja wykorzystywana w komunikatach IND
11	*CGA*	[RFC3971]	Adres CGA; opcja wykorzystywana w komunikatach SEND
12	*RSA Signature*	[RFC3971]	Uwierzytelnienie podpisu cyfrowego hosta w protokole SEND
13	*Timestamp*	[RFC3971]	Znacznik czasowy zabezpieczający przed atakami powtarzania komunikatów SEND
14	*Nonce*	[RFC3971]	Unikatowa wartość losowa zabezpieczająca przed atakami powtarzania komunikatów SEND
15	*Trust Anchor*	[RFC3971]	Typ uwierzytelniania (SEND)
16	*Certificate*	[RFC3971]	Zakodowany certyfikat (SEND)
17	*IP Address/Prefix*	[RFC5568]	Adres przekierowania lub adres NAR; opcja wykorzystywana w komunikatach PrRtAdv FMIPv6

Tabela 8.5. *Zdefiniowane przez IANA opcje protokołu ND — ciąg dalszy*

Typ	Nazwa oficjalna	Dokument	Znaczenie
19	*Link-Layer Address*	[RFC5568]	Żądany adres nowego punktu dostępowego lub węzła mobilnego; opcja wykorzystywana w komunikatach *RtSolPr* i *PrRtAdv* FMIPv6
20	*Neighbor Advertisement ACK*	[RFC5568]	Informacja dla węzła mobilnego o jego nowym adresie przekierowania; opcja wykorzystywana w komunikatach RA
24	*Route Information*	[RFC4191]	Prefiks trasy, lista preferowanych routerów
25	*Recursive DNS Server*	[RFC6106]	Adres IP serwera DNS; opcja dodawana do komunikatów RA
26	*RA Flags Extension*	[RFC5175]	Rozszerzenie przestrzeni dla znaczników RA
27	*Handover Key Request*	[RFC5269]	Żądanie klucza FMIPv6 dla SEND
28	*Handover Key Reply*	[RFC5269]	Udostępnienie klucza FMIPv6 dla SEND
31	*DNS Search List*	[RFC6106]	Lista poszukiwania nazw domen; opcja dodawana do komunikatów RA
32	*Proxy Signature (PS)*	[RFC6496]	Sygnatura kryptograficzna serwera proxy komunikatów ND
33	*Address Registration Option*	[RFC6775]	Opcje bezprzewodowych sieci prywatnych małej mocy (LoWPAN — *Low power Wireless Personal Area Networks*)
34	*6LoWPAN Context Option*	[RFC6775]	
35	*Authoritative Border Router Option*	[RFC6775]	
138	*CARD Request option*	[RFC4065]	Żądanie *Seamoby Candidate Access Router Discovery* (CARD)
139	*CARD Reply option*	[RFC4065]	Odpowiedź *Seamoby Candidate Access Router Discovery* (CARD)
253, 254	*Experimental*	[RFC4727]	Opcje eksperymentalne — typ ustalony w zgodzie z wytycznymi dokumentu [RFC3692]



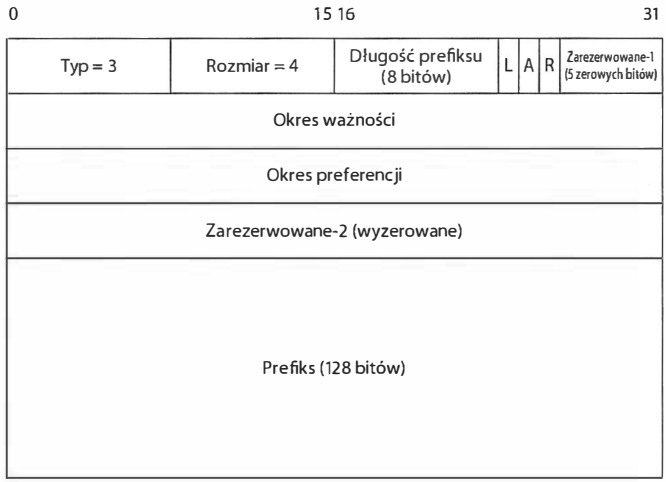
Rysunek 8.42. *Format opcji Source (typ 1) i Target (typ 2) Link Layer Address. Jeśli adresem warstwy łącza danych jest 48-bitowy adres ethernetowy, cała opcja ma rozmiar 8 bajtów i w polu Rozmiar znajduje się wartość 1*

Opcja *Source Link-Layer Address* (typ 1) oczekiwana jest w komunikatach ICMPv6 RS, NS i RA, jeśli komunikaty te przesyłane są w sieciach realizujących adresowanie na poziomie warstwy łącza danych. Opcja ta zawiera adres źródłowy warstwy łącza danych nadawcy komunikatu. Jeśli węzeł posługuje się kilkoma różnymi adresami, w wysyłanych przez niego komunikatach wymienionych kategorii opisywana opcja może występować wielokrotnie.

Opcja *Target Link-Layer Address* (typ 2) załączona jest do komunikat NA stanowiącego odpowiedź na komunikat NS. Opcja ta używana jest też zwykle w komunikatach *Redirect* — obowiązkowo musi wystąpić w tych komunikatach, gdy są przesyłane w sieciach NBMA.

8.5.6.2. Opcja Prefix Information (typ 3)

Opcja ta, oznaczana niekiedy skrótem PIO, występująca w komunikatach RA i komunikatach *Mobile Prefix Advertisement*, zawiera prefiks adresu IPv6, a w niektórych przypadkach kompletny adres. Format opcji przedstawiono na rysunku 8.43.



Rysunek 8.43. *Opcja Prefix Information zawiera prefiks adresu IPv6, do wykorzystania na potrzeby lokalnej sieci. Hosty mogą stosować tę opcję do celów autokonfiguracji, jeśli ustawiony jest bit A. Ustawienie bitu L oznacza możliwość używania prefiksu do wykrywania węzłów „na łączu", natomiast ustawiony bit R oznacza, że w opcji znajduje się kompletny adres IPv6, nie tylko prefiks*

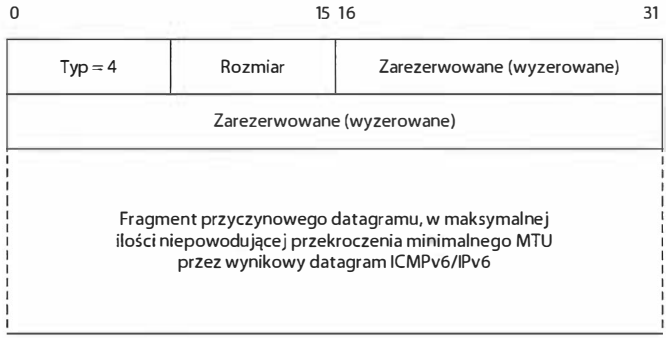
Gdy z danym komunikatem powiązanych jest kilka prefiksów lub adresów, w komunikacie tym opisywana opcja występuje wielokrotnie — od routera oczekuje się opcji PIO dla każdego używanego przez niego prefiksu. Gdy bit *R* ma wartość 1, w polu *Prefiks* znajduje się *kompletny adres* routera nadawcy; przy zerowej wartości bitu *R* po właściwym prefiksie następuje ciąg zerowych bitów. Wariant ten jest użyteczny w przypadku odnajdywania agenta domowego, Mobile IPv6 — routery ogłaszające gotowość do pełnienia tej funkcji muszą wysyłać komunikaty RA z ustawionym bitem *R* w przynajmniej jednym wystąpieniu opcji PIO. *Długość prefiksu* określa liczbę bitów faktycznie tworzących prefiks. Ustawienie bitu *L* oznacza, że zawarty w opcji prefiks może być użyty

do określania statusu „na łączu" (co wyjaśniamy w następnym akapicie), natomiast ustawiony bit *A* sygnalizuje przydatność wspomnianego prefiksu do autokonfiguracji (opisywanej w rozdziale 6.). Pola *Okres ważności* i *Okres preferencji* wyrażają (w sekundach) czas, w którym zawarty w opcji prefiks może być używany do (odpowiednio) określania statusu „na łączu" i autokonfiguracji adresów. Wartość `0xffffffff` oznacza czas nieskończony.

W IPv6 określenie „na łączu" odnosi się do tych węzłów, które osiągalne są w ramach dostarczania bezpośredniego (patrz rozdział 5.). W IPv4 są to węzły o tym samym prefiksie (numerze podsieci), różniące się jedynie numerami w ramach swej podsieci; jakkolwiek taka aranżacja adresów możliwa jest do zrealizowania także na gruncie IPv6, to jednak nie jest konieczna, zatem status on-link nigdy nie jest przyjmowany domyślnie, a jedynie na podstawie bitu *L* opisywanej opcji, zgodnie z [RFC5942]. Oczywiście, status ten można również określić za pomocą innych mechanizmów, m.in. protokołu DHCPv6, komunikatów *Redirect* czy też konfiguracji manualnej. Węzeł, co do którego nie udowodniono w sposób jawny statusu on-link, uważany jest za węzeł „poza łączem".

8.5.6.3. Opcja Redirected Header (typ 4)

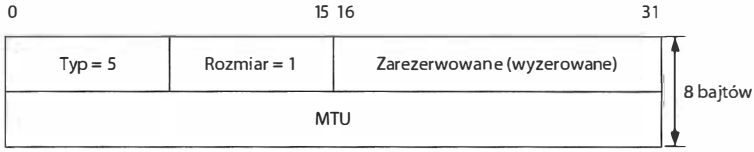
Opcja *Redirected Header* enkapsuluje część przyczynowego datagramu, który spowodował wysłanie komunikatu *Redirect*. Zależnie od wielkości owego datagramu enkapsulowana jest jego całość lub tylko początkowa część — co najmniej nagłówek, stąd nazwa opcji; jak pamiętamy, wielkość pakietu IP enkapsulującego komunikat ICMPv6 nie może być większa od ustalonej dla łącza wartości MTU i ten właśnie czynnik (jako jedyny) limituje wielkość załączanej porcji przyczynowego datagramu. Wystąpienie opisywanej opcji w komunikacie innym niż *Redirect* jest po prostu ignorowane. Format opcji — niewymagający dalszych komentarzy — przedstawiamy na rysunku 8.44.



Rysunek 8.44. *Opcja Redirected Header zawiera początkową część przyczynowego datagramu IPv6. Wielkość załączonej części limitowana jest wymogiem, by rozmiar zewnętrznego datagramu IPv6 enkapsulującego komunikat ICMPv6 nie przekraczał minimalnej wartość MTU dla IPv6 (obecnie 1280 bajtów)*

8.5.6.4. Opcja MTU (typ 5)

Opcja MTU, zawarta w komunikacie RA, informuje host o wielkości MTU, jaką ma przyjmować w procesie formowania wysyłanych datagramów IP — oczywiście, w warunkach gdy możliwe jest ustalanie MTU jako parametru konfiguracji. Wystąpienie opcji w komunikacie innym niż RA jest ignorowane. Format opcji przedstawiamy na rysunku 8.45.

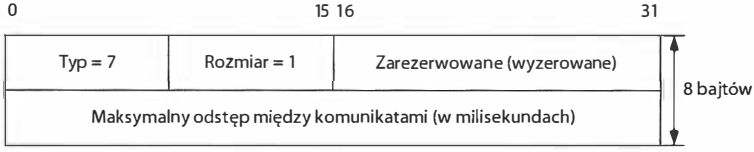


Rysunek 8.45. *Opcja MTU zawiera wartość MTU właściwą do użycia na danym łączu. Opcja ta, używana przez komunikaty RA, jest wyjątkowo użyteczna w sytuacji, gdy na łączu stosowana jest niestandardowa wartość MTU*

Opcja MTU jest istotna m.in. w przypadku mostkowania dwóch (lub więcej) sieci opartych na różnych technologiach, cechujących się odmiennymi wartościami MTU. Jeżeli używany mostek nie generuje ostrzeżeń o zbyt dużych pakietach (czyli komunikatów PTB), brak opcji MTU może uniemożliwiać bezbłędną komunikację danego hosta z hostami w partnerskiej (mostkowanej) podsieci. Zauważmy, że na MTU przeznaczono w opcji 32 bity, co daje możliwość specyfikowania nawet bardzo dużych wartości, do 4GB.

8.5.6.5. Opcja Advertisement Interval (typ 7)

Zadaniem tej opcji, załączanej do komunikatów RA, jest poinformowanie hostów o maksymalnym odstępie czasu między wysyłaniem „niezamówionych" (czyli niestanowiących odpowiedzi na RS/NS) komunikatów RA. Informacja ta wykorzystywana jest przez węzły mobilne MIPv6 w algorytmie wykrywania zmiany sieci (patrz RFC6275]). Wspomniany odstęp wyrażany jest w milisekundach — maksymalna wartość, jaką zapisać można na 16 bitach, to 65 535 milisekund. Specyfikowany odstęp czasu jest wartością graniczną — router zobowiązuje się, że komunikaty RA wysyłane będą w odstępach nie większych od podanego, lecz mogą być wysyłane częściej. Format opcji przedstawiamy na rysunku 8.46.



Rysunek 8.46. *Opcja Advertisement Interval specyfikuje maksymalny odstęp czasowy między wysyłaniem „niezamówionych" komunikatów Router Advertisement*

8.5.6.6. Opcja Home Agent Information (typ 8)

Opcja ta włączana jest do komunikatów RA wysyłanych przez routery skłonne do pełnienia funkcji agenta domowego (HA) w mobilnym IPv6 (patrz [RFC6275]), czyli komunikatów RA, w których ustawiony jest bit H (występowanie opcji jest niedozwolone

przy zerowej wartości bitu H). W przypadku, gdy seria komunikatów RA, po jednym dla każdego adresu HA, wysyłana jest w odpowiedzi na jedno żądanie RS, z ustawionym bitem R, opisywana opcja wystąpić musi w każdym z tych komunikatów i w identycznej postaci. Wystąpienie opisywanej opcji w komunikacie innym niż RA jest ignorowane. Format opcji przedstawiamy na rysunku 8.47.

Rysunek 8.47. *Opcja Home Agent Information określa stopień preferencji w zakresie używania danego routera jako agenta domowego w mobilnym IPv6 oraz okres czasu, w jakim router ten skłonny jest podjąć się pełnienia tej funkcji*

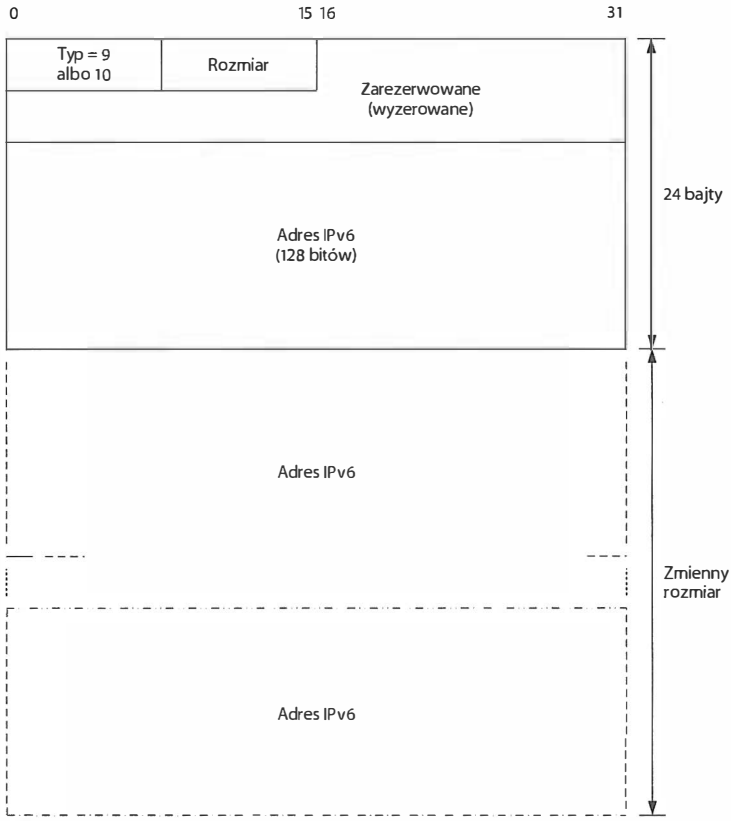
Stopień preferencji odnosi się do pełnienia funkcji agenta domowego przez router i wyrażony jest liczbą całkowitą z zakresu od 0 do 65535 — większa wartość oznacza większą przydatność routera do wspomnianej funkcji. Jeśli opisywana opcja nie występuje w komunikacie RA z ustawionym bitem H, preferencję ogłaszającego się routera uważa się za zerową, czyli najniższą.

Pole *Czas istnienia* określa (w sekundach) czas, w którym aktualna jest oferta routera pełnienia funkcji agenta domowego ze wskazaną preferencją. W polu określić można wartości z zakresu od 1 do 65535 (0 nie jest dozwolone), czyli od 1 sekundy do nieco ponad 18 godzin. Jeśli opisywana opcja nie występuje w komunikacie RA z ustawionym bitem *H*, czas ten przyjmuje się domyślnie jak równy wartości pola *Czas ważności routera* w komunikacie.

Ponieważ oba wymienione pola stanowią jedyną treść opisywanej opcji, nie musi ona występować w komunikacie, gdy preferencje i czas wartości oferty pozostawione są z wartościami domyślnymi.

8.5.6.7. Opcje Source/Target Address List (typy 9 i 10)

Opcje te, o wspólnym formacie widocznym na rysunku 8.48, związane są z komunikatami IND (patrz punkt 8.5.3 i dokument [RFC3122]). Treścią każdej z nich jest lista adresów IPv6. Dla opcji *Source Address List* (typ 9) są to adresy IPv6 wykorzystywane przez nadawcę na interfejsie, z którego wysłany został komunikat; odpowiadają one adresowi zawartemu w powiązanej opcji *Source Link-Layer Address*. Analogicznie opcja *Target Address List* (typ 10) zawiera listę adresów odpowiadających adresowi łącza danych zawartemu w opcji *Destination Link-Layer Address*. Jak wynika z rysunku, pierwszy adres rozpoczyna się na offsecie 8 bajtów w stosunku do początku opcji, zatem opcja zawierająca *N* komunikatów liczy $8 + 16 \cdot N$ *bajtów*, co wyrażone w jednostkach 8-bajtowych daje wartość $2 \cdot N + 1$ pola *Rozmiar*. Łatwo więc na podstawie tego pola wyliczyć liczbę adresów — $N = \frac{Rozmiar - 1}{2}$.



Rysunek 8.48. *Opcje Source (typ 9) i Target (typ 10) Address List, wykorzystywane w komunikatach IND do przenoszenia listy adresów IPv6; na liście tej znajdują się wyłącznie adresy używane przez węzeł dla interfejsu, przez który wysyłany jest komunikat*

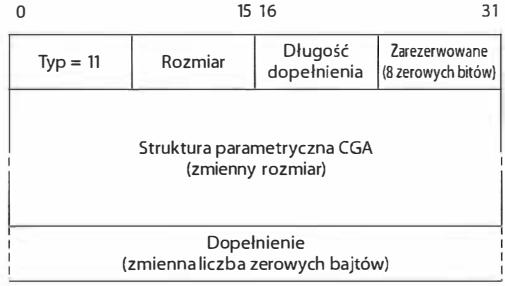
8.5.6.8. Opcja CGA (typ 11)

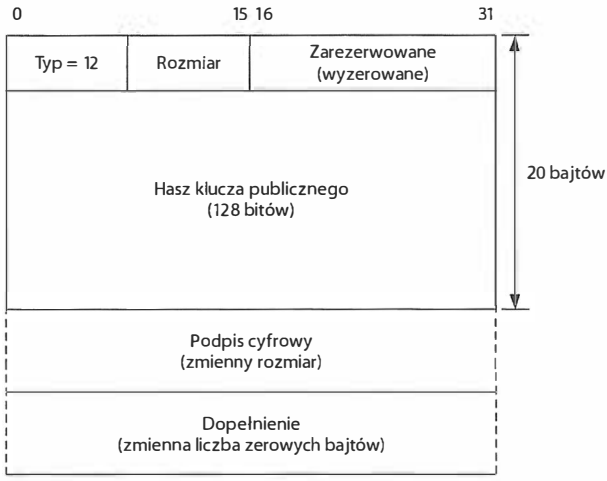
Zadaniem tej opcji jest transfer struktury parametrycznej adresu CGA (patrz podpunkt 8.5.5.1 i dokument [RFC3971]), prezentowanej na rysunku 8.38, niezbędnej do weryfikacji poprawności i legalności używania tegoż adresu. Format opcji przedstawiony jest na rysunku 8.49 — jak widać, opcja dopełniana jest bajtami zerowymi do wielokrotności 8 bajtów, wielkość dopełnienia wykazywana jest *explicite*.

8.5.6.9. Opcja RSA Signature (typ 12)

Opcja *RSA Signature* wykorzystywana jest przez SEND do przenoszenia podpisu cyfrowego RSA (patrz rozdział 18.), za pomocą którego, w połączeniu ze strukturą parametryczną CGA, weryfikator przekonuje się o posiadaniu przez nadawcę prywatnego klucza kompatybilnego z kluczem publicznym użytym do skonstruowania adresu CGA. Format opcji przedstawiamy na rysunku 8.50.

Rysunek 8.49. *Opcja CGA wykorzystywana w komunikatach SEND, przenosząca strukturę parametryczną pokazaną na rysunku 8.38*



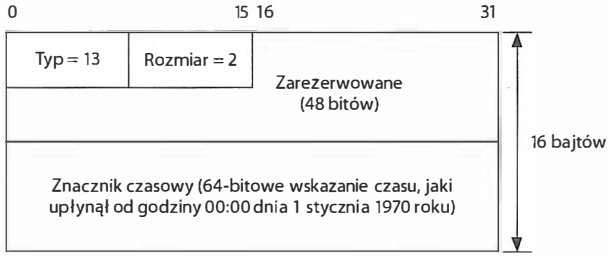


Rysunek 8.50. *Opcja RSA Signature wykorzystywana przez SEND do weryfikacji posiadania przez nadawcę właściwego klucza prywatnego, a tym samym do weryfikacji legalności używania określonego adresu CGA*

Hasz klucza publicznego to 128 najbardziej znaczących bitów wyniku funkcji haszującej SHA-1 zastosowanej do klucza publicznego, komplementarnego z kluczem prywatnym użytym do sporządzenia podpisu. Sam *Podpis cyfrowy* sporządzony jest zgodnie z algorytmem PKCS#1 v1.5 (patrz rozdział 18.) i obejmuje następujące obszary: pole *Typ* komunikatu SEND, adresy źródłowy i docelowy, pierwsze 32-bitowe słowo nagłówka ICMPv6 (czyli pola *Typ*, *Kod* i *Suma kontrolna*), nagłówek protokołu ND oraz opcje komunikatu (z wyjątkiem opcji *RSA Signature*). Opcja dopełniana jest zerowymi bajtami do wielokrotności 8 bajtów.

8.5.6.10. Opcja Timestamp (typ 13)

W ramach opcji *Timestamp* nadawca informuje o wskazaniu własnego zegara w momencie formowania komunikatu. Zawarty w opcji znacznik czasowy pomaga w ochronie przed atakami powtarzania komunikatów. Format opcji przedstawiamy na rysunku 8.51.



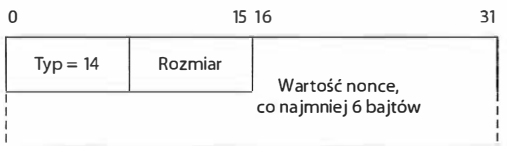
Rysunek 8.51. *Opcja Timestamp wykorzystywana przez SEND do przenoszenia aktualnego wskazania zegara nadawcy*

Wskazanie to wyrażane jest jako interwał czasu dzielącego chwilę utworzenia komunikatu od północy 1 stycznia 1970 roku; interwał ten wyrażony jest w jednostkach $\frac{1}{65536}$ sekundy. Uzależnienie legalności komunikatu od adekwatności tego znacznika stanowi zabezpieczenie przed atakami powtarzania.

Znacznik czasowy stanowiący treść opcji wyrażany jest relatywnie do północy (według UTC) rozpoczynającej rok 1970. Wartość znacznika jest liczbą całkowitą wyrażającą dystans czasowy od tego momentu, liczony w jednostkach $\frac{1}{65536}$ sekundy; najbardziej znaczące 48 bitów wyraża więc liczbę pełnych sekund.

8.5.6.11. Opcja Nonce (typ 14)

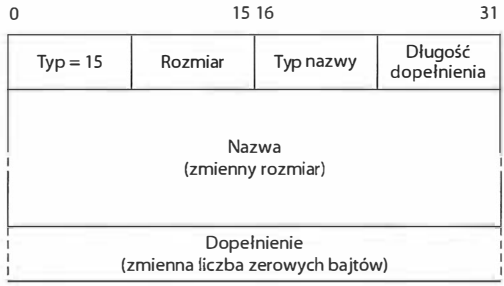
Opcja *Nonce*, zawierająca ostatnio użytą losową, unikatową wartość *nonce*, ma w założeniu — podobnie jak opcja *Timestamp* — pomagać w zapobieganiu atakom powtarzania komunikatów SEND. Zagadnienie to omawiamy obszerniej w rozdziale 18. Format opcji widoczny jest na rysunku 8.52; wartość *nonce* musi być co najmniej 6-bajtowa.



Rysunek 8.52. *Opcja Nonce wykorzystywana jest przez SEND do przenoszenia unikatowych wartości pseudolosowych w celu ochrony przed atakami powtarzania komunikatów*

8.5.6.12. Opcja Trust Anchor (typ 15)

Opcja *Trust Anchor* zawiera nazwę *root* ścieżki certyfikacji (patrz rozdział 18.), wykorzystywaną przez host do weryfikacji autentyczności komunikatów RA wysyłanych w ramach SEND. Format opcji widoczny jest na rysunku 8.53. W jednym komunikacie wystąpić może kilka takich opcji.

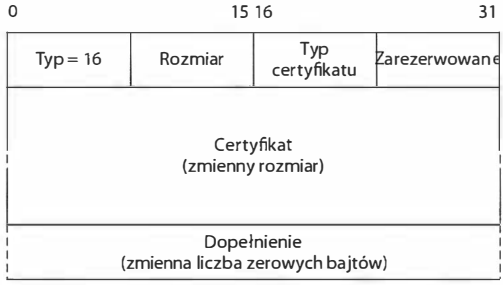


Rysunek 8.53. *Opcja Trust Anchor wykorzystywana przez SEND, niosąca nazwę root łańcucha certyfikatów wykorzystywaną jako punkt odniesienia do weryfikacji poprawności certyfikatów. Certyfikaty używane są przez SEND do walidacji komunikatów Router Advertisement*

Nazwa jest nazwą kotwicy zaufania, stanowiącą z kolei początek łańcucha certyfikatów, udostępnianego przez nadawcę w celu weryfikacji jego wiarygodności. Dotąd zdefiniowano dwa *Typy nazwy*: DER X.502 (wartość 1) i FQDN (wartość 2). Opcja dopełniana jest zerowymi bajtami do wielokrotności 8 bajtów.

8.5.6.13. Opcja Certificate (typ 16)

Opcja *Certificate* zawiera pojedynczy certyfikat spośród ścieżki certyfikacji wykorzystywanej przez SEND. Format opcji widoczny jest na rysunku 8.54.

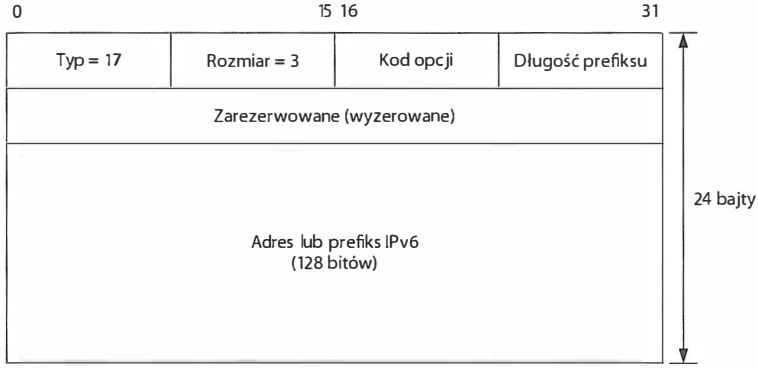


Rysunek 8.54. *Opcja Certificate wykorzystywana przez SEND, zawierająca kryptograficzny certyfikat stanowiący komponent ścieżki certyfikacji. Wykorzystywana do walidacji komunikatów Router Advertisement*

Aktualnie zdefiniowany jest tylko jeden *Typ certyfikatu* — X.509.v3 (wartość 1). Certyfikaty i sposób zarządzania nimi omawiamy w rozdziale 18.

8.5.6.14. Opcja IP Address/Prefix (typ 17)

Tytułowa opcja wykorzystywana jest w komunikatach FMIPv6 (typ 154) (patrz [RFC5568]). Jej format widoczny jest na rysunku 8.55.

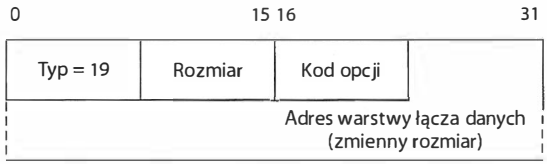


Rysunek 8.55. *Opcja Address/Prefix wykorzystywana przez FMIPv6. Zawiera prefiks lub adres IPv6 następnego routera lub adres przekierowania węzła mobilnego*

W polu *Kod opcji* wskazywany jest typ zawartości pola *Adres lub prefiks IPv6*: wartość 1 oznacza stary adres przekierowania, wartość 2 — nowy adres przekierowania, wartość 3 — adres nowego routera dostępowego (*New Access Router* — NAR), zaś wartość 4 — prefiks nowego routera dostępowego (w komunikacie *PrRtAdv*). Pole *Długość prefiksu* wskazuje liczbę początkowych bitów tworzących prefiks (w polu *Adres lub prefiks IPv6*).

8.5.6.15. Opcja Link-Layer Address (LLA) (typ 19)

Ta opcja, podobnie jak poprzednia, wykorzystywana jest w komunikatach FMIPv6 (typ ICMPv6 154, patrz [RFC5568]). Jej format widoczny jest na rysunku 8.56.

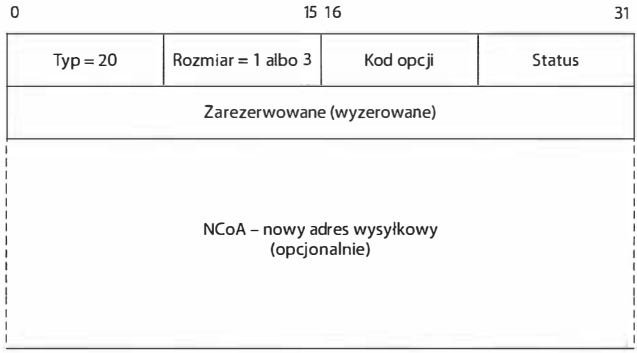


Rysunek 8.56. *Opcja Link-Layer Address wykorzystywana w komunikatach FMIPv6. W polu Kod opcji wskazywana jest encja powiązana z następującym dalej adresem*

W polu *Kod opcji* identyfikowana jest encja (encje), do której odnosi się załączony *Adres warstwy łącza danych*: 0 oznacza dowolny z pobliskich punktów dostępowych, 1 — nowy punkt dostępowy, 2 — węzeł mobilny, 3 — nowy router dostępowy, 4 — węzeł o adresie źródłowym w komunikatach *RtSolPr*/*PrRtAdv*, 5 — bieżący router, 6 — brak informacji o prefiksie dla punktu dostępowego odpowiadającego załączonemu adresowi, 7 — punkt dostępowy o podanym adresie nie realizuje opcji szybkiego przełączania

8.5.6.16. Opcja Neighbor Advertisement Acknowledgment (NAACK) (typ 20)

Opcja również wykorzystywana jest w ramach FMIPv6; jej format przedstawiamy na rysunku 8.57.



Rysunek 8.57. *Opcja Neighbor Advertisement Acknowledgment wykorzystywana przez FMIPv6. Gdy węzeł mobilny przemieszcza się między routerami dostępowymi i proponuje swój nowy adres przekierowania, nowy router przekazuje w ten sposób akceptację dla tego adresu*

Pole *Kod opcji* zawiera wartość 0, w polu *Status* znajduje się dyspozycja dotycząca „niezamówionego" komunikatu NA. Poszczególne wartości mają następujące znaczenie:

- 1 — adres NCoA jest nieprawidłowy, wykonaj konfigurację adresów;
- 2 — adres NCoA jest nieprawidłowy, wykorzystaj NCoA zawarty w opcji *IP Address*;
- 3 — adres NCoA jest nieprawidłowy, wykorzystaj w tej roli adres NAR;
- 4 — adres NCoA jest tożsamy z dotychczas wykorzystywanym, nie wykonuj aktualizacji wiązania;
- 128 — nie rozpoznano adresu warstwy łącza danych.

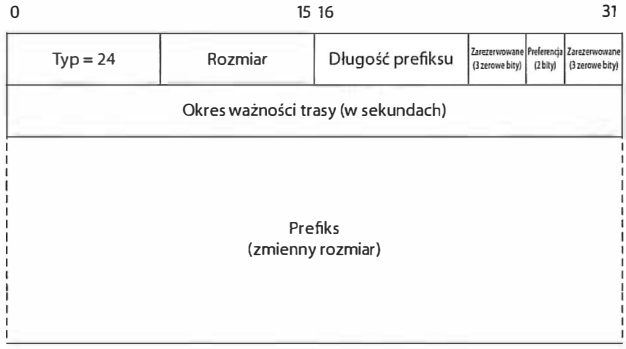
8.5.6.17. Opcja Route Information (typ 24)

Opcja *Route Information*, występująca w komunikacie RA, wskazuje, które prefiksy spoza łącza osiągalne są za pośrednictwem konkretnego routera (patrz [RFC4191]). Format opcji przedstawiamy na rysunku 8.58.

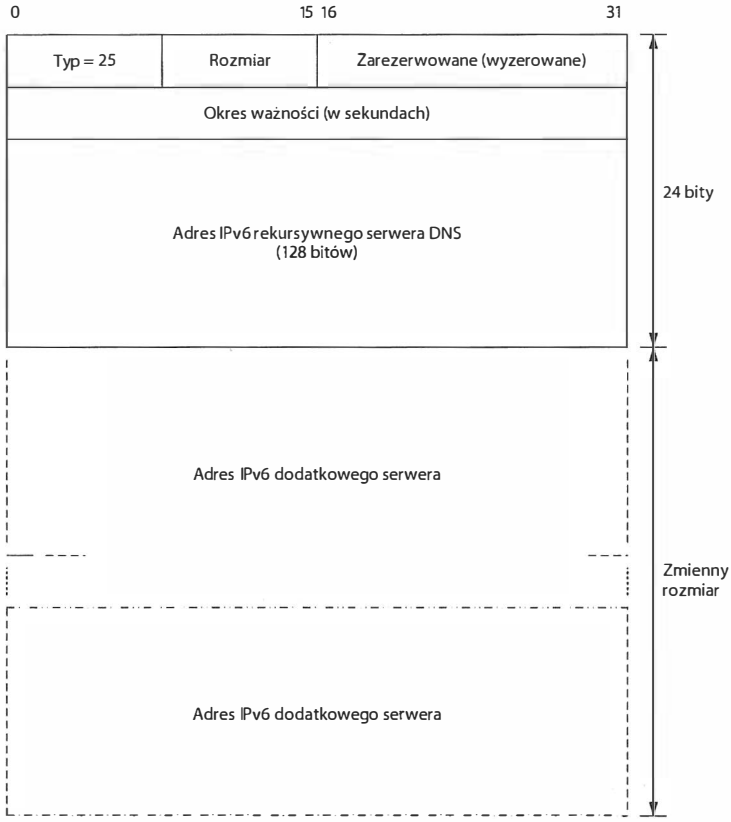
Jak zwykle, w polu *Długość prefiksu* znajduje się liczba bitów składających się na prefiks w polu *Prefiks*. Wartość 1 w polu *Preferencja* oznacza, że router wysyłający komunikat powinien być preferowany kosztem innych routerów; jeżeli pole to zawiera wartość 2, należy zignorować całą opcję. *Okres ważności trasy* to liczba sekund, w czasie których *Prefiks* może być uważany za poprawny; wartość maksymalna `0xffffffff` oznacza okres nieskończony.

8.5.6.18. Opcja Recursive DNS Server (RDNSS) (Typ 25)

Celem tej opcji, występującej w komunikatach RA, zdefiniowanej w dokumencie [RFC6106], jest ułatwienie procesu autokonfiguracji bezstanowej poprzez dostarczenie adresu (adresów) IPv6 jednego lub kilku serwerów DNS (patrz rozdziały 6. i 11.). Format opcji przedstawiamy na rysunku 8.59; w jednym komunikacie wystąpić może kilka takich opcji.



Rysunek 8.58. *Opcja Route Information wskazuje preferencję i okres ważności dla używania konkretnego routera „poza łączem". Stosowana jest w sytuacji, gdy do wyboru jest wiele domyślnych routerów, oferujących określoną jakość forwardowania w kierunku określonego miejsca przeznaczenia*

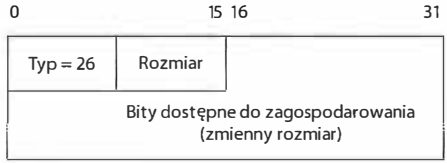


Rysunek 8.59. *Opcja Recursive DNS Server zawiera adresy serwerów DNS zdolnych do przeprowadzania rekursywnego poszukiwania (patrz rozdział 11.)*

Okres ważności wyraża liczbę sekund, w trakcie których lista adresów może być uważana za obowiązującą; wartość maksymalna `0xffffffff` oznacza okres nieskończony. Okres ważności jest wspólny dla wszystkich adresów na liście; jeśli ma być zróżnicowany dla poszczególnych adresów, konieczne jest użycie kilku opcji RDNSS w tym samym komunikacie.

8.5.6.19. Opcja Router Advertisement Flags Extension (EFO) (typ 26)

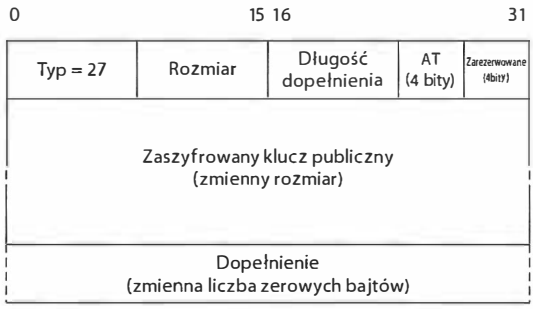
Zadaniem tej opcji, widocznej na rysunku 8.60, jest dostarczenie obszaru dla dodatkowych znaczników w komunikatach RA (patrz [RFC5175]). „Zagospodarowanie" dostępnych bitów leży w gestii IANA; obecnie do dyspozycji jest 48 bitów i pole *Rozmiar* ma wartość 1, co może się w przyszłości zmienić.



Rysunek 8.60. *Opcja Router Advertisement Flags Extension udostępniająca obszar dla dodatkowych znaczników w komunikatach RA*

8.5.6.20. Opcja Handover Key Request (typ 27)

Opcja *Handover Key Request* wykorzystywana jest w komunikatach FMIPv6/SEND jako element zabezpieczenia informacji sygnalizacyjnej (patrz [RFC5269]). Format opcji widoczny jest na rysunku 8.61.

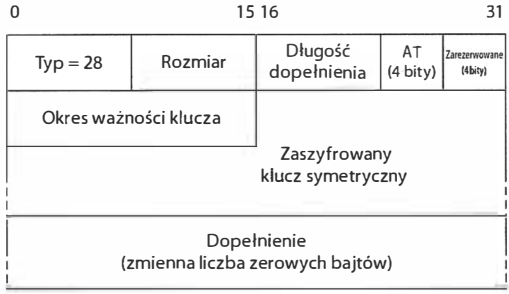


Rysunek 8.61. *Opcja Handover Key Request wykorzystywana przez komunikaty sygnalizacyjne FMIPv6 zabezpieczane przez SEND. Router wykorzystuje tę opcję do generowania klucza przełączania, dostarczanego węzłowi mobilnemu w postaci zaszyfrowanej*

Zaszyfrowany klucz publiczny utworzony jest jako wynik szyfrowania klucza publicznego nadawcy za pomocą klucza publicznego związanego z przełączaniem węzła (*handover*), w polu *AT* znajduje się natomiast identyfikator algorytmu wykorzystywanego w uwierzytelnianiu (patrz [RFC5568]). Opcja dopełniona jest zerowymi bajtami do wielokrotności 8 bajtów.

8.5.6.21. Opcja Handover Key Reply (typ 28)

Podobnie jak poprzednia opcja *Handover Key Request*, ta również wykorzystywana jest w komunikatach FMIPv6/SEND jako element zabezpieczenia informacji sygnalizacyjnej. Jej format widoczny jest na rysunku 8.62. Znaczenie pól *Długość dopełnienia* i *AT* jest takie samo jak w poprzedniej opcji. Domyślny okres ważności klucza wynosi 43200 sekund (parametr `HK-LIFETIME`). *Zaszyfrowany klucz symetryczny* jest wynikiem szyfrowania klucza symetrycznego (patrz rozdział 18.) przy użyciu klucza publicznego węzła, zakodowanym w formacie RSAES-PKCS1-v1_5 (patrz [RFC3447]). Opcja dopełniana jest bajtami zerowymi do wielokrotności 8 bajtów.



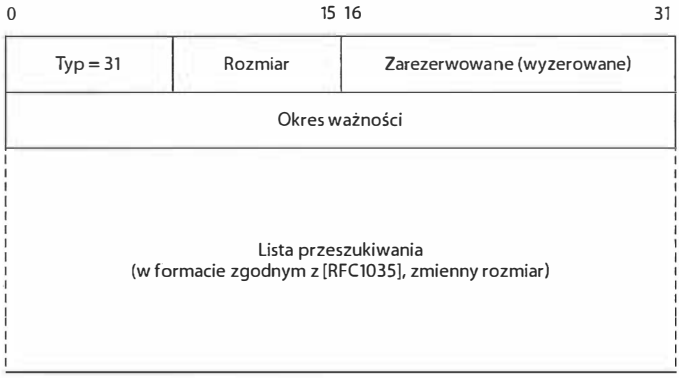
Rysunek 8.62. *Opcja Handover Key Reply wykorzystywana przez komunikaty sygnalizacyjne FMIPv6 zabezpieczane przez SEND. Zawiera symetryczny klucz przełączania zaszyfrowany publicznym kluczem węzła mobilnego; tylko węzeł będący w posiadaniu odpowiedniego klucza prywatnego może wykonać właściwe rozszyfrowanie*

8.5.6.22. Opcja DNS Search List (DNSSL) (typ 31)

Celem opcji *DNS Search List* (DNSSL — patrz [RFC6106]) jest dostarczenie listy przyrostków (sufiksów) nazw domen do żądań DNS formułowanych przez host, gdy ten realizuje wyszukiwanie na podstawie niekwalifikowanych krótkich nazw. Listy poszukiwań nazw domen są częścią informacji konfiguracyjnej DNS, która może być dostarczana hostowi podczas inicjowania jego pracy (patrz rozdział 6.). Format opcji widoczny jest na rysunku 8.63. *Lista przeszukiwania* zawiera nieskompresowane rozszerzenia nazw domen, wykorzystywane jako domyślne przy konstruowaniu pełnych kwalifikowanych nazw domen (FQDN — patrz rozdział 11.). *Okres ważności* jest wspólny dla wszystkich nazw występujących na liście; jego zróżnicowanie dla poszczególnych nazw możliwe jest jedynie w drodze wielokrotnego użycia opcji w tym samym komunikacie.

8.5.6.23. Opcje eksperymentalne (typy 253 i 254)

Wartości 253 i 254 pola *Typ* opcji ND zarezerwowane są dla celów eksperymentalnych, zgodnie z wytycznymi dokumentu [RFC3692].



Rysunek 8.63. *Opcja DNS Search List dostarcza listę domyślnych rozszerzeń nazw domen DNS, wykorzystywaną w procesie konfigurowania hosta. Format kodowania pozycji na liście jest identyczny z kodowaniem nazw DNS (patrz rozdział 11.)*

8.6. Translacja komunikatów między ICMPv4 a ICMPv6

W rozdziale 7. omawialiśmy konwersję (translację) pakietów IPv4 na równoważne pakiety IPv6 — i odwrotnie, ograniczając się do konwersji nagłówków, zgodnie z dokumentami [RFC6144] i [RFC6145]. W dokumencie [RFC6145] opisana jest także translacja między pakietami ICMPv4 i ICMPv6. Obejmuje ona nie tylko translację nagłówków IP i ICMP, lecz dodatkowo w komunikatach ICMP o błędach konwertowany jest także nagłówek IP przyczynowego datagramu. To jeszcze nie koniec: wobec zasadniczych różnic między protokołami IPv4 i IPv6 wspomniana konwersja uwzględniać musi konsekwencje tych różnic w obszarach, takich jak fragmentacja, wielkości MTU czy obliczanie sum kontrolnych — jak pamiętamy, w ICMPv4 suma kontrolna ogranicza się do pól samego protokołu ICMP, podczas gdy w ICMPv6 obejmuje także pseudonagłówek złożony z niektórych pól zewnętrznego datagramu IPv6.

8.6.1. Translacja z ICMPv4 na ICMPv6

Spośród informacyjnych komunikatów ICMPv4 translacji podlegają jedynie *Echo Request* i *Echo Reply*: wartości typów 8 i 0 zastępowane są wartościami (odpowiednio) 128 i 129. Po wykonaniu translacji obliczana jest na nowo suma kontrolna ICMPv6. Spośród komunikatów ICMPv4 o błędach translacji podlegają komunikaty *Destination Unreachable* (typ 3), *Time Exceeded* (typ 11) i *Parameter Problem* (typ 12) — reguły zastępowania pól *Typ* i *Kod* zestawione są w tabeli 8.6. Wszystkie inne typy komunikatów nie są poddawane translacji — niosące je pakiety są odrzucane.

Tabela 8.6. *Mapowanie pól Typ i Kod przy translacji komunikatów ICMPv4 o błędach na komunikaty ICMPv6*

Typ/Kod ICMPv4	Nazwa oficjalna ICMPv4	Typ/Kod ICMPv6	Nazwa oficjalna ICMPv6, uwagi
3/0	*Destination Unreachable — Network*	1/0	*Destination Unreachable — No Router*
3/1	*Destination Unreachable — Host*	1/0	*Destination Unreachable — No Router*
3/2	*Destination Unreachable — Protocol*	4/1	*Parameter Problem — Unrecognized Next Header* (*Wskaźnik* ustawiony na pole *Następny nagłówek*)
3/3	*Destination Unreachable — Port*	1/4	*Destination Unreachable — Port*
3/4	*Destination Unreachable — Fragmentation Required (PTB)*	2/0	*PTB* (aktualizacja pola *MTU* w związku z większym rozmiarem nagłówka IPv6)
3/5	*Destination Unreachable — Source Route Failed*	1/0	*Destination Unreachable — No Route* (wystąpienie mało prawdopodobne)
3/6	*Destination Unreachable — Destination Network Unknown*	1/0	*Destination Unreachable — No router*
3/7	*Destination Unreachable — Destination Host Unknown*		
3/8	*Destination Unreachable — Source Host Isolated*	1/0	*Destination Unreachable — No Router*
3/9	*Destination Unreachable — DestinationNetwork*	1/1	*Destination Unreachable — Communication with Destination Administratively Prohibited*
3/10	*Host Administratively Prohibited*		
3/11	*Destination Network Unreachable for Type of Service*	1/0	*Destination Unreachable — No Router*
3/12	*Destination Host Unreachable for Type of Service*		
3/13	*Destination Unreachable — Administratively Prohibited*	1/1	*Destination Unreachable — Communication with Destination Administratively Prohibited*
3/14	*Destination Unreachable — Host Precedence Violation*	Pakiet zostaje odrzucony	
3/15	*Destination Unreachable — Precedence Cutoff in Effect*	1/1	*Destination Unreachable — Communication with Destination Administratively Prohibited*
11/0	*Time Exceeded — TTL*	3/0	*Time Exceeded*
11/1	*Time Exceeded — Fragment Reassembly*	3/1	

Tabela 8.6. *Mapowanie pól Typ i Kod przy translacji komunikatów ICMPv4 o błędach na komunikaty ICMPv6 — ciąg dalszy*

Typ/Kod ICMPv4	Nazwa oficjalna ICMPv4	Typ/Kod ICMPv6	Nazwa oficjalna ICMPv6, uwagi
12/0	*Parameter Problem — Pointer Contains Byte Offset of Error*	4/0	*Parameter Problem — Erroneous Header Field Encountered* (zaktualizowany *Wskaźnik*, według tabeli 8.7)
12/1	*Parameter Problem — Missing Option*	Pakiet zostaje odrzucony	
12/2	*Parameter Problem — Bad Length*	4/0	*Parameter Problem — Erroneous Header Field Encountered* (zaktualizowany *Wskaźnik*, według tabeli 8.7)

Jak widać, w niektórych przypadkach (3/2, 12/0 i 12/2) zaktualizowane zostaje pole *Wskaźnik* w taki sposób, by w nowej wersji wskazywać pole nagłówka IPv6 stanowiące odpowiednik przyczynowego pola w IPv4. Szczegóły tej translacji dla dwóch ostatnich przypadków zestawione są w tabeli 8.7.

Tabela 8.7. *Szczegóły konwersji pola* Wskaźnik *przy translacji komunikatu ICMPv4 na ICMPv6*

Wartość pola Wskaźnik w ICMPv4	Przyczynowe pole nagłówka IPv4	Wartość pola Wskaźnik w ICMPv6	Odpowiadające pole nagłówka IPv6
0	*Wersja/IHL*	0	*Wersja/DSF/ECN (Klasa ruchu)*
1	*DSF/ECN (typ usługi)*	1	*DSF/ECN (Klasa ruchu)/Etykieta przepływu*
2, 3	*Całkowity rozmiar*	4	*Rozmiar ładunku użytecznego*
4, 5	*Identyfikacja*	Brak odpowiednika	
6	*Znaczniki/Offset fragmentu*		
7	*Offset fragmentu*		
8	*Czas życia*	7	*Limit przeskoków*
9	*Protokół*	6	*Następny nagłówek*
10,11	*Suma kontrolna nagłówka*	Brak odpowiednika	
12 – 15	*Źródłowy adres IP*	8	*Źródłowy adres IP*
16 – 19	*Docelowy adres IP*	24	*Docelowy adres IP*

Oprócz translacji nagłówka komunikatu konwersji podlega również zawarty w komunikacie fragment przyczynowego datagramu IP, zgodnie z opisanymi w rozdziale 7. regułami konwersji pakietów IP. Konwersja realizowana jest jedynie na pierwszym poziomie — poziomie nagłówka IP przyczynowego pakietu; gdy stwierdzona zostanie

obecność innych wewnętrznych nagłówków IP (np. w ramach tunelowania), cały komunikat ICMP jest odrzucany. Ponieważ konwersja ta zwiększa nagłówek wewnętrznego pakietu, powiększa się w rezultacie rozmiar zewnętrznego datagramu IP, enkapsulującego komunikat ICMP. Efekt ten zostaje — oczywiście — uwzględniony w wynikowej wartości pola *Rozmiar całkowity*. Ogólnie rzecz biorąc, w przypadku odrzucenia pakietu IPv4 w procesie translacji translator wysyła do nadawcy komunikat ICMPv4 typu 3 o kodzie 13 (*Destination Unreachable — Administratively Prohibited*), wyjątkiem jest jednak sytuacja, gdy odrzucany pakiet sam jest pakietem ICMPv4. Jak pamiętamy z opisu translacji nagłówków IP w rozdziale 7., pakiety IPv4 z wyzerowanym bitem DF generować mogą wynikowe pakiety IPv6 w kilku fragmentach, o rozmiarach nieprzekraczających minimalnej wartości MTU. Takie fragmentowanie zawczasu podyktowane jest faktem, że routery IPv4 mogą fragmentować datagramy IPv4 „po drodze" (w tym również datagramy niosące komunikaty ICMP), podczas gdy w protokole IPv6 fragmentacja datagramu dopuszczalna jest jedynie w hoście generującym ów datagram. Komunikaty PTB IMCPv4 mogą być konwertowane na komunikaty PTB ICMPv6 określające wartość MTU mniejszą niż minimalne dla łącza 1280 bajtów. Prawidłowo funkcjonujący stos IPv6 przetworzy takie komunikaty, przesyłając kolejne fragmenty wynikowego datagramu IPv6 do tego samego przeznaczenia, które specyfikowane jest w oryginalnym datagramie IPv4.

8.6.2. Translacja z ICMPv6 na ICMPv4

Spośród informacyjnych komunikatów ICMPv4 translacji podlegają jedynie *Echo Request* i *Echo Reply*: wartości typów 128 i 129 zastępowane są wartościami (odpowiednio) 8 i 0. Po wykonaniu translacji obliczana jest na nowo suma kontrolna ICMPv4 — w przeciwieństwie do IPv6 obejmująca wyłącznie obszar komunikatu ICMP. Komunikaty ICMPv6 o błędach konwertowane są zgodnie z tabelą 8.8. Pole *Wskaźnik* zachowuje swe znaczenie, musi jednak zmienić swą wartość ze względu na znaczące różnice w strukturze nagłówków IPv6 i IPv4. Szczegóły konwersji tej wartości zestawione są w tabeli 8.9.

Tabela 8.8. *Mapowanie pól Typ i Kod przy translacji komunikatów ICMPv6 o błędach na komunikaty ICMPv4*

Typ/Kod ICMPv6	Nazwa oficjalna ICMPv6	Typ/Kod ICMPv4	Nazwa oficjalna ICMPv4, uwagi
1/0	*Destination Unreachable — No Route*	3/1	*Destination Unreachable — Host*
1/1	*Destination Unreachable — Communication with Destination Administratively Prohibited*	3/10	*Destination Unreachable — Destination Host Administratively Prohibited*
1/2	*Destination Unreachable — Beyond Scope of Source Address*	3/1	*Destination Unreachable — Host*
1/3	*Destination Unreachable — Address*	3/1	*Destination Unreachable — Host*
1/4	*Destination Unreachable — Port*	3/3	*Destination Unreachable — Port*
2/0	PTB	3/4	*Destination Unreachable — Fragmentation Required* (PTB)

Tabela 8.8. *Mapowanie pól Typ i Kod przy translacji komunikatów ICMPv6 o błędach na komunikaty ICMPv4 — ciąg dalszy*

Typ/Kod ICMPv6	Nazwa oficjalna ICMPv6	Typ/Kod ICMPv4	Nazwa oficjalna ICMPv4, uwagi
3/0	*Time Exceeded — Hop Limit*	11/0	*Time Exceeded — TTL*
3/1	*Time Exceeded — Fragment Reassembly*	11/1	*Time Exceeded — Fragment Reassembly*
4/0	*Parameter Problem — Erroneous Header Field Encountered*	12/0	*Parameter Problem* (zaktualizowany *Wskaźnik*, według tabeli 8.9)
4/1	*Parameter Problem — Unrecognized Next Header*	3/2	*Destination Unreachable—Protocol* (*Wskaźnik* ustawiony na pole *Protokół*)
4/2	*Parameter Problem — Unrecognized IPv6 Option Encountered*	Pakiet zostaje odrzucony	

Tabela 8.9. *Szczegóły konwersji pola Wskaźnik przy translacji komunikatu ICMPv6 na ICMPv4*

Wartość pola Wskaźnik w ICMPv6	Przyczynowe pole nagłówka IPv6	Wartość pola Wskaźnik w ICMPv4	Odpowiadające pole nagłówka IPv4
0	*Wersja/DSF/ECN* (*Klasa ruchu*)	0	*Wersja/IHL/DSF/ECN* (*Typ usługi*)
1	*DSF/ECN* (*Klasa ruchu*)/ *Etykieta przepływu*	1	*DSF/ECN* (*Typ usługi*)
2, 3	*Etykieta przepływu*	Brak odpowiednika	
4, 5	*Rozmiar ładunku użytecznego*	Nie dotyczy	*Rozmiar całkowity*
6	*Następny nagłówek*	9	*Protokół*
7	*Limit przeskoków*	8	*Czas życia*
8 – 23	*Źródłowy adres IP*	12	*Źródłowy adres IP*
24 – 39	*Docelowy adres IP*	16	*Docelowy adres IP*

Jak pamiętamy, suma kontrolna ICMPv4 nie obejmuje pseudonagłówka, więc po translacji nagłówka z IPv6 na IPv4 musi być ponownie obliczona (chyba że wykonywana jest tzw. neutralna translacja adresów, niewpływająca na sumę kontrolną). Należy ponadto uwzględnić sytuację, gdy źródłowy datagram IPv6 zawiera adresy niedające się konwertować na IPv4 — przeprowadzana jest wówczas translacja wykorzystująca informację o stanie (NAT64 — pisaliśmy o niej w podpunkcie 7.6.2.3).

Gdy pojawia się problem dużego pakietu IPv6 — zbyt dużego w stosunku do MTU protokołu IPv4 — sprawa jest o tyle skomplikowana, że w datagramie IPv6 nie istnieje znacznik zabraniający fragmentowania, bo jest ono z definicji zabronione w routerach pośredniczących. Jeśli więc do translatora nadchodzi pakiet IPv6 przekraczający rozmiarem wartość MTU interfejsu IPv4 skierowanego w stronę następnego przeskoku, pakiet ten zostaje odrzucony, a do nadawcy wysyłany jest komunikat ICMPv6 PTB.

8.7. Ataki wykorzystujące ICMP

Ataki wykorzystujące protokół ICMP podzielić można z grubsza na trzy kategorie, scharakteryzowane obrazowo jako powodzie (*floods*), bomby (*bombs*) i *udostępnianie informacji*. Ogólnie rzecz biorąc, „powodzie" to stwarzanie warunków powodujących generowanie lawinowego ruchu, prowadzącego w krótkiej perspektywie do ataku DoS na jeden lub więcej komputerów. „Bomby" zwane także „pociskami nuklearnymi" (*nukes*) to specjalnie skonstruowane komunikaty, wywołujące przetwarzanie IP lub ICMP prowadzące do załamania lub zawieszenia systemu. *Udostępnianie informacji* nie powoduje zazwyczaj zakłócenia samo z siebie, lecz może pełnić rolę pomocniczą dla innych rodzajów ataku — także w ich ukrywaniu. Atakom na protokół TCP z wykorzystaniem ICMP poświęcono osobny dokument [RFC5927].

Jeden z pierwszych ataków, zwany *atakiem smurfa*, polegał na wysyłaniu komunikatów ICMPv4 na adres rozgłoszeniowy, z zamiarem wywołania odpowiedzi z dużej liczby komputerów. W rezultacie komputer, którego adres figuruje w komunikacie jako źródłowy, zostaje zarzucony lawiną odpowiedzi, których przetwarzanie nie pozostawia już wolnej mocy procesora do wykonywania czegokolwiek innego. Atakom tego typu można jednak łatwo przeciwdziałać, zatrzymując na firewallu komunikaty rozgłaszania ukierunkowanego (*directed broadcast*).

Kolejna możliwość ataku kryje się we fragmentacji pakietów: za pomocą komunikatów *Echo* można skonstruować ciąg pakietów IP formalnie stanowiących fragmenty pojedynczego datagramu IPv4; jeśli fragmenty te zostaną tak spreparowane, że po reasemblacji rozmiar wynikowego datagramu przekraczać będzie 64 kB. Co prawda, większość implementacji odrzuci wszystkie fragmenty, ale np. w Windows 3.11, Windows 95 i Windows NT oraz w sieciach Novell w wersji 3.x sytuacja taka spowoduje awarię aplikacji i przeważnie zawieszenie systemu. Znowu mamy więc skuteczny przepis na atak DoS, zwany popularnie „śmiertelnym pingiem" (*ping of death*), możliwy do realizacji za pomocą prostego polecenia

```
ping -l 65510 <adres_ip>
```

Podobny atak, zwany popularnie „łezką" (*teardrop*), polega na manipulowaniu zawartością pola *Offset fragmentu* w nagłówku IPv4 poszczególnych fragmentów; niepoprawna wartość w tym polu może spowodować dezorganizację implementacji protokołu IP.

Jednym z założeń protokołu ICMP jest różnica między adresem źródłowym a adresem docelowym w wysyłanych komunikatach. Naruszenie tej zasady — czyli obsadzenie adresu komputera ofiary w obu tych rolach — w wielu implementacjach może skutkować naprawdę kuriozalną reakcją na odebrany pakiet. Ten rodzaj ataku nazywają hakerzy (po prostu) *lądowaniem* pakietu (*land*).

Przy wykorzystaniu funkcji przekierowania (*redirect*) protokołu ICMP można wymusić zmianę trasy pakietu, wskazując jego adresatowi niewłaściwy router w roli następnego przeskoku. Mimo iż przetwarzanie komunikatów ICMP *Redirect* wiąże się z szeregiem kontroli, m.in. z upewnieniem się, że komunikat wysłany został przez aktualnie domyślny router, nie istnieje możliwość zweryfikowania autentyczności tego komunikatu. Stanowi to doskonałą pożywkę dla ataków „z człowiekiem pośrodku" (*man-in-the-middle*

— patrz rozdział 18.), w ramach których intruz przechwytuje i zapamiętuje przesyłane komunikaty, po czym po pewnym czasie wysyła je powtórnie do wybranego przeznaczenia, być może po określonym zmodyfikowaniu. Osiąga się w ten sposób efekty podobne do *zatruwania ARP* (*ARP poisoning*). Za pomocą spreparowanych komunikatów ICMP można przekonać host ofiarę, że to *on sam* jest dla siebie domyślnym routerem (bramą) w kierunku określonego przeznaczenia; wywołane w ten sposób zapętlenie prowadzi najczęściej do zablokowania komputera.

Za pomocą spreparowanych komunikatów *Router Solicitation* i *Router advertisement* można uzyskać efekt podobny do przekierowania — system ofiara zmienia swe domyślne trasy w taki sposób, że wysyłane pakiety trafiają do wskazanego hosta intruza zamiast do oryginalnego przeznaczenia. Nawet jeśli intruz nie zrobi użytku z przechwyconej w ten sposób informacji, to i tak bierne obserwowanie przychodzącego ruchu daje mu możliwość rozpoznawania topologii sieci nadawczej. Tego rodzaju „oszukańczym" komunikatom, przypadkowym lub spreparowanym celowo, poświęcono dokument [RFC6104].

Komunikaty ICMP mogą stanowić znakomite narzędzie do komunikowania się rozproszonych komponentów aplikacji, np. wirusów rozmieszczonych na tysiącach zainfekowanych komputerów i czekających tylko na sygnał do skoordynowanego ataku. Ten rodzaj ataku urzeczywistniony został po raz pierwszy za pomocą narzędzia o nazwie *Tribal Flood Network* (dosł. „plemienna sieć przepływu") stworzonego przez niemieckiego hakera o pseudonimie Mixter; w obiegu znajdują się podobne narzędzia o nazwie *Trinoo* i *Stacheldraht* (dosł. „drut kolczasty").

Prymitywną w swej naturze, acz często skuteczną, odmianą ataku DoS jest wykorzystywanie komunikatów grupy ICMP *Destination Unreachable*; zwane żartobliwie *smack* („cmok") lub *bloop*. Wiele implementacji TCP/IP po odebraniu takiego komunikatu z określonego adresu IP zamyka istniejące połączenia z tym adresem na poziomie warstwy transportowej, nawet jeśli połączenia te funkcjonują w pełni poprawnie; wystarczy tylko odpowiednio spreparować adres źródłowy w komunikacie ICMP.

Wiele mechanizmów bezpieczeństwa opiera się na technikach kryptograficznych, wykorzystujących liczby pseudolosowe (piszemy o tym obszernie w rozdziale 18.). Te zaś generowane są na bazie zalążka (*seed*) pozostającego w prostej relacji do aktualnego wskazania zegara, które poznać możemy za pomocą komunikatów grupy ICMP *Timestamp*, normalnie niewykorzystywanych samodzielnie. Znając algorytm generowania i wartość wspomnianego zalążka, można z dużym prawdopodobieństwem przewidzieć np. wartość najbliższej użytej *nonce* czy też generalnie ciąg takich wartości (stąd fraza „pseudo") albo sekretne klucze wykorzystywane do ukrywania i uwierzytelniania informacji.

Obiektem szkodliwych działań może się również stać wartość MTU, zawarta w komunikacie ICMP PTB. Protokoły transportowe (m.in. TCP) wykorzystują tę wartość jako podstawę do określenia rozmiaru generowanych pakietów; zaniżając nienaturalnie tę wartość, zmuszamy TCP do generowania dużej liczby mikroskopijnych pakietów, co niezawodnie objawi się wyraźną degradacją wydajności.

Wielu z wymienionych ataków można skutecznie przeciwdziałać, ulepszając implementacje protokołu ICMP funkcjonujące w ramach popularnych systemów operacyjnych. Jednak atakom, takim jak maskarada czy preparowanie komunikatów, skuteczny odpór

dać można wyłącznie za pomocą technik kryptograficznych. Protokoły korzystające z takich technik (np. SEND) oferują zatem większe bezpieczeństwo, są jednak bardziej skomplikowane we wdrażaniu i pod względem poszukiwania przyczyn ich (ewentualnego) problematycznego zachowania.

8.8. Podsumowanie

W tym rozdziale opisywaliśmy protokół ICMP, w wersjach ICMPv4 i ICMPv6, nieodłączną część każdej implementacji protokołu IP. Komunikaty ICMP przenoszone są w datagramach IP, a ich integralność weryfikowana jest za pomocą sumy kontrolnej obejmującej całość komunikatu, a w ICMPv6 dodatkowo także pseudonagłówek; jest to istotna różnica w porównaniu chociażby z datagramami IP, gdzie suma kontrolna uwzględnia jedynie nagłówek. Komunikaty ICMP podzielić można z grubsza na dwie grupy: komunikaty informacyjne i komunikaty o błędach — te ostatnie nie są jednak generowane przy problemach z komunikatami ICMP, co pozwala uniknąć zagrożenia lawiną takich komunikatów. Komunikaty o błędach zapewniają sygnalizowanie różnych sytuacji uniemożliwiających dostarczanie datagramów IP — nieosiągalność węzła docelowego, zbyt duży rozmiar pakietu, wyczerpanie limitu przeskoków, wyczerpanie czasu przeznaczonego na reasemblację pakietu itd. Mechanizm przekierowania (*redirect*) umożliwia także korygowanie błędnie wybieranych przez host routerów. Komunikaty informacyjne udostępniają na żądanie określone elementy informacji konfiguracyjnej; do kategorii informacyjnych należą też komunikaty *Echo*, stanowiące podstawę działania popularnych aplikacji `ping` i `traceroute`.

Omówiliśmy szczegółowo komunikaty ICMP grupy *Destination Unreachable*, *Redirect* i *Echo*. W miarę potrzeb do komunikatu ICMP załączany jest fragment (być może — całość) datagramu stanowiącego przyczynę odnośnego błędu. Rozmiar załączonej porcji limitowany jest maksymalną wielkością datagramu IP enkapsulującego komunikat ICMP — wielkość ta nie może przekroczyć wartości MTU na łączu, bo komunikaty ICMP przesyłane są bez fragmentowania. Informacja zawarta w kopii przyczynowego datagramu pozwala zidentyfikować kontekst wystąpienia błędnej sytuacji — protokół warstwy wyższej, znaczniki MPLS, adres następnego przeskoku itp.

Protokół ICMPv6 spełnia w ramach IPv6 rolę o wiele istotniejszą niż ta, jaką ICMPv4 pełnił na potrzeby IPv4. Przede wszystkim ICMPv6 spełnia krytyczną rolę w konfigurowaniu systemów implementujących IPv6. ICMPv6 przejmuje większość komunikatów z ICMPv4 — m.in. *Destination Unreachable*, *Time Exceeded*, *Fragmentation Required* i *Echo* — a także funkcjonalność protokołu ARP, w postaci mechanizmu odnajdywania sąsiadów (*Neighbor Discovery* — ND) umożliwiającego wykrywanie hostów znajdujących się na tym samym łączu, wykrywanie domyślnych routerów i dynamiczne konfigurowanie węzłów mobilnych MIPv6. ICMPv6 sprawuje także kontrolę nad przynależnością hostów do określonych grup multicast — na gruncie IPv4 funkcję tę spełniał protokół IGMP (o multicastingu piszemy dokładniej w rozdziale 9.). Komunikatom ND towarzyszyć mogą rozmaite opcje, w wielu sytuacjach obowiązkowe.

Ponieważ komunikaty ICMPv6 spełniają istotną rolę w konfigurowaniu hostów, ważnym zagadnieniem staje się ich ochrona przed rozmaitymi atakami. Z myślą o zwiększeniu bezpieczeństwa komunikatu zaprojektowano jego bezpieczną wersję o nazwie SEND,

obejmującą weryfikację poprawności legalności używanych adresów IPv6, generowanych na drodze kryptograficznej. Adresy generowane w ten sposób, określane skrótowo adresami CGA (*Cryptographically Generated Addresses*), stanowią same z siebie interesujący mechanizm, wykraczający swym zastosowaniem poza protokół SEND.

8.9. Bibliografia

[A03] T. Aura „Cryptographically Generated Addresses (CGA)", *Proc. 6th Information Security Conference (ISC)*, październik 2003.

[ICMP6TYPES] *http://www.iana.org/assignments/icmpv6-parameters*

[ICMPTYPES] *http://www.iana.org/assignments/icmp-parameters*

[PING] *http://ftp.arl.army.mil/~mike/ping.html*

[RFC0792] J. Postel, *Internet Control Message Protocol*, Internet RFC 0792/STD 0005, wrzesień 1981.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1191] J. C. Mogul, S.E. Deering, *Path MTU Discovery*, Internet RFC 1191, listopad 1990.

[RFC1256] S. Deering (red.), *ICMP Router Discovery Messages*, Internet RFC 1256, wrzesień 1991.

[RFC1350] K. Sollins, *The TFTP Protocol (Revision 2)*, Internet RFC 1350/STD 0033, lipiec 1992.

[RFC1812] F. Baker (red.), *Requirements for IP Version 4 Routers*, Internet RFC 1812, czerwiec 1995.

[RFC2004] C. Perkins, *Minimal Encapsulation within IP*, Internet RFC 2004, październik 1996.

[RFC2349] G. Malkin, A. Harkin, *TFTP Timeout Interval and Transfer Size Options*, Internet RFC 2349, maj 1998.

[RFC2460] S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet RFC 2460, grudzień 1998.

[RFC2491] G. Armitage, P. Schulter, M. Jork, G. Harter, *IPv6 over Non-Broadcast Multiple Access (NBMA) Networks*, Internet RFC 2491, styczeń 1999.

[RFC2710] S. Deering, W. Fenner, B. Haberman, *Multicast Listener Discovery (MLD) for IPv6*, Internet RFC 2710, październik 1999.

[RFC3024], G. Montenegro (red.), *Reverse Tunneling for Mobile IP, Revised*, Internet RFC 3024, styczeń 2001.

[RFC3122] A. Conta, *Extensions to IPv6 Neighbor Discovery for Inverse Discovery Specification*, Internet RFC 3122, czerwiec 2001.

[RFC3447] J. Jonsson, B. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, Internet RFC 3447 (informational), luty 2003.

[RFC3519] H. Levkowetz, S. Vaarala, *Mobile IP Traversal of Network Address Translation (NAT) Devices*, Internet RFC 3519, kwiecień 2003.

[RFC3543] S. Glass, M. Chandra, *Registration Revocation in Mobile IPv4*, Internet RFC 3543, sierpień 2003.

[RFC3590] B. Haberman, *Source Address Selection for the Multicast Listener Discovery (MLD) Protocol*, Internet RFC 3590, wrzesień 2003.

[RFC3692] T. Narten, *Assigning Experimental and Testing Numbers Considered Useful*, Internet RFC 3692/BCP 0082, styczeń 2004.

[RFC3704] F. Baker, P. Savola, *Ingress Filtering for Multihomed Networks*, Internet RFC 3704/BCP 0084, marzec 2004.

[RFC3756] P. Nikander (red.), J. Kempf, E. Nordmark, *IPv6 Neighbor Discovery (ND) Trust Models and Threats*, Internet RFC 3756 (informational), maj 2004.

[RFC3810] R. Vida, L. Costa (red.), *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*, Internet RFC 3810, czerwiec 2004.

[RFC3971] J. Arkko (red.), J. Kempf, B. Zill, P. Nikander, *SEcure Neighbor Discovery (SEND)*, Internet RFC 3971, marzec 2005.

[RFC3972] T. Aura, *Cryptographically Generated Addresses (CGA)*, Internet RFC 4972, marzec 2005.

[RFC4191] R. Draves, D. Thaler, *Default Router Preferences and More-Specific Routes*, Internet RFC 4191, listopad 2005.

[RFC4286] B. Haberman, J. Martin, *Multicast Router Discovery*, Internet RFC 4286, grudzień 2005.

[RFC4389] D. Thaler, M. Talwar, C. Patel, *Neighbor Discovery Proxies (ND Proxy)*, Internet RFC 4389 (experimental), kwiecień 2006.

[RFC4443] A. Conta, S. Deering, M. Gupta (red.), *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, Internet RFC 4443, marzec 2006.

[RFC4581] M. Bagnulo, J. Arkko, *Cryptographically Generated Addresses (CGA) Extension Field Format*, Internet RFC 4581, październik 2006.

[RFC4604] H. Holbrook, B. Cain, B. Haberman, *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*, Internet RFC 4604, sierpień 2006.

[RFC4607] H. Holbrook, B. Cain, *Source-Specific Multicast for IP*, Internet RFC 4607, sierpień 2006.

[RFC4727] B. Fenner, *Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers*, Internet RFC 4727, listopad 2006.

[RFC4857] E. Fogelstroem, A. Jonsson, C. Perkins, *Mobile IPv4 Regional Registration*, Internet RFC 4857 (experimental), czerwiec 2007.

[RFC4861] T. Narten, E. Nordmark, W. Simpson, H. Soliman, *Neighbor Discovery for IP Version 6 (IPv6)*, Internet RFC 4861, wrzesień 2007.

[RFC4884] R. Bonica, D. Gan, D. Tappan, C. Pignataro, *Extended ICMP to Support Multi-Part Messages*, Internet RFC 4884, kwiecień 2007.

[RFC4890] E. Davies, J. Mohacsi, *Recommendations for Filtering ICMPv6 Messages in Firewalls*, Internet RFC 4890 (informational), maj 2007.

[RFC4950] R. Bonica, D. Gan, D. Tappan, C. Pignataro, *ICMP Extensions for Multiprotocol Label Switching*, Internet RFC 4950, sierpień 2007.

[RFC4982] M. Bagnulo, J. Arkko, *Support for Multiple Hash Algorithms in Cryptographically Generated Addresses (CGAs)*, Internet RFC 4982, lipiec 2007.

[RFC5175] B. Haberman (red.), R. Hinden, *IPv6 Router Advertisement Flags Option*, Internet RFC 5175, marzec 2008.

[RFC5269] J. Kempf, R. Koodli, *Distributing a Symmetric Fast Mobile IPv6 (FMIPv6) Handover Key Using SEcure Neighbor Discovery (SEND)*, Internet RFC 5269, czerwiec 2008.

[RFC5461] F. Gont, *TCP's Reaction to Soft Errors*, Internet RFC 5461 (informational), luty 2009.

[RFC5508] P. Srisuresh, B. Ford, S. Sivakumar, S. Guha, *NAT Behavioral Requirements for ICMP*, Internet RFC 5508/BCP 0148, kwiecień 2009.

[RFC5535] M. Bagnulo, *Hash-Based Addresses (HBA)*, Internet RFC 5535, czerwiec 2009.

[RFC5568] R. Koodli (red.), *Mobile IPv6 Fast Handovers*, Internet RFC 5568, lipiec 2009.

[RFC5790] H. Liu, W. Cao, H. Asaeda, *Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols*, Internet RFC 5790, luty 2010.

[RFC5837] A. Atlas (red.), R. Bonica (red.), C. Pignataro (red.), N. Shen, JR. Rivers, *Extending ICMP for Interface and Next-Hop Identification*, Internet RFC 5837, kwiecień 2010.

[RFC5927] F. Gont, *ICMP Attacks against TCP*, Internet RFC 5927 (informational), lipiec 2010.

[RFC5942] H. Singh, W. Beebee, E. Nordmark, *IPv6 Subnet Model: The Relationship between Links and Subnet Prefixes*, Internet RFC 5942, lipiec 2010.

[RFC5944] C. Perkins (red.), *IP Mobility Support for IPv4, Revised*, Internet RFC 5944, listopad 2010.

[RFC6104] T. Chown, S. Venaas, *Rogue IPv6 Advertisement Problem Statement*, Internet RFC 6104 (informational), luty 2011.

[RFC6106] J. Jeong, S. Park, L. Beloeil, S. Madanapalli, *IPv6 Router Advertisement Options for DNS Configuration*, Internet RFC 6106, listopad 2010.

[RFC6144] F. Baker, X. Li, C. Bao, K. Yin, *Framework for IPv4/IPv6 Translation*, Internet RFC 6144 (informational), kwiecień 2011.

[RFC6145] X. Li, C. Bao, F. Baker, *IP/ICMP Translation Algorithm*, Internet RFC 6145, kwiecień 2011.

[RFC6273] A. Kubec, S. Krishnan, S. Jiang, *The Secure Neighbor Discovery (SEND) Hash Threat Analysis*, Internet RFC 6273 (informational), czerwiec 2011.

[RFC6275] C. Perkins, D. Johnson, J. Arkko, *Mobility Support in IPv6*, Internet RFC 6275, czerwiec 2011.

[RFC6775] Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs), listopad 2012. *http://datatracker.ietf.org/doc/rfc6775/*

[SI] *http://www.iana.org/assignments/cga-message-types*

Rozdział 9. Broadcasting i lokalny multicasting

9.1. Wprowadzenie

W rozdziale 2. opisywaliśmy cztery kategorie adresów IP: *unicast*, *anycast*, *multicast* i *broadcast*. W protokole IPv4 wykorzystywane są wszystkie, w protokole IPv6 zrezygnowano z kategorii broadcast. W tym rozdziale przeanalizujemy szczegółowo dwa ostatnie z wymienionych — *multicast* i *broadcast* — pokazując m.in., w jaki sposób ich wykorzystywanie wspomagane jest przez mechanizmy warstwy łącza danych. Omówimy także protokoły IGMP (*Internet Group Management Protocol*, patrz [RFC3376]) oraz MLD (*Multicast Listener Discovery*, patrz [RFC3810]), wykorzystywane w protokołach (odpowiednio) IPv4 i IPv6 do zarządzania przynależnością hostów do poszczególnych grup multicast i generalnie do rozpoznawania adresów multicast używanych w poszczególnych podsieciach. Pominiemy natomiast całkowicie implementację trasowania ruchu multicast w sieciach rozległych, takich jak Internet, bo związane z nimi protokoły cechują się dużym stopniem komplikacji w porównaniu z używanymi w sieciach lokalnych i korporacyjnych, w których korzyści ze stosowania multicastingu są relatywnie największe. Czytelników zainteresowanych tematem odsyłamy do książki [EGW02].

Broadcasting i multicasting wykorzystywane są w aplikacjach do dwóch celów:

- **rozsyłania pakietów do wielu miejsc przeznaczenia**, nieodłącznie związanego z wszelkiego rodzaju serwisami informacyjnymi, powiadomieniami e-mailowymi itp.; znacznie mniej efektywną alternatywą dla multicastingu i broadcastingu jest w tym przypadku jawne rozsyłanie pakietów do każdego adresata z osobna, na poziomie protokołu TCP,
- **przepytywania (indagowania) serwerów przez stacje klienckie**, które w ten sposób zdobywają informacje dotyczące nieznanej (lub mało znanej) podsieci; tak właśnie urządzenia mobilne uzyskują — za pośrednictwem protokołu DHCP — adresy IP oraz rozpoznają pobliskie routery (o czym pisaliśmy w rozdziale 6.).

Chociaż opisany efekt można uzyskać za pomocą zarówno broadcastingu, jak i multicastingu, to jednak multicasting jest zdecydowanie środkiem preferowanym, ze względu na większą selektywność: można go ukierunkować na systemy świadczące (lub wykorzystujące)

określoną usługę, podczas gdy broadcasting obejmuje *wszystkie* węzły z danego zakresu — różnica ta stanie się wyraźniejsza, gdy przyjrzymy się szczegółom obu technik. Wybór między broadcastingiem a multicastingiem jest zatem wyborem między niekontrolowaną prostotą a selektywnością uzyskiwaną za cenę większej komplikacji.

Broadcasting odgrywał istotną rolę w protokole IPv4 od samych jego początków, multicasting pojawił się dopiero później, za sprawą publikacji [RFC1112]; w protokole IPv6 zrezygnowano natomiast zupełnie z broadcastingu. Generalnie zalety multicastingu i broadcastingu dostępne są dla aplikacji opierających swe działanie na protokole UDP (patrz rozdział 10.), który rozsyła pojedynczy komunikat do wielu miejsc przeznaczenia; protokół TCP, działający na bazie połączenia między dwoma punktami „adres-port", z natury wykorzystuje adresy unicast i (podobne do nich) adresy anycast.

Broadcasting i multicasting wykorzystywane są także przez istotne procesy systemowe, uruchamiane m.in. w protokołach trasowania, protokole ARP (w IPv4) i protokole ND (w IPv6). I choć początkowo multicasting traktowany był jako dodatek (*add-on*) do implementacji IP, we współczesnych systemach operacyjnych stał się integralną częścią tych implementacji. W protokole IPv4 multicasting jest mechanizmem użytecznym, choć niekoniecznym; natomiast protokół IPv6 nie mógłby nawet zaistnieć bez multicastingu, stanowiącego podstawę protokołu ND (patrz podrozdział 8.5) — usługi krytycznej także w komunikacji unicast.

9.2. Broadcasting

Broadcasting, zwany także **rozgłaszaniem**, to wysyłanie pojedynczego komunikatu do wszystkich potencjalnie możliwych odbiorców w sieci. Realizuje się go w sposób zgoła nieskomplikowany — router wysyła kopię komunikatu przez wszystkie interfejsy z wyjątkiem tego, przez który otrzymał wspomniany komunikat. Gdy w danej sieci lokalnej znajduje się wiele hostów, najbardziej efektywnym rozwiązaniem jest powierzenie funkcji powielania komunikatu mechanizmom warstwy łącza danych.

W rozdziale 3. opisywaliśmy sieci Ethernet realizujące broadcasting na poziomie warstwy łącza danych. Każda ramka ethernetowa zawiera 48-bitowe adresy źródłowy i docelowy. Większość pakietów IP przeznaczona jest dla konkretnych hostów, dla każdego z takich pakietów adres docelowy unicast konwertowany jest na adres MAC konkretnego węzła docelowego, przy użyciu ARP (w IPv4) lub ND (w IPv6); ramka przemieszcza się między dwoma konkretnymi węzłami, bez angażowania lub wiedzy pozostałych węzłów sieci. Takie właśnie ethernetowe adresy unicast znajdują się w pamięciach cache przełączników i mostków. Jednak niektóre ramki przeznaczone są do dostarczenia do wszystkich innych hostów w sieci (lub sieci wirtualnej — VLAN) — to właśnie jest broadcasting, którego przykład zastosowania widzieliśmy w rozdziale 4.

9.2.1. Adresy rozgłoszeniowe

W sieci Ethernet i podobnych sieciach adres multicast MAC to adres, w którym ustawiony jest na 1 najmłodszy bit w najbardziej znaczącym bajcie. Szczególnym przypadkiem takiego adres jest adres `ff:ff:ff:ff:ff:ff`, w którym wszystkie bity są jedynkami. Jak pamiętamy z rozdziału 2., w każdej podsieci specjalne znaczenie ma adres, w którym

„końcówka" identyfikująca numer hosta ma wartość maksymalną, czyli składa się z samych jedynek — to adres identyfikujący wszystkie hosty tej podsieci. Adres złożony *w całości* z jedynek — 255.255.255.255 — traktowany jest jako adres **ograniczonego rozgłaszania** (*limited broadcast*).

9.2.1.1. Przykład

W systemie Linux adresy ograniczonego rozgłaszania przyporządkowane poszczególnym interfejsom można poznać, wydając polecenie `ifconfig`:

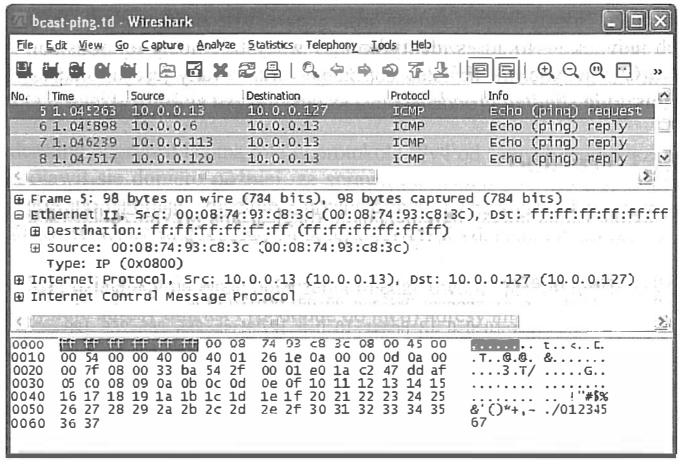
```
Linux% ifconfig eth0
eth0      Link encap:Ethernet HWaddr 00:08:74:93:CB:3C
          inet addr:10.0.0.13 Bcast:10.0.0.127 Mask:255.255.255.128
          inet6 addr: 2001:5c0:9ae2:0:208:74ff:fe93:c83c/64
                      Scope:Global
          inet6 addr: fe80::208:74ff:fe93:c83c/64
                      Scope:Link
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:426469 errors:0 dropped:0 overruns:1 frame:0
          TX packets:779338 errors:0 dropped:0 overruns:0 carrier:0
          collisions:298048 txqueuelen:1000
          RX bytes:44414543 (42.3 MiB) TX bytes:1094425223 (1.0 GiB)
          Interrupt:19 Base address:0xec00
```

Adres `10.0.0.127` to adres ograniczonego rozgłaszania wykorzystywany w sieci, do której przyłączone jest urządzenie `eth0`. Zgodnie z prefiksem sieci — `10.0.0.0/25` — numer hosta utworzony jest przez 32 – 25 = 7 najmłodszych bitów adresu, zatem, składając 25-bitowy prefiks `1010 0000 0000 0000 1` z ciągiem 7 bitów jedynkowych, otrzymujemy `1010 0000 0000 0000 1111 1111`, czyli `10.0.0.127`. W wykonywaniu tego rodzaju obliczeń, choć nie są one szczególnie skomplikowane, pomocny okazuje się prosty program użytkowy `ipcalc`.

Aby zaobserwować mechanizm broadcastingu w akcji, użyjemy prostego polecenia, w wyniku którego na wspomniany adres rozgłaszania wysłane zostaną pakiety ICMPv4 *Echo Request*:

```
Linux# ping -b 10.0.0.127
WARNING: pinging broadcast address
PING 10.0.0.127 (10.0.0.127) 56(84) bytes of data.
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=1.05 ms
64 bytes from 10.0.0.113: icmp_seq=1 ttl=64 time=1.55 ms (DUP!)
64 bytes from 10.0.0.120: icmp_seq=1 ttl=64 time=3.09 ms (DUP!)
--- 10.0.0.127 ping statistics ---
1 packets transmitted, 1 received, +2 duplicates,
0% packet loss, time 0ms
```

Jak wyjaśnialiśmy w punkcie 8.4.1, ponieważ każde z żądań *Echo* trafia do wielu hostów, na każde z tych żądań przychodzi ogólnie wiele odpowiedzi; program `ping`, kiedy zauważy powtórne nadejście odpowiedzi z tym samym numerem sekwencyjnym, opatruje ją w raporcie stosownym znacznikiem `(DUP!)`. Za pomocą programu Wireshark możemy zajrzeć do wnętrza przesyłanych pakietów i zaobserwować wykorzystywane w nich adresy (patrz rysunek 9.1).



Rysunek 9.1. *Komunikaty ICMPv4 Echo Request wysyłane w ramach rozgłaszania ukierunkowanego w lokalnej podsieci, enkapsulowane w ramkach rozgłoszeniowych łącza danych z adresem docelowym ff:ff:ff:ff:ff:ff*

Komunikaty *Echo Request* wysyłane są na adres `10.0.0.127`, który implementacja IPv4 rozpoznaje jako adres rozgłaszania ukierunkowanego, na podstawie informacji zawartych w lokalnych tablicach trasowania i konfiguracji interfejsu. W rezultacie w ramkę enkapsulującą wysyłany datagram wpisany zostaje adres docelowy `ff:ff:ff:ff:ff:ff`, nie jest więc konieczne sięganie do protokołu ARP w celu odnalezienia adresu MAC dla każdego węzła docelowego z osobna. Co więcej, host nadawca nie zna nawet potencjalnych odbiorców rzeczonego datagramu — do czasu, aż nie nadejdą z ich strony odpowiedzi *Echo Reply*. Adresem źródłowym zarówno IP, jak i warstwy łącza danych jest natomiast adres unicast hosta nadawcy (`10.0.0.13`).

Zauważmy także, iż wszystkie odpowiedzi *Echo Reply* kierowane są na adres hosta nadawcy żądania (`10.0.0.13`), zaś w polu adresu źródłowego poszczególnych pakietów znajdują się adresy węzłów odpowiadających — `10.0.0.6`, `10.0.0.113` i `10.0.0.120`. W ten oto sposób host `10.0.0.13` uzyskuje informację o adresach innych węzłów znajdujących się w tej samej podsieci — informację, która dotąd pozostawała dla niego nieznana. To właśnie jedna z cech broadcastingu i (jak się za chwilę przekonamy) także multicastingu.

9.2.2. Rozsyłanie datagramów rozgłoszeniowych

Aplikacje realizujące broadcasting zwykle używają do tego celu protokołów UDP lub ICMPv4, wywołując odpowiednie funkcje API. Ponieważ jednak broadcasting wiąże się najczęściej z generowaniem sporego ruchu, grożącego przeciążeniem sieci, niektóre systemy operacyjne wymagają dodatkowo od wspomnianych aplikacji ustawienia specjalnego znacznika (`SO_BROADCAST`) w parametrze znaczników przekazywanym do wywoływanych funkcji API; stanowi to ze strony programisty dodatkowe zapewnienie, że uruchomienie broadcastingu przez wywoływaną funkcję jest działaniem zamierzonym, a nie dziełem przypadku czy przeoczenia. Zasada ta przekłada się często na parametry wywołania

tychże aplikacji z poziomu wiersza poleceń, przykładowo dla programu `ping` uruchamianego w Linuksie wymagane jest w związku z tym użycie parametru `-b` jako potwierdzenia, iż użytkownik wie, co robi:

```
Linux% ping 10.0.0.127
Do you want to ping broadcast? Then -b
```

Brak parametru `-b` w wywołaniu programu powoduje nieustawienie wspomnianego znacznika `SO_BROADCAST` w wywołaniach funkcji API i zwracanie sygnału o błędzie przez te funkcje, a w konsekwencji stosowną reakcję programu `ping` wobec użytkownika.

Do określenia, które interfejsy wykorzystywane są na potrzeby broadcastingu, następuje odwołanie do tabel trasowania IPv4. Poniższy raport, pochodzący z systemu Windows 7 (podobnie jest w Windows Vista), ukazuje listę interfejsów wraz z informacjami dotyczącymi broadcastingu (dla czytelności usunęliśmy inne informacje).

```
C:\> netstat -rn
===========================================================================
Lista interfejsów
10 ...02 00 4c 4f 4f 50 ...... Microsoft Loopback Adapter
9 ...00 13 02 20 b9 18 ...... Intel(R) PRO/Wireless 3945ABG Network Connection
8 ...00 14 22 f4 19 5f ...... Broadcom 440x 10/100 Integrated Controller
1 ........................... Software Loopback Interface 1
12 ...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter
13 ...00 00 00 00 00 00 00 e0 Microsoft ISATAP Adapter #2
11 ...00 00 00 00 00 00 00 e0 isatap.{2523E0D6-ABE2-42F1-8188-6AA108FEA1EA}
===========================================================================

Tabela tras IPv4
===========================================================================
Aktywne trasy:
Miejsce docelowe w sieci   Maska sieci      Brama          Interfejs Metryka
          0.0.0.0          0.0.0.0        10.0.0.1      10.0.0.57        25
       10.0.0.127  255.255.255.255         On-link       10.0.0.57       281
  127.255.255.255  255.255.255.255         On-link        127.0.0.1       306
  169.254.255.255  255.255.255.255         On-link 169.254.57.240       286
  255.255.255.255  255.255.255.255         On-link        127.0.0.1       306
  255.255.255.255  255.255.255.255         On-link 169.254.57.240       286
  255.255.255.255  255.255.255.255         On-link       10.0.0.57       281
```

W pierwszej części raportu widzimy listę siedmiu interfejsów sieciowych: wirtualnej pętli zwrotnej (*loopback*), Wi-Fi, Ethernetu przewodowego (tu wyłączonego), kolejnej pętli zwrotnej oraz trzech niestandardowych interfejsów ISATAP (*Intra-Site Automatic Tunnel Addressing Protocol*, patrz [RFC5214] i [RFC5579]) — technologii wspierającej transfer pakietów IPv6 przez sieć IPv4 między dwoma hostami implementującymi dualny stos TCP/IP.

W drugiej części raportu mamy siedem pozycji pomocnych w ustaleniu trasy, którą kierowany ma być ruch broadcast. Pierwsza pozycja, domyślna, o masce `0.0.0.0`, zgodna z dowolnym adresem docelowym, będzie używana do broadcastingu prowadzącego poza sieć, o ile taką możliwość przewidziano w konfiguracji tejże sieci — rozgłaszanie ukierunkowane prowadzące poza sieć jest zazwyczaj blokowane przez routery w celu zapobiegania ewentualnym problemom, zgodnie z sugestią przedstawioną w dokumencie [RFC2644].

Trzy kolejne pozycje, wyznaczone przez maskę do obsługi rozgłaszania ograniczonego, prowadzą do interfejsów o adresach 10.0.0.57, 127.0.0.1 i 169.254.57.240 — dwa ostatnie identyfikują programowe interfejsy pętli zwrotnej.

Widać wyraźnie, w jaki sposób wyznaczana jest trasa dla rozgłaszania ukierunkowanego: prefiks sieci konkatenowany jest z ciągiem jedynek w części identyfikującej numer hosta, całość składana jest za pomocą operacji AND z maską /32 (czyli 255.255.255.255). Status On-link w kolumnie *Brama* oznacza dostarczanie bezpośrednie (*direct delivery* — patrz rozdział 5.) przez interfejs wskazywany w kolumnie *Interface*. Ponieważ w grę wchodzi tylko jeden interfejs, wartość w kolumnie *Metryka* jest bez znaczenia.

Trzy ostatnie pozycje związane są z adresem ograniczonego rozgłaszania (255.255.255.255). Jest to w pewnym sensie adres multicast, bo nie jest bezpośrednio powiązany z adresami używanymi w którejkolwiek z bezpośrednio przyłączonych sieci. Nie jest więc oczywiste, który (które) z interfejsów powinien uczestniczyć w forwardowaniu ograniczonego rozgłaszania, sekcja 3.3.6 dokumentu [RFC1122] (*Requirements for Internet Hosts*) dostarcza nikłych sugestii w tym względzie. W stosunku do hostów multihomed oczywiste pozostają więc tylko dwie skrajności: forwardowanie ograniczonego rozgłaszania równolegle przez *wszystkie* „pasujące" interfejsy oraz forwardowanie ograniczonego rozgłaszania przez jeden arbitralnie wybrany interfejs spośród „pasujących". Wspomniana specyfikacja nie rozstrzyga tej kwestii.

W konsekwencji realizacja forwardowania ograniczonego rozgłaszania w hostach multihomed uzależniona jest ściśle od implementacji. Większość systemów, w tym Linux i FreeBSD, stosuje wybór pierwszego „pasującego" interfejsu, przy czym FreeBSD wykonuje jednocześnie konwersję adresu ograniczonego rozgłaszania na adres ukierunkowanego rozgłaszania w podsieci, do której prowadzi wybrany interfejs (aplikacje mogą jednak zakazywać takiej konwersji, ustawiając w tym celu znacznik IP_ONESBCAST). Systemy grupy Windows nie są konsekwentne w opisywanym względzie: aż do wersji 2000 forwardowanie realizowane było równolegle przez wszystkie pasujące interfejsy, od wersji XP projektanci zmienili strategię na wybór „optymalnego" interfejsu, czyli interfejsu o najmniejszej metryce. W prezentowanym przykładzie jest to interfejs o adresie 10.0.0.57.

9.3. Multicasting

Można zredukować natężenie ruchu generowanego w ramach broadcastigu, ograniczając wysyłanie komunikatów tylko do tych adresatów, którzy faktycznie są nimi zainteresowani; takie zredukowane rozgłaszanie nazywane jest *multicastingiem*. Zasadniczo multicasting może być realizowany na dwa sposoby: poprzez jawne wskazywanie adresatów przez nadawcę bądź też w drodze okazywania przez wspomnianych adresatów zainteresowania komunikatami określonej kategorii; w tym drugim przypadku sieć przyjmuje na siebie zadanie odpowiedniego kierowania poszczególnych kategorii ruchu wyłącznie do zainteresowanych nim adresatów. Implementowanie multicastingu jest zadaniem znacznie trudniejszym niż implementowanie broadcastingu, bo wiąże się z koniecznością utrzymywania *informacji o stanie* poszczególnych grup multicast.

W modelu TCP/IP przyjęto drugą z wymienionych filozofii multicastingu: zainteresowany odbiorca zgłasza akces do konkretnej grupy multicast identyfikowanej za pomocą określonego adresu grupowego; opcjonalnie adresat ów zawęzić może także grupę po-

tencjalnych nadawców, od których życzy sobie (albo nie życzy) otrzymywać komunikaty w trybie multicast. Informacja o przynależności poszczególnych hostów do poszczególnych grup multicast utrzymywana jest przez hosty i routery w postaci wektorów, których pozycje mają charakter **miękkiego stanu** (*soft state* — patrz podrozdział 4.6): w ten sposób określa się potocznie informację, której istnienie wymaga periodycznego odświeżania — jego brak w założonym interwale czasowym powoduje zanik tejże informacji. Ruch multicast kierowany na nieistniejący adres multicast (czyli adres multicast, do którego nie zgłosił akcesu żaden węzeł) jest bądź to odrzucany, bądź też przekształcany na ruch broadcasting, zależnie od implementacji.

Choć nadmiar ruchu generowanego w ramach broadcastingu staje się szczególnie odczuwalny w dużych sieciach rozległych, może stanowić nadmierne obciążenie także dla sieci lokalnych, gdzie każdy host należący do tej samej podsieci lub sieci VLAN przetwarzać musi wszystkie pakiety rozgłoszeniowe. Dzięki multicastingowi IP można wykonywać te same zadania znacznie mniejszym kosztem: określone komunikaty trafiają wyłącznie do adresatów, którzy sobie tego życzą, a pakiety wysyłane są tylko na te łącza, które prowadzą do wspomnianych adresatów, przy czym na każde z łączy wysyłana jest tylko jedna kopia pakietu. Oczywiście, aplikacje zainteresowane udziałem w komunikacji multicast muszą (za pomocą jakiegoś mechanizmu) informować o tym zamiarze implementacje protokołu IP funkcjonujące w ich komputerach, w wyniku czego implementacje te nastawione zostają na odbiór pakietów spełniających określone kryteria.

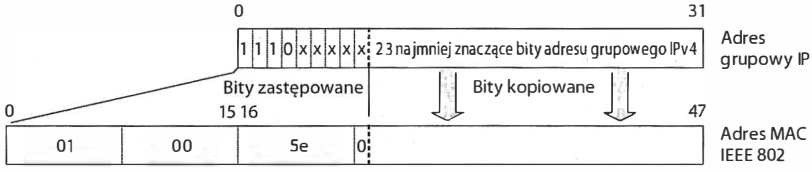
Idea multicastingu IP wywodzi się z koncepcji adresowania grupowego na poziomie warstwy łącza danych w sieciach, takich jak Ethernet. Zgodnie z tą koncepcją, każda stacja zainteresowana odbiorem ruchu wysyłanego na określony adres grupowy, bez względu na nadawcę, wybiera ten adres w sposób jawny. Selekcja tolerowanych nadawców ma w tym przypadku formę „wszystko albo nic" — jeśli stacja w ogóle odbierać ma pakiety kierowane na określony adres grupowy, musi akceptować je wszystkie bez względu na adres źródłowy. W związku z tym ten wariant multicastingu oznaczany bywa akronimem ASM (*Any-Source Multicast* — multicasting z dowolnego źródła). Wraz z rozwojem protokołu IP wprowadzono jednak do multicastingu możliwość selektywnego wyboru źródeł, z których odbiorca uczestnik określonej grupy życzy sobie (albo nie życzy sobie) otrzymywać komunikaty multicast; ten wariant, opisywany w dokumencie [RFC4607], opatrzony został akronimem SSM (*Source-Specific Multicast* — multicasting ze specyficznego źródła). Generalnie model usługowy SSM jest łatwiejszy w implementacji niż ASM: łatwiej sprawować kontrolę nad pojedynczym źródłem niż nad lokalizacjami wielu rozproszonych źródeł. Jednak w wielu sieciach lokalnych obsługa obu modeli przebiega w sposób niemal identyczny, będziemy więc dalej w tym rozdziale traktować oba warianty w sposób łączny, zwracając jedynie uwagę na różnice między nimi, tam gdzie będzie to istotne. Rozpoczniemy od przyjrzenia się, w jaki sposób multicasting IP zgodny z technologią IEEE LAN wykorzystuje adresowanie grupowe na poziomie adresów MAC.

9.3.1. Konwersja adresów grupowych IP na adresy MAC IEEE-802

W pakietach wysyłanych na adresy unicast w Ethernecie i podobnych sieciach docelowy adres MAC odpowiadający adresowi IP uzyskiwany jest za pomocą odpowiednich protokołów — ARP w IPv4 (patrz rozdział 4.) i ND w IPv6 (patrz rozdział 8.). Z kolei adresy rozgłoszeniowe IP konwertowane są na „dobrze znane" adresy rozgłoszeniowe

Ethernetu, bez konieczności sięgania do wspomnianych protokołów. Naturalne staje się więc w tym momencie pytanie: jak konwersja adresów IP na adresy MAC odbywa się w multicastingu? Jest zrozumiałe, iż pożądanym rozwiązaniem byłaby konwersja wykonywana na drodze automatycznego mapowania, bez uciekania się do protokołów pomocniczych — i rzeczywiście: przedstawimy szczegóły takiego mapowania na przykładzie sieci IEEE 802, do których zaliczają się m.in. Ethernet i Wi-Fi — czyli sieci, w których multicasting stosowany jest najczęściej.

Zobaczmy na początek, jak mają się sprawy na gruncie IPv4. Na stronie [OUI36] widoczna jest lista tzw. identyfikatorów OUI (*Organizationally Unique Identifier* — identyfikator unikatowy dla organizacji) stanowiących unikatowe 24-bitowe liczby całkowite, przydzielanych centralnie przez IANA producentom sprzętu jako prefiksy na potrzeby unikatowego numerowania produktów, standardów itp. Na swoje potrzeby IANA zarezerwowała sobie dwa OUI w postaci `00:00:5e` oraz `01:00:5e` — pierwszy na potrzeby formowania adresów MAC unicast, drugi na potrzeby formowania adresów MAC multicast. Daje to możliwość tworzenia adresów MAC multicast z zakresu od `01:00:5e:00:00:00` do `01:00:5e:ff:ff:ff`; na potrzeby konwertowania adresów multicast IPv4 IANA przeznaczyła pierwszą połowę tego zakresu — od `01:00:5e:00:00:00` do `01:00:5e:7f:ff:ff`. Najbardziej znaczący bit w czwartym bajcie adresu MAC jest więc zawsze zerowy, pozostałe 23 bity tegoż adresu kopiowane są z najmniej znaczących pozycji adresu IPv4. Tak więc np. adres `224.0.1.17` konwertowany jest na adres MAC `01:00:5e:00:01:11` (patrz rysunek 9.2).



Przykład: 224.0.1.17 → 01:00:5e:00:01:11

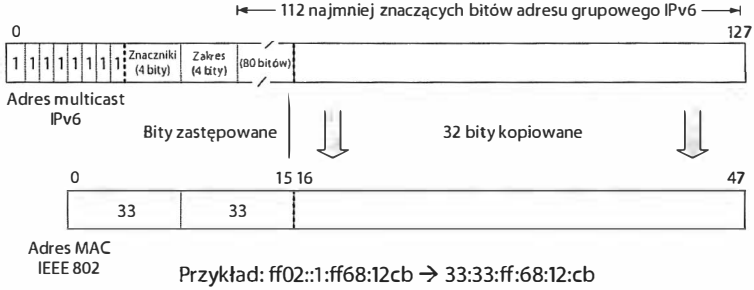
Rysunek 9.2. *Mapowanie adresu IPv4 na adres MAC IEEE-802. Do prefiksu 01:00:5e dołączony zostaje bit zerowy i 23 najmniej znaczące bity adresu IP. Ponieważ można w ten sposób utworzyć 2^{23} różnych adresów MAC, a liczba możliwych adresów IPv4 multicast wynosi 2^{28}, więc odwzorowanie nie jest jednoznaczne — na dany adres MAC mapowane są $2^{28-23} = 32$ różne adresy IPv4*

Wykorzystujemy tu przedstawienie uporządkowania bitów zgodnie z konwencją notacyjną używaną w standardach internetowych, odzwierciedlającą ułożenie bitów w pamięci. W dokumentach IEEE bity prezentowane są w kolejności ich wysyłania[1].

[1] Gdy posługujemy się systemem pisowni „od lewej do prawej" (obowiązującym m.in. we wszystkich językach Europy), zapisując liczbę (dziesiętną lub binarną) rozpoczynamy od cyfry *najbardziej znaczącej*, czyli w przypadku liczby binarnej od najbardziej znaczącego bitu (w zapisie szesnastkowym od bardziej znaczącej tetrady); przykładowo liczba „siedemdziesiąt osiem" przyjmuje w tej konwencji postać `78`, `01001110` lub `0x4E`. Gdy bajt ma zostać przesłany poprzez medium transmisji szeregowej, czyli w podziale na poszczególne bity, przesyłanie rozpoczyna się od bitu *najmniej znaczącego*: wspomniany bajt o dziesiętnej wartości `78` przesłany zostanie jako ciąg bitów (kolejno) `0, 1, 1, 1, 0, 0, 1, 0`. Innymi słowy, zwyczajowy zapis liczby binarnej opiera się na konwencji *big-endian*, podczas gdy techniki transmisji szeregowej hołdują konwencji *litte-endian*. Autor, mówiąc o „konwencji notacyjnej używanej w standardach internetowych" ma na myśli pierwszą z powyższych, czyli *big-endian*, a przeciwstawia ją konwencji *little-endian* używanej w dokumentach IEEE — *przyp. tłum.*

Jak pamiętamy z rozdziału 2., w protokole IPv4 adresami multicast są adresy dawnej klasy D, czyli adresy z zakresu od 224.0.0.0 do 239.255.255.255. Cztery pierwsze bity mają ustaloną postać 1110, liczba różnych adresów wynosi więc 2^{28} — czyli 32 razy więcej niż wynosi liczba możliwych adresów MAC tworzonych według zaprezentowanego przepisu. *Odwzorowanie nie jest więc jednoznaczne* — na dany adres MAC konwertowane są $2^{28-23} = 32$ różne adresy IPv4 — przykładowo każdy z adresów 224.128.64.32, 224.0.64.32 i 226.0.64.32 (szesnastkowo odpowiednio 0xe0804020, 0xe0004020 i 0xe2004020) mapowany jest na ten sam adres 01:00:5e:00:40:20.

W protokole IPv6 adresy MAC multicast rozpoczynają się od prefiksu 33:33, pozostałe 32 bity kopiowane są z najmniej znaczących pozycji adresu IPv6 (patrz rysunek 9.3). To odwzorowanie także nie jest jednoznaczne. W adresie multicast IPv6 najbardziej znaczący baj ma wartość 0xff, kolejny bajt przeznaczony jest na informację o zakresie i inne znaczniki. Dziedzinę odwzorowania stanowi więc zbiór $2^{128-16} = 2^{112}$ adresów IPv6, po stronie przeciwdziedziny mamy 2^{32} różnych adresów MAC. Na ten sam adres MAC mapowanych jest więc $2^{112-32} = 2^{80}$ różnych adresów IPv6.



Rysunek 9.3. *Mapowanie adresu IPv6 na adres MAC IEEE-802. Do prefiksu 33:33 dołączone zostają 32 najmniej znaczące bity adresu IP. Ponieważ można w ten sposób utworzyć 2^{32} różnych adresów MAC, a liczba możliwych adresów IPv6 multicast wynosi 2^{112}, więc odwzorowanie nie jest jednoznaczne — na dany adres MAC mapowane jest 2^{80} różnych adresów IPv6*

9.3.2. Przykłady

W poprzednim przykładzie pokazaliśmy zastosowanie rozgłaszania komunikatów ICMPv4 *Echo Request* do rozpoznawania hostów znajdujących się w tej samej podsieci, co nadawca tych komunikatów. W podobny sposób rozpoznać można obecność w podsieci hostów oferujących określoną usługę: należy mianowicie wysłać komunikat *Echo Request* na adres multicast reprezentujący grupę, do której hosty świadczące wspomnianą usługę zgłosiły akces. Zilustrujemy tę zasadę przy użyciu usługi *Multicast DNS* (mDNS — patrz [CK11]) — świadczące ją serwery przynależą do grupy multicast o adresie 224.0.0.251.

```
Linux% ping 224.0.0.251
PING 224.0.0.251 (224.0.0.251) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_seq=1 ttl=60 time=1.10 ms
64 bytes from 10.0.0.11: icmp_seq=1 ttl=60 time=1.60 ms (DUP!)
64 bytes from 10.0.0.120: icmp_seq=1 ttl=64 time=2.59 ms (DUP!)
--- 224.0.0.251 ping statistics ---
1 packets transmitted, 1 received, +2 duplicates,
0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.109/1.767/2.590/0.615 ms
```

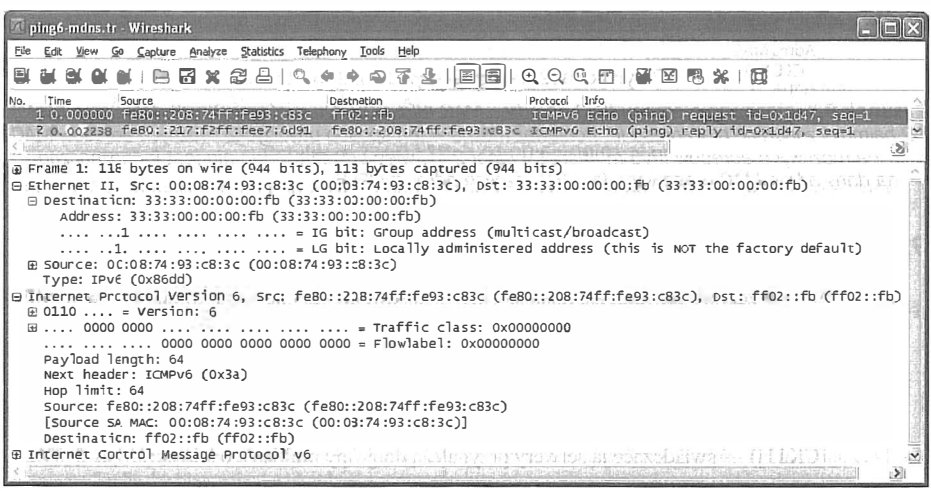
Odpowiedzi *Echo Reply* nadesłane przez hosty 10.0.0.2, 10.0.0.11 i 10.0.0.120 są dowodem ich członkowstwa w grupie 224.0.0.251; zauważmy, że nie są to te same hosty, które odpowiedziały na rozgłaszane komunikaty *Echo Request*, wysyłane na adres 10.0.0.127. Nic w tym dziwnego, przecież nie wszystkie hosty są serwerami usługi mDNS.

Multicast DNS (mDNS) to usługa zaprojektowania na użytek sieci z zerową (bezwysiłkową) konfiguracją systemu i urządzeń. Stanowi ona część usługi *Bonjour* w systemach firmy Apple; Microsoft promuje alternatywny protokół o podobnej funkcjonalności, znany pod akronimem LLMNR (*Link Local Multicast Name Resolution* — lokalne dla łącza odwzorowywanie nazw multicast, patrz [RFC4795]). Żaden z wymienionych protokołów nie został dotąd uznany przez IETF za standard, jednakże mDNS posiada dłuższą historię niż LLMNR. Powrócimy do tego tematu w rozdziale 11.

Podobny eksperyment można wykonać w odniesieniu do implementacji IPv6:

```
Linux% ping6 -I eth0 ff02::fb
PING ff02::fb(ff02::fb) from fe80::208:74ff:fe93:c83c eth0:
56 data bytes
64 bytes from fe80::217:f2ff:fee7:6d91: icmp_seq=1 ttl=64 time=2.76 ms
--- ff02::fb ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 2.768/2.768/2.768/0.000 ms
```

Tym razem jawnie wskazaliśmy interfejs dla wychodzących pakietów ICMPv6 *Echo Request*, co pozwoliło programowi wybrać odpowiedni adres źródłowy; jak widać na rysunku 9.4, jest to lokalny dla łącza adres związany z urządzeniem eth0.



Rysunek 9.4. *Komunikat ICMPv6 Echo Request wysyłany przez interfejs eth0 na podstawie jego adresu lokalnego dla łącza; odpowiedzi Echo Reply kierowane będą na ten właśnie adres*

Pakiety ICMPv6 *Echo Request* i *Echo Reply* posiadają wspólny *Identyfikator* 0x1d47 i *Numer sekwencyjny* równy 1. Adres źródłowy jest zawsze adresem lokalnym dla łącza. Adresem docelowym dla komunikatów *Echo Request* jest adres multicast ff02::fb, który mapowany jest na adres MAC 33:33:00:00:00:fb, zgodnie z opisaną wcześniej regułą.

Odpowiedzi *Echo Reply* odsyłane są natomiast na adres unicast nadawcy `fe80::208:74ff:fe93:c83c`, przy czym adresem źródłowym jest adres unicast hosta odpowiadającego `fe80::217:f2ff:fee7:6d91` — zauważmy, że ten ostatni wybrany został jako lokalny dla łącza, czyli zgodny pod względem zakresu z adresem docelowym, co wynika bezpośrednio ze strategii selekcji adresów, opisanej w punktach 5.4.3 i 5.6.2 tej książki (zwróćmy także uwagę na korelację między schematem z rysunku 5.16 a informacjami wyświetlanymi w oknie na rysunku 9.4).

9.3.3. Rozsyłanie datagramów multicastingu

Przy tworzeniu pakietu IP przeznaczonego do wysłania pojawia się problem wyboru interfejsu, przez który wysłanie to ma być zrealizowane, oraz wyboru adresu, jaki ma zostać umieszczony we wspomnianym pakiecie jako źródłowy. Problem ten jest szczególnie istotny w wersji IPv6, gdzie typową sytuacją jest przyporządkowywanie wielu adresów IP temu samemu interfejsowi. Aby zrozumieć mechanizm tego wyboru, przeanalizujmy raport z zawartości tablicy forwardowania hosta; zarówno w Windows, jak i Linuksie raport ten wykonuje się za pomocą polecenia `netstat`, poniższy przykład pochodzi z systemu Windows 7:

```
C:\> netstat -rn
... Lista interfejsów ...
Tabela tras IPv4
===========================================================================
Aktywne trasy:
Miejsce docelowe w sieci   Maska sieci      Brama          Interfejs Metryka
          0.0.0.0          0.0.0.0         10.0.0.1     10.0.0.57        25
        224.0.0.0        240.0.0.0         On-link      127.0.0.1       306
        224.0.0.0        240.0.0.0         On-link 169.254.57.240       286
        224.0.0.0        240.0.0.0         On-link      10.0.0.57       281
  255.255.255.255  255.255.255.255         On-link      127.0.0.1       306
  255.255.255.255  255.255.255.255         On-link 169.254.57.240       286
  255.255.255.255  255.255.255.255         On-link      10.0.0.57       281
===========================================================================
Trasy trwałe:
  Brak
===========================================================================

Tabela tras IPv6
===========================================================================
Aktywne trasy:
 Jeśli[2] Metryka Miejsce docelowe w sieci      Brama
  9    281 ::/0                     fe80::204:5aff:fe9f:9e80
  1    306 ff00::/8                 On-link
 10    286 ff00::/8                 On-link
  9    281 ff00::/8                 On-link
===========================================================================
Trasy trwałe:
  Brak
```

[2] Ten kuriozalny błąd językowy pojawił się w Windows Vista i wciąż pokutuje w Windows 7: autorzy polskiej wersji Windows błędnie przetłumaczyli angielskie „If" jako „Jeśli", podczas gdy oryginalnie jest to skrót od *Interface*, zatem w wersji polskiej kolumna powinna być zatytułowana *Interfejs — przyp. tłum.*

Jak widzimy, domyślna trasa dla ruchu IPv4 prowadzi do bramy `10.0.0.1` przez interfejs `10.0.0.57`. Mimo iż maska `0.0.0.0` „pasuje" do ruchu multicast (bo pasuje do każdego), w tablicy istnieją lepsze dopasowania: pozycje specyfikujące miejsce docelowe `224.0.0.0/4` (z maską podsieci `240.0.0.0`) wskazują trzy różne interfejsy, spośród których domyślnie wybierany jest ten o najmniejszej metryce (czyli `10.0.0.57`, z metryką `281`), chyba że aplikacja zażyczy sobie innego wyboru.

W protokole IPv6 nie ma adresów broadcast, a wszystkie adresy multicast rozpoczynają się od bajta `0xff`, więc do obsługi ruchu multicast mogą być użyte interfejsy `1`, `9` i `10`; najmniejszą metrykę posiada interfejs `9` (notabene ten sam, który obsługuje multicasting w IPv4) i to on zostaje wybrany do pełnienia (domyślnie) tej funkcji (zauważmy, że jest to również interfejs domyślny dla ruchu unicast). Ponownie, aplikacja może jawnie zażyczyć sobie innego interfejsu.

Przyporządkowanie adresów IP poszczególnym interfejsom można w Windows rozpoznać za pomocą polecenia `ipconfig /all`. W Linuksie analogiczny raport można uzyskać osobno dla IPv4 albo IPv6, wydając polecenie `netstat` z argumentami wskazującymi kategorię żądanej informacji. W przypadku protokołu IPv4 nie ma nic interesującego w kwestii ruchu multicast, ponieważ nie wyznaczono dla niego oddzielnej trasy i jego obsługa odbywa się przez konwencjonalną trasę domyślną. W przypadku IPv6 jest jednak inaczej:

```
Linux% netstat -rn -A inet6
Kernel IPv6 routing table
Destination  Next Hop            Flags Metric Ref    Use  Iface
ff00::/8     ::                   U       256  0      0  eth0
```

Nie wskazano tu bezpośrednio następnego przeskoku — w kolumnie *Next Hop* widnieje adres nieokreślony — wyznaczono jednak interfejs `eth0` jako domyślny do obsługi multicastingu. Ustawiony znacznik `U` oznacza, że trasa jest dostępna (*usable*), brak znacznika `G` identyfikuje ją jako trasę „na łączu", niewymagającą forwardowania przez router.

9.3.4. Odbieranie datagramów multicastingu

Fundamentalną dla multicastingu koncepcją jest *dołączanie do grupy multicast* (lub kilku takich grup) przez określony interfejs hosta oraz *opuszczanie tejże grupy*. Uczestnictwo w grupie multicast realizowane jest na poziomie interfejsu, z inicjatywy procesu: dany proces może specyfikować włączanie kilku interfejsów do tej samej grupy, jednego interfejsu do kilku grup bądź też realizować rozwiązania stanowiące kombinację tych skrajności (pod pojęciem „procesu" rozumiemy tu program wykonywany pod kontrolą systemu operacyjnego, często w imieniu konkretnego użytkownika). Członkostwo interfejsu w grupie multicast ma charakter dynamiczny, zmienia się w miarę uruchamiania i kończenia poszczególnych procesów. Oprócz statusu członkostwa w określonych grupach multicast, nie mniej ważna dla konkretnego hosta jest kwestia określenia adresów źródłowych, z których zainteresowany jest otrzymywaniem komunikatów w ramach określonej grupy. Oba te aspekty multicastingu są obowiązkowymi składnikami każdej implementacji multicastingu, dostępnymi za pośrednictwem określonych funkcji API.

9.3.4.1. Przykład

Uczestnictwo konkretnych interfejsów w poszczególnych grupach multicast można poznać za pomocą odpowiednich poleceń systemu operacyjnego. W systemie Windows polecenia te są komponentami pakietu `netsh` — dla IPv6 polecenie ma następującą postać:

```
C:\> netsh interface ipv6 show joins
Interfejs 1: Loopback Pseudo-Interface 1
Zakres      Odwołania   Ost   Adres
----------  ----------  ----  ---------------------------------
0                    1  Tak   ff02::c

Interfejs 8: Połączenie lokalne
Zakres      Odwołania   Ost   Adres
----------  ----------  ----  ---------------------------------
0                    0  Tak   ff01::1
0                    0  Tak   ff02::1
0                    1  Tak   ff02::c
0                    1  Tak   ff02::1:3
0                    1  Tak   ff02::1:ffdc:fc85
```

(Analogiczne informacje dla protokołu IPv4 można uzyskać, zastępując parametr `ipv6` przez `ip`). Widzimy (naturalne dla IPv6) przyporządkowanie kilku adresów IP do pojedynczego interfejsu. Pierwszym z wymienionych jest lokalny interfejs pętli zwrotnej, uczestniczący w grupie multicast protokołu SSDP (*Simple Service Discovery Protocol*) — pisaliśmy o nim w punkcie 7.5.3 — identyfikowany przez lokalny dla łącza adres `ff02::c`.

SSDP opisywany jest w (unieważnionym) dokumencie szkicu [GCLG99] autorstwa firm Microsoft i Hewlett-Packard. SSDP dostępny jest także w środowisku IPv4 za pośrednictwem adresu `239.255.255.250` i portu UDP 1900.

Na drugim z interfejsów widzimy najpierw przyporządkowanie adresów grupowych dla wszystkich węzłów (*All Nodes*) w zakresie lokalnym dla łącza (`ff02::1`) i w zakresie lokalnym dla węzła (`ff01::1`), adresu grupowego dla protokołu SSDP (`ff02::1`) oraz adresu grupowego dla protokołu LLMNR (`ff02::1:3`). Ostatni z adresów (`ff02::1:ffdc:fc85`) to adres typu *Solicited-Node multicast* dla węzła, wykorzystywany przez protokół ND w IPv6 (jak pamiętamy, w IPv6 uzyskiwanie adresów MAC pobliskich węzłów odbywa się za pomocą multicastingu realizowanego przez ND, odmiennie niż w IPv4, gdzie funkcję tę powierzono protokołowi ARP).

W Linuksie podobny raport uzyskać można za pomocą polecenia `netstat`:

```
Linux% netstat -gn
IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      224.0.0.1
eth1            1      224.0.0.1
lo              1      ff02::1
eth1            1      ff02::1:ff2a:1988
eth1            1      ff02::1
```

Jak widać, w raporcie tym uwzględnione są wszystkie interfejsy zaangażowane w grupy multicast — najpierw te z adresami IPv4, następnie te z adresami IPv6. Oba interfejsy — `lo` i `eth1` — uczestniczą w grupie wszystkich hostów (*All Hosts* — `224.0.0.1`) i grupie wszystkich węzłów (*All Nodes* — lokalny dla łącza adres `ff02::1`). Interfejs `eth1` uczestniczy także w grupie *Solicited-Node*, identyfikowanej przez adres `ff02::1:ff2a:1988`.

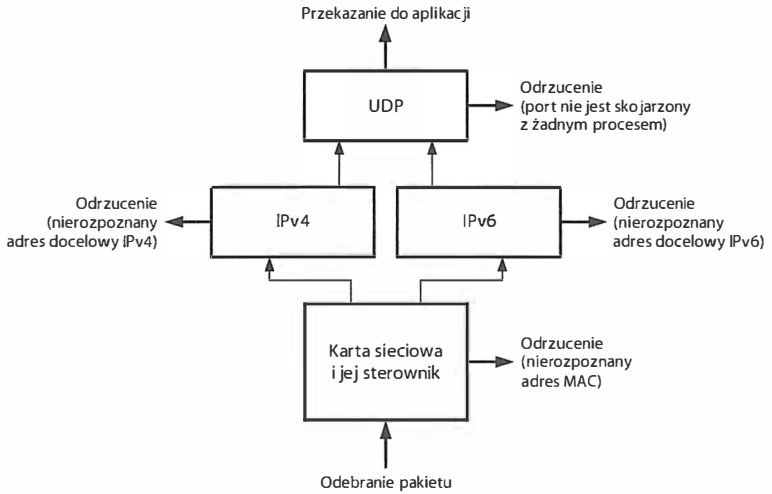
Celem dołączania do grup multicast jest zwykle zapewnienie sobie możliwości odbierania komunikatów adresowanych do tych grup; *wysyłanie* komunikatów multicast do określonej grupy wcale nie wymaga uczestnictwa w tej grupie. Mimo to, często zdarza się tak, że procesy współdziałające z jakąś grupą multicast dołączają do tej grupy; uczestnik grupy może więc być nie tylko biernym *odbiorcą* adresowanych do niej komunikatów, lecz także czynnym ich *nadawcą*. W tym kontekście istotnego znaczenia nabiera interpretacja pewnego szczególnego przypadku — interakcji procesów tworzących *klikę*, czyli uczestniczących *w tej samej grupie multicast*, działających *w tym samym hoście* i dołączonych do wspomnianej grupy *na tym samym interfejsie*. Otóż, domyślnie, procesy należące do kliki nie informują się nawzajem o wysyłanych komunikatach — komunikat wysłany na adres multicast wspólnej grupy przez jeden z procesów nie trafia do pozostałych. To domyślne zachowanie ulega zmianie, gdy któryś z procesów ustawi znacznik `IP_MULTICAST_LOOP` w wywołaniach funkcji z biblioteki *socket API*, przy czym efekt tego ustawienia jest odmienny w różnych systemach operacyjnych. W Linuksie znacznik `IP_MULTICAST_LOOP` interpretowany jest *na poziomie wysyłania*: proces wysyłający komunikat wyraża w ten sposób życzenie, by komunikat ten dotarł do pozostałych procesów w klice, przy czym nie ma znaczenia, jak indywidualnie poustawiały one ów znacznik na własne potrzeby. W Windows znacznik `IP_MULTICAST_LOOP` interpretowany jest na poziomie *odbierania* — ustawiając go, proces zapewnia sobie otrzymywanie wszystkich komunikatów wysyłanych przez dowolny inny proces ze wspólnej kliki i ponownie nie ma znaczenia, jak inne procesy tej kliki poustawiały ów znacznik na własne potrzeby.

9.3.5. Filtrowanie adresów przez host

Integralną częścią przetwarzania otrzymywanych przez host datagramów adresowanych do grup multicast jest *filtrowanie* nadchodzących datagramów już na poziomie ramek MAC, opisywane w rozdziale 3. Filtrowanie to rozpoczyna się na etapie karty sieciowej (NIC) przełącznika lub mostka, która może otrzymaną ramkę odrzucić albo przekazać do dalszego przetwarzania. Ideę tę przedstawiono na rysunku 9.5.

W typowych sieciach opartych na „przełączanym" Ethernecie ramki związane z broadcastingiem i multicastingiem replikowane są na wszystkie segmenty sieci VLAN, zgodnie z drzewem rozpinającym wyznaczonym przez przełączniki. Karta sieciowa w każdym z hostów po otrzymaniu takiej ramki i pomyślnym zweryfikowaniu jej integralności (za pomocą kodu CRC) podejmuje decyzję bądź to o jej zaakceptowaniu i przekazaniu do przetworzenia przez oprogramowanie sterownika, bądź też o natychmiastowym jej odrzuceniu. Zwykle karta sieciowa akceptuje tylko te ramki, w których adres docelowy jest tożsamy z adresem sprzętowym interfejsu lub jest adresem broadcast, gdy jednak w grę wchodzi multicasting, sprawy stają się bardziej skomplikowane — decyzja o przyjęciu albo odrzuceniu ramki musi opierać się na przesłankach nieco bardziej zaawansowanych.

Z perspektywy przeprowadzanej weryfikacji akceptowalności adresów MAC dostępne karty sieciowe podzielić można z grubsza na dwie grupy. W pierwszej plasują się karty przeprowadzające wspomnianą weryfikację w oparciu o *haszowanie*, realizowane na



Rysunek 9.5. *Każda warstwa stosu protokołów TCP/IP odpowiedzialna jest za pewną część filtrowania otrzymywanych komunikatów. Filtrowanie na poziomie adresów MAC może być wykonywane przez oprogramowanie sterownika karty sieciowej bądź bezpośrednio przez jej komponenty sprzętowe; tanie karty sieciowe o oszczędnym profilu sprzętowym powierzają oprogramowaniu większość funkcji związanych z filtrowaniem ramek*

bazie tablicy haszowanej, będącej w istocie ciągiem bitów ponumerowanych od 0 do N – 1 oraz funkcji haszującej $H()$, odwzorowującej adresy MAC na zbiór liczb całkowitych z zakresu od 0 do N – 1. Zasada jest prosta — ramka o adresie docelowym M zostaje zaakceptowana tylko wtedy, gdy bit o numerze $H(M)$ we wspomnianym ciągu ma wartość 1. Rozwiązanie to jest bardzo efektywne z perspektywy zapotrzebowania na pamięć, ponieważ haszowanie z definicji jest odwzorowaniem wieloznacznym — dla ustalonej wartości Y istnieje wiele takich wartości X, że $H(X) = Y$ — filtrowanie oparte na haszowaniu nie może być więc filtrowaniem precyzyjnym. Ustawiając zatem na 1 bit o numerze $H(M)$ w celu akceptowania M jako adresu docelowego, nieuchronnie powodujemy akceptowanie również każdego adresu M', który na akceptację nie zasługuje, lecz spełnia warunek $H(M') = H(M)$. Żadna to jednak katastrofa, po to wszak istnieją następne etapy filtrowania, by niwelować skutki zbytniej tolerancji na etapach poprzednich; gdyby filtrowanie na którymś z etapów realizowane było zbyt rygorystycznie, następne etapy nie miałyby już szansy na skorygowanie błędów poprzednika.

Do drugiej ze wspomnianych grup zaliczają się karty sieciowe stosujące filtrowanie adresów MAC na bazie tablicy przeglądowej, której pozycje odpowiadają wprost akceptowanym adresom MAC. Tablica taka ma jednak zwykle ustalony rozmiar, który może okazać się zbyt mały w stosunku do potrzeb i wówczas jedynym wyjściem jest przełączenie karty sieciowej w tryb nasłuchiwania multicastingowego (*multicast-promiscuous mode*), w którym *wszystkie* adresy multicast MAC traktowane są jako akceptowalne.

Zauważmy jednocześnie, że nawet gdy weryfikowanie akceptowalności adresów MAC odbywa się z perfekcyjną dokładnością, to i tak niezbędna jest dodatkowa weryfikacja otrzymywanych pakietów w warstwach wyższych z innego względu — niejednoznaczności mapowania adresów IP na adresy MAC, wyjaśnionej w punkcie 9.3.1.

Niezależnie jednak od opisanych komplikacji, multicasting wciąż stanowi wyraźnie atrakcyjną alternatywę dla broadcastingu.

Konkretna realizacja jednego z opisanych wariantów (lub obu) różna jest u różnych producentów kart sieciowych. Przykładowo kontroler ethernetowy 82583V firmy Intel oferuje 16-pozycyjną tablicę przeglądową, 4096-bitową tablicę haszowaną oraz obsługę trybu *promiscuous* dla adresów MAC multicast lub adresów MAC generalnie.

Ramka, po formalnym zaakceptowaniu przez sprzęt karty sieciowej (pod względem integralności potwierdzonej przez kod CRC, zgodność znaczników VLAN i obecność adresu docelowego MAC w tablicach NIC) zostaje przekazana do sterownika tejże karty, gdzie realizowany jest kolejny etap filtrowania. Na tym etapie należy przede wszystkim określić protokół warstwy sieciowej, dla którego przeznaczona jest ramka (IPv4, IPv6, ARP itd.) — nierozpoznanie właściwego protokołu powoduje odrzucenie ramki.

Gdy wydobyty z ramki pakiet okaże się być datagramem IPv4 lub IPv6, następuje określenie właściwego protokołu warstwy transportowej (najczęściej TCP lub UDP), którego jednostkę enkapsuluje wspomniany datagram. Na poziomie protokołu UDP następuje weryfikacja poprawności numeru portu — jeśli z portem tym nie jest aktualnie skojarzony uruchomiony proces, datagram UDP zostaje odrzucony, a do nadawcy wysłany zostaje komunikat ICMPv4 lub ICMPv6 *Port Unreachable* (analogiczna weryfikacja numeru portu dokonywana jest także przez protokół TCP). Po pomyślnym zweryfikowaniu poprawności numeru portu następuje weryfikacja integralności datagramu UDP — niepoprawna wartość sumy kontrolnej powoduje odrzucenie datagramu bez ostrzeżenia.

Podstawową motywacją kryjącą się za projektem multicastingu było uniknięcie zbędnego ruchu nieuchronnie generowanego w ramach broadcastingu. Rozważmy w charakterze przykładu sieć (LAN lub VLAN) złożoną z 50 hostów i działającą w tej sieci aplikację rozproszoną obejmującą 20 hostów. Gdyby hosty te komunikowały się za pomocą broadcastingu, oznaczałoby to niepotrzebne zawracanie głowy (procesora?) pozostałym 30 hostom, każdy z nich zmuszony byłby do bezcelowego przyjmowania ramek, które poddane zostałyby rutynowemu przetwarzaniu, i rzeczona bezcelowość okazałaby się ewidentna dopiero na etapie protokołu UDP (po stwierdzeniu, że numer portu nie odpowiada żadnemu uruchomionemu procesowi). Dzięki multicastingowi zainteresowane hosty formują grupę ukierunkowaną na odbiór datagramów związanych z aplikacją; karty sieciowe wszystkich hostów uzyskują informację o przynależności każdego z nich do określonych grup multicast, wskutek czego ramki wysyłane w ramach wspomnianej aplikacji, kiedy trafiają do niezainteresowanych nimi hostów, odfiltrowywane są już na poziomie karty sieciowej. Przyczynia się to do znacznego odciążenia hostów od zbędnego przetwarzania, za cenę jednakże dodatkowej złożoności związanej z zarządzaniem adresami grupowymi i przynależnością konkretnych hostów do poszczególnych grup multicast.

9.4. Protokoły IGMP i MLD

Dotychczas opisywaliśmy transmisję, filtrowanie i odbieranie datagramów multicast z punktu widzenia konkretnego hosta. Multicasting wiąże się jednak często z wędrówką datagramu przez rozległą sieć lub wiele podsieci, co rodzi zagadnienie **trasowania multicast**, wykonywanego przez jeden lub więcej routerów. To komplikuje koncepcję multicastingu jeszcze bardziej, bo routery multicast wymagają zarówno informacji o zdefinio-

wanych grupach multicast, jak i zbioru hostów tworzących każdą z tych grup. Routery te realizują ponadto procedurę **forwardowania z uwzględnieniem ścieżki odwrotnej** (*Reverse Path Forwarding*, w skrócie RPF). Procedura ta ma na celu zabezpieczenie przed powstawaniem zapętlonych tras, jej podstawą jest kontrola trasy przybywającego datagramu multicast: datagram odebrany na danym interfejsie akceptowany jest tylko wtedy, jeśli na poprzednim przeskoku wysłany został przez interfejs sprzężony z bieżącym.

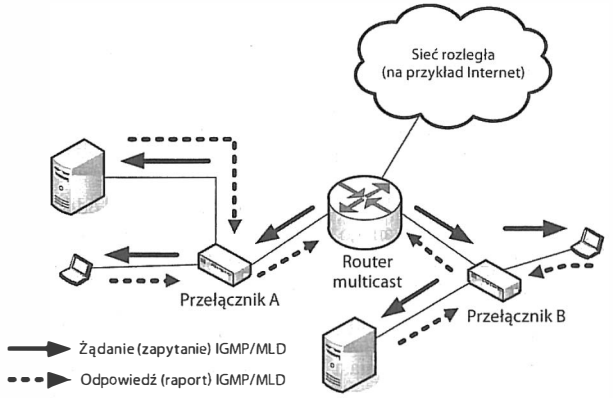
Trasowanie ruchu multicast oddzielone jest w dużym stopniu od konwencjonalnego trasowania ruchu unicast wykonywanego przez routery IP, choć w IPv6 protokół ND, niezbędny do trasowania unicast (patrz rozdział 8.), korzysta również z mechanizmów trasowania multicastingowego. Do sprawowania kontroli nad przynależnością poszczególnych hostów do konkretnych grup multicast routery wykorzystują głównie dwa protokoły: w IPv4 jest to protokół IGMP (*Internet Group Management Protocol* — internetowy protokół zarządzania grupami), w IPv6 natomiast funkcje tę pełni protokół MLD (*Multicast Listener Discovery* — odnajdywanie obserwatorów multicastingu). Protokoły te są podobne, a routery wykorzystują je do wyboru odpowiednich interfejsów dla forwardowania poszczególnych datagramów multicast. W większości przypadków dla routerów multicast nie jest istotna liczba uczestników danej grupy, lecz sam fakt jej istnienia — czyli istnienie *co najmniej jednego hosta*, osiągalnego za pośrednictwem określonego interfejsu, który to host członkostwo we wspomnianej grupie anonsuje lub potwierdza na żądanie. Dzięki mechanizmom adresowania multicast w warstwie łącza danych router nie musi zajmować się rozsyłaniem datagramów do każdego członka grupy z osobna, wystarczające jest wysłanie jednej kopii na określony adres grupy.

Protokół IGMP przeszedł kilka przeobrażeń; obecnie najczęściej używana jest wersja 3., opisywana w dokumencie [RFC3376]. Równolegle dokonywała się także ewolucja protokołu MLD, jego wersja 2. definiowana jest w dokumencie [RFC3810]. Obie wymienione wersje — IGMPv3 w IPv4 oraz MLDv2 w IPv6 — niezbędne są do realizacji SSM; w dokumencie [RFC4604] opisywane są szczegóły ich użycia w sytuacji, gdy grupa multicast toleruje wyłącznie datagramy wysyłane do niej z konkretnego, pojedynczego źródła.

Wersja IGMPv1 była pierwszą powszechnie używaną, w wersji IGMPv2 dodano mechanizm szybszego opuszczania grupy multicast przez host (mechanizm ten istniał również w MLDv1). W wersjach IGMPv3 i MLDv2 pojawiły się wspomniane przed chwilą mechanizmy selekcji źródeł nadawców datagramów multicast, niezbędne do realizacji SSM. Podczas gdy IGMP jest oddzielnym protokołem używanym w połączeniu z IPv4, MLD jest w rzeczywistości częścią protokołu ICMPv6 (patrz rozdział 8.).

Na rysunku 9.6 widoczny jest schemat funkcjonowania protokołu IGMP (MLD) w kontekście routera realizującego multicasting IPv4 (IPv6). Dla routera tego najistotniejszą informacją jest związek jego poszczególnych interfejsów z poszczególnymi grupami multicast — informacja ta pozwala unikać niepotrzebnego broadcastingu pakietów przez wszystkie interfejsy.

Protokół IGMP rozsyła swe żądania na adres grupowy wszystkich hostów (*All Hosts* — `224.0.0.1`), protokół MLD — na lokalny dla łącza adres grupowy wszystkich węzłów (*link-local All Nodes* — `ff02::1`). Komunikaty te przetwarzane są przez wszystkie hosty implementujące multicasting IP (z wyjątkiem opisanym w punkcie 9.4.2), które to hosty odsyłają w odpowiedzi raporty dotyczące swego członkostwa w grupach multicast. Raporty



Rysunek 9.6. *Routery multicast periodycznie wysyłają żądania IGMP (MLD) do każdej przyłączonej podsieci w celu uzyskania informacji na temat istniejących grup multicast. W odpowiedzi na takie żądanie każdy host należący do którejkolwiek grupy (grup) odsyła raport wskazujący tę grupę (grupy) wraz z informacjami o potencjalnych nadawcach, od których życzy sobie otrzymywać komunikaty multicast. Hosty mogą także wysyłać raporty z własnej inicjatywy, w sytuacjach gdy zmienia się ich status uczestnictwa w grupach multicast*

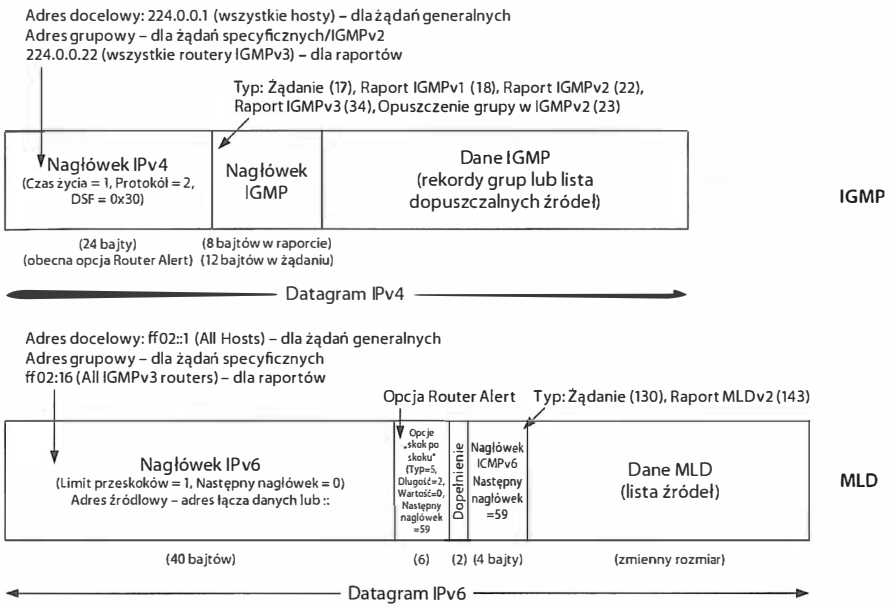
takie mogą być wysyłane także z inicjatywy samych hostów, gdy zmieniają swój status członkowski bądź modyfikują listę tolerowanych nadawców komunikatów multicast. Takie „spontaniczne" raporty wysyłane są na adres IP reprezentujący wszystkie routery multicast, w IGMPv3 jest to adres `224.0.0.22`, a w MLDv2 — adres `ff02::16`. Zauważmy przy okazji, że routery multicast, oprócz realizacji opisanego przetwarzania, także mogą uzyskiwać członkostwo w grupach multicast.

W wersjach IGMPv1 i IGMPv2 host po otrzymaniu żądania nie udziela odpowiedzi natychmiast, lecz odczekuje losowy interwał czasu w nadziei, iż w międzyczasie wyręczy go w tym inny host tej samej grupy. Jak wcześniej wyjaśnialiśmy, dla pytającego routera obojętna jest informacja na temat liczebności grupy — jemu chodzi tylko o związki istniejących grup z jego poszczególnymi interfejsami i dla każdej grupy multicast rozstrzygnięcie dokonuje się na zasadzie dylematu „jest odpowiedź" albo „brak odpowiedzi". Ten użyteczny mechanizm, notabene przyczyniający się do redukcji natężenia ruchu w sieci, został jednak zarzucony w wersji IGMPv3, z powodów wyjaśnionych w szczegółowo w punkcie 2. dodatku A do dokumentu [RFC3376]. Ujmując rzecz skrótowo, chodzi tu m.in. o

- zapewnienie routerowi możliwości szczegółowego (na poziomie konkretnych hostów) śledzenia statusu członkostwa w grupach związanych z poszczególnymi interfejsami, co może być pomocne w implementacji szybkiego opuszczania grupy przez host (*fast leave*) oraz jest niezbędne dla zbierania informacji statystycznych, np. na potrzeby rozliczeń;
- komplikację implementacji „podsłuchiwania" IGMP (o którym piszemy w punkcie 9.4.7), wynikającą z konieczności zatrzymania forwardowania raportów IGMP między segmentami sieci LAN;
- uproszczenie maszyny stanowej hosta — jeżeli jest on członkiem przynajmniej jednej grupy multicast, otrzymanie żądania IGMP skutkuje wysłaniem stosownego raportu, bez względu na kontekst innych hostów w grupie;
- małą użyteczność całego mechanizmu w sytuacji, gdy wysyłany raport dotyczy członkostwa w wielu grupach multicast.

> Specyfikacje IGMPv3 oraz MLDv2 wymagają jednocześnie zgodności ze swymi poprzednimi wersjami w celu zapewnienia współpracy ze starszymi hostami i routerami rezydującymi w tej samej podsieci.

Na rysunku 9.7 pokazano schemat enkapsulacji komunikatów IGMP i MLD. Podobnie jak ICMP, protokół IGMP może być uważany za część warstwy IP — jego komunikaty transmitowane są w otoczce datagramów IPv4. W odróżnieniu od innych omawianych dotąd protokołów, pole *Czas życia* w nagłówku tych datagramów otrzymuje wartość 1, co uniemożliwia przedostawanie się tychże datagramów poza lokalną podsieć. W nagłówku tych datagramów obecna jest także opcja *Router Alert*, a w polu *DSF* znajduje się wartość 0x30 oznaczająca (zgodnie z tabelą 5.1) priorytet na poziomie sterowania międzysiecią (CS6).



Rysunek 9.7. *Pakiety IGMP, jako niezależnego protokołu, enkapsulowane są w datagramach IPv4; komunikat MLD jest odmianą komunikatu ICMPv6. Wartość 0 w polu Następny nagłówek nagłówka podstawowego IPv6 identyfikuje nagłówek rozszerzeń Opcje „skok po skoku"*

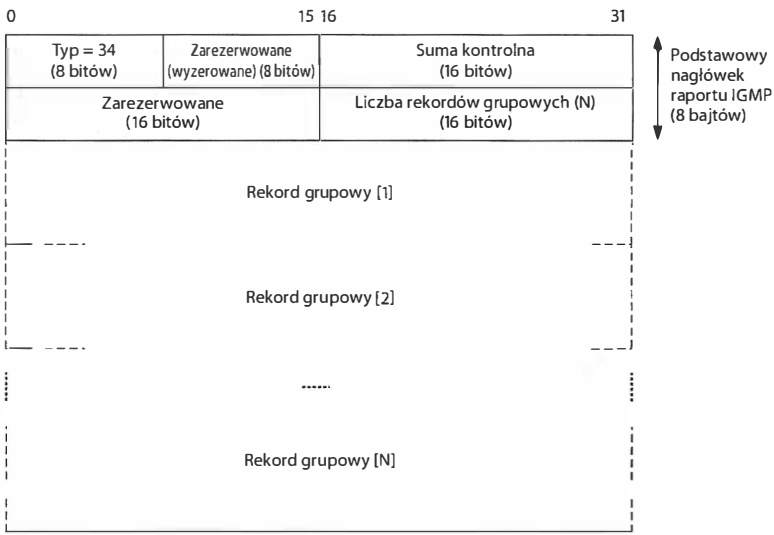
Protokół MLD nie jest niezależnym protokołem, lecz częścią protokołu ICMPv6. Jego komunikaty enkapsulowane są w datagramach IPv6 zawierających opcję *Router Alert* w nagłówku rozszerzeń opcji *Skok po skoku*. Strukturę tych komunikatów omówiliśmy pokrótce w punktach 8.4.6 i 8.4.7. Ponieważ funkcjonalnie protokoły IGMP i MLD są niemal identycznie, opiszemy ich funkcjonalność łącznie.

Protokoły IGMP i MLD definiują dwa zbiory reguł przetwarzania — dla hostów będących uczestnikami grup multicast (lub pretendujących do takiego uczestnictwa) oraz dla routerów multicast. Mówiąc ogólnie, zadania hostów uczestniczących w grupach multicast są dwa: pierwszym jest spontaniczne wysyłanie raportów powiadamiających o zmianach w szcze-

gółach tego uczestnictwa i (lub) zmianach w zestawie tolerowanych źródeł, drugie natomiast polega na generowaniu tychże raportów w odpowiedzi na żądania (zapytania) routerów. Routery multicast zobowiązane są do monitorowania statusu konkretnych grup multicast na poszczególnych łączach, mają też możliwość badania statusu konkretnej grupy, wraz ze zbiorem dopuszczalnych źródeł określonym przez tę grupę. Routery te muszą ponadto współdziałać z protokołami multicastingu dla sieci rozległych, m.in. protokołami PIM-SM i BIDIR-SM — nie będziemy ich tu omawiać, zainteresowanym czytelnikom polecamy lekturę dokumentów [RFC4601] i [RFC5015].

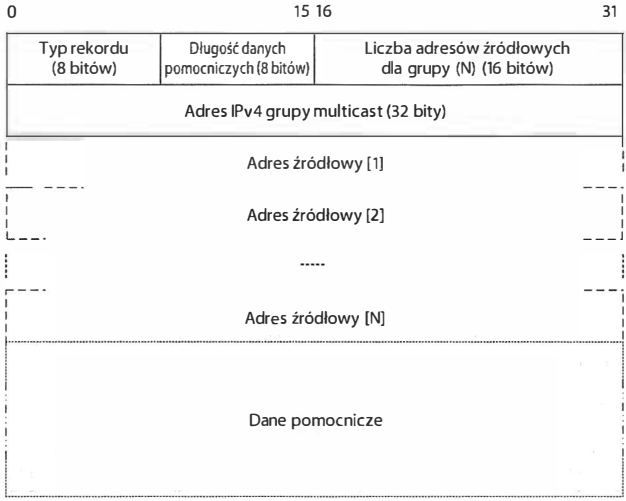
9.4.1. Przetwarzanie komunikatów IGMP i MLD przez hosty

W części dotyczącej hostów członkowskich protokoły IGMP i MLD umożliwiają tym hostom przystępowanie do grup multicast i ich opuszczanie oraz specyfikowanie źródeł, z których (w ramach poszczególnych grup) hosty te życzą sobie (lub nie życzą) otrzymywać komunikaty multicast. Rzeczone hosty sygnalizują swe zamiary przez wysyłanie raportów do jednego lub kilku routerów przyłączonych do tej samej podsieci (oraz do innych hostów partycypujących w tej samej grupie). Raporty te mogą być wysyłane bądź to w odpowiedzi na periodycznie napływające zapytania routerów, bądź też z inicjatywy samych hostów, w reakcji na lokalne zmiany statusu, czyli zgłaszanie i odwoływanie akcesu do poszczególnych grup przez procesy działające w tych hostach. Format raportu IGMP przedstawiony jest na rysunku 9.8.



Rysunek 9.8. *Raport IGMPv3 odzwierciedlający uczestnictwo hosta w grupie (grupach) multicast zawiera ciąg rekordów grupowych, po jednym dla każdej grupy. Rekord grupowy specyfikuje adres multicast odnośnej grupy oraz listę tolerowanych przez tę grupę nadawców komunikatów multicast*

Zasadniczą część raportu IGMP stanowi wektor *rekordów grupowych*, z których każdy specyfikuje adres jednej grupy multicast oraz (opcjonalnie) zestaw *filtrów*, określających szczegółowo zbiór tolerowanych nadawców komunikatów multicast dla tej grupy. Strukturę pojedynczego rekordu grupowego przedstawiamy na rysunku 9.9.



Rysunek 9.9. *Rekord grupowy IGMPv3 zawiera adres multicast grupy oraz listę adresów źródłowych tolerowanych albo nietolerowanych (zależnie od Typu rekordu) w roli nadawców komunikatów multicast dla tej grupy. W wersji IGMPv1 i IGMPv2 rekordy grupowe nie zawierały listy adresów źródłowych*

Oczywistym elementem rekordu grupowego jest adres odnośnej grupy multicast, w IGMP 32-bitowy, w MLD 128-bitowy. *Długość danych pomocniczych* wyrażona jest w 32-bitowych słowach; protokół IGMPv3 nie przypisuje obecnie żadnego znaczenia *Danym pomocniczym*, rekord kończy się więc na ostatnim adresie źródłowym, a wspomniane pole zawiera wartość 0. Lista *Adresów źródłowych* może być traktowana na dwa sposoby, zależnie od wartości pola *Typ rekordu*: w *trybie akceptującym* (*include mode*) obecne na liście adresy traktowane są jako *dozwolone* źródła komunikatów multicast dla grupy, natomiast w *trybie eliminującym* (*exclude mode*) obecne na liście adresy traktowane są jako zabronione — każdy adres niefigurujący na tej liście jest adresem dozwolonym. Znaczenie i kontekst użycia poszczególnych wartości *Typu rekordu* wyjaśnione są w tabeli 9.1; wartości te są identyczne dla IGMPv3 i MLD. W tej konwencji zamiar opuszczenia grupy przez host sygnalizowany jest za pomocą rekordu grupowego z trybem akceptującym i pustą listą adresów źródłowych, podobnie rekord grupowy w trybie eliminującym, z pustą listą adresów źródłowych traktowany jest jako przystąpienie hosta do grupy. Zauważmy, że w przypadku SSM, gdy dozwolone jest tylko jedno źródło komunikatów multicast dla danej grupy, typy 0x02 i 0x04 rekordu grupowego nie mają zastosowania.

Rekordy dwóch pierwszych typów (0x01 i 0x02) określane są nazwą **rekordów bieżącego stanu** (*current-state records*) i wykorzystywane są do raportowania bieżącego stanu akceptowania albo ignorowania poszczególnych źródeł przez grupę. Dwa kolejne typy (0x03 i 0x04) identyfikują **rekordy przełączania trybu filtrowania** (*filter-mode-change records*) — z eliminującego na akceptujący i odwrotnie, zaś dwa ostatnie (0x05 i 0x06) to typy związane z **rekordami zmiany listy źródeł** (*source-list-change records*). Cztery ostatnie tryby (od 0x03 do 0x06) określane są również bardziej ogólnym mianem **rekordów zmiany stanu** (*state-change records*) lub **raportów zmiany stanu** (*state-change reports*), bo wysyłane są przez host w związku ze zmianą lokalnego stanu wynikającego z uruchamiania i (lub) kończenia procesów działających w hoście. Należy jednocześnie zaznaczyć, że same żądania i raporty IGMP/MLD zawsze są akceptowane przez każdą grupę.

Tabela 9.1. *Typy rekordów grupowych IGMPv3 i MLD określające tryb filtrowania i kontekst jego zmiany*

Typ	Nazwa i znaczenie	Rekord w raporcie wysyłanym w reakcji na...
0x01	MODE_IS_INCLUDE (IS_IN): komunikaty wysyłane przez dowolne źródło z listy mają być akceptowane przez grupę	... żądanie routera multicast
0x02	MODE_IS_EXCLUDE (IS_EX): komunikaty wysyłane przez dowolne źródło z listy mają być ignorowane przez grupę	
0x03	CHANGE_TO_INCLUDE_MODE (TO_IN): zmiana trybu eliminującego na akceptujący — komunikaty wysyłane przez dowolne źródło z listy mają być odtąd akceptowane przez grupę	... akcję lokalnego procesu wymagającą zmiany trybu filtrowania
0x04	CHANGE_TO_EXCLUDE_MODE (TO_EX): zmiana trybu akceptującego na eliminujący — komunikaty wysyłane przez dowolne źródło z listy mają być odtąd ignorowane przez grupę	
0x05	ALLOW_NEW_SOURCES (ALLOW): zmiana listy adresów źródłowych — komunikaty wysyłane przez dowolne źródło z nowej listy mają być odtąd akceptowane przez grupę	... akcję lokalnego procesu wymagającą zmiany listy akceptowanych źródeł
0x06	BLOCK_OLD_SOURCES (BLOCK): zmiana listy adresów źródłowych — komunikaty wysyłane przez dowolne źródło z listy mają być odtąd ignorowane przez grupę	... akcję lokalnego procesu wymagającą zmiany listy ignorowanych źródeł

Format raportu w protokole MLD jest podobny, z dwiema różnicami: *Typ* komunikatu równy jest 143 (patrz rozdział 8.), a adres multicast w rekordzie grupowym jest 128-bitowy.

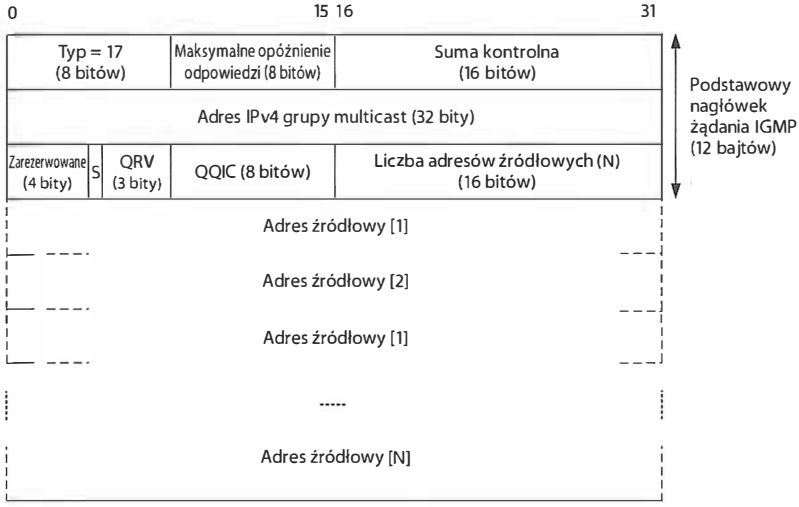
Host po odebraniu żądania IGMP/MLD nie wysyła raportu natychmiast, lecz odczekuje przez pewien (losowy) interwał czasu; dzięki temu wszystko, co w międzyczasie wydarzy się w kontekście multicastingu, host może ująć w formie pojedynczego raportu zamiast kilku raportów cząstkowych. Optymalizuje to zarówno obciążenie sieci (mniejszy ruch), jak i wykorzystanie mocy obliczeniowej routerów (mniej nagłówków do przetworzenia). Przypomnijmy jednocześnie, że począwszy od IGMPv3, odpowiedź hosta na żądanie IGMP jest obowiązkowa — nie zwalnia z tego obowiązku udzielenie odpowiedz przez inny host z tej samej grupy, a więc inercja hosta ma tu cel inny niż w poprzednich wersjach IGMP.

Adresem źródłowym w komunikacie IGMP jest główny (lub preferowany) adres interfejsu, przez który komunikat ten jest wysyłany. W komunikacie MLD adresem źródłowym jest adres IPv6 lokalny dla łącza. Wyjątkiem od tej zasady jest sytuacja, gdy bootujący właśnie host IPv6 realizuje dopiero proces uzyskiwania własnego adresu — elementem tego procesu jest procedura DAD, wykorzystująca komunikaty MLD (patrz rozdział 6.). W tej sytuacji, opisanej w dokumencie [RFC3590], we wspomnianych komunikatach MLD rolę adresu źródłowego może pełnić adres nieokreślony (::).

9.4.2. Funkcjonowanie routerów multicast

W zakresie dotyczącym routerów multicast zadaniem protokołów IGMP i MLD jest monitorowanie obecności grup multicast na poszczególnych łączach routera, a także monitorowanie istnienia poszczególnych grup (grupa multicast przestaje istnieć, gdy

opuszcza ją ostatni uczestnik). Wspomniane protokoły wykonują to zadanie przez periodyczne rozsyłanie żądań wśród hostów i wykorzystywanie otrzymywanych w odpowiedzi raportów do budowania informacji o stanie multicastingu. Informacja ta jest przykładem **miękkiego stanu** (*soft state*) — nieodświeżona w założonym interwale czasowym ulega skasowaniu. Format żądania IGMPv3 widoczny jest na rysunku 9.10.



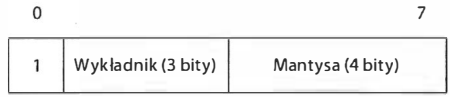
Rysunek 9.10. *Żądanie IGMPv3 zawiera adres grupy multicast i opcjonalnie listę adresów źródłowych. Żądania generalne specyfikują zerowy adres multicast i wysyłane są na adres grupowy All Hosts (224.0.0.1). W polu QRV znajduje się maksymalna dopuszczalna liczba retransmisji, dozwolona dla nadawcy, pole QQIC określa natomiast odstęp czasowy między kolejno wysyłanymi żądaniami. Żądania specyficzne, dotyczące konkretnego adresu multicast, używane są przez routery do potwierdzenia zamknięcia grupy (czyli opuszczenia jej przez ostatniego uczestnika) oraz do weryfikacji zmian na liście dopuszczalnych adresów źródłowych; we wszystkich wersjach IGMP żądania te wysyłane są na adres przedmiotowej grupy*

Od razu daje się zauważyć podobieństwo tego formatu do formatu komunikatu MLDv2, przedstawionego na rysunku 8.25. Pole *Maksymalne opóźnienie odpowiedzi* określa dopuszczalną dla hosta zwłokę w odesłaniu raportu, liczoną w jednostkach 100 milisekund; wartość w polu *QQIC* oznacza liczbę sekund dzielących kolejno wysyłane żądania. Oba pola kodowane są w konwencji omówionej w punkcie 8.4.7. Dla przypomnienia — wartości nieprzekraczające 127 reprezentowane są w postaci 7-bitowych liczb całkowitych, większe wartości przedstawiane są natomiast w 8-bitowej reprezentacji zmiennopozycyjnej, wyjaśnionej na rysunku 8.27 i dla ułatwienia powtórzonej na rysunku 9.11. Wynikowa wartość obliczana jest ze wzoru

$$(mantysa + 16) \cdot 2^{(wykładnik + 3)}$$

co daje maksymalną wartość $(15 + 16) * 2^{10} = 31\,744$.

Dobór odpowiedniej wartości *Maksymalnego opóźnienia odpowiedzi* wymaga rozstrzygnięcia nieuchronnego kompromisu: niewielka wartość tego opóźnienia zwiększa czułość protokołu, czyli zapewnia szybsze rozpoczęcie zatrzymywania komunikatów multicast



Maksymalne opóźnienie odpowiedzi = 100 ms • (mantysa + 16) • $2^{(wykładnik+3)}$

Rysunek 9.11. *Kodowanie maksymalnego opóźnienia odpowiedzi w jednostkach 100 milisekund. Wartości nie większe niż 127 kodowane są jako liczby całkowite, pozostałe wartości kodowane są w 8-bitowej reprezentacji zmiennopozycyjnej*

płynących do nieistniejącej już grupy; duża wartość opóźnienia redukuje natężenie komunikatów protokołu wymienianych między hostami a routerami, lecz stwarza ryzyko dłuższego okresu niepotrzebnego przetwarzania ruchu płynącego donikąd.

Znaczenie pozostałych pól jest identyczne ze znaczeniem pól w żądaniu MLD, omawianym w punkcie 8.4.7: pola *S* i *QRV* związane są z odpornością protokołu na awarię — omówimy je szczegółowo w punkcie 9.4.5. *Suma kontrolna* tworzona jest zgodnie z typowym algorytmem opisanym w punkcie 5.2.2.

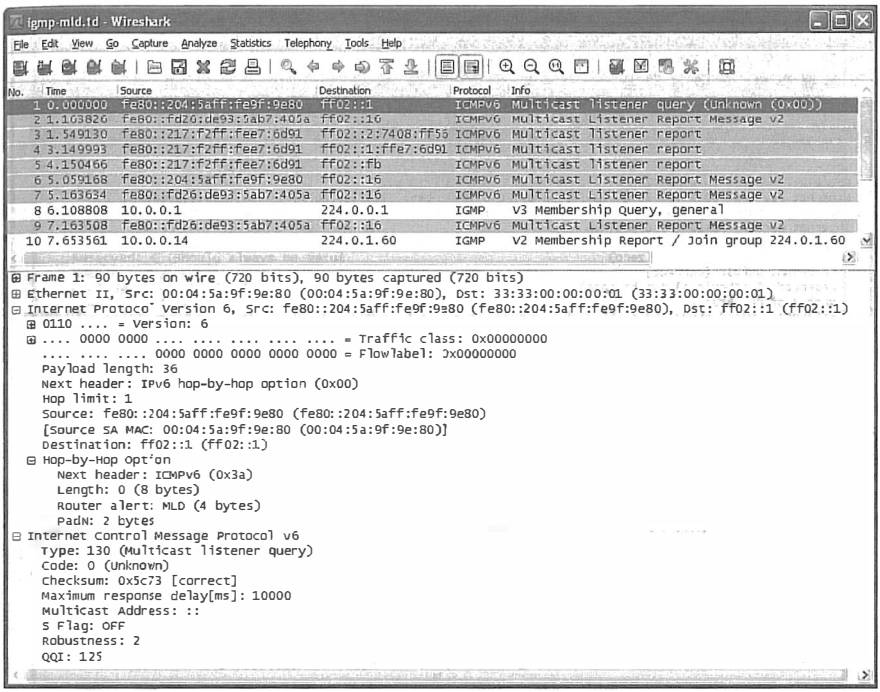
Żądania wysyłane przez routery multicast występują w trzech odmianach. Pierwsza z nich — *generalna* — ma na celu ogólne rozeznanie istniejących grup multicast; w komunikatach tego rodzaju *Adres grupy multicast* jest adresem zerowym, komunikaty te wysyłane są na adres grupowy wszystkich węzłów (`224.0.0.1` w IPv4 i `ff02::1` w IPv6). Żądanie *ukierunkowane*, zwane także *specyficznym dla grupy*, skierowane jest do konkretnej grupy multicast, której adres figuruje w polu *Adres grupy multicast* i jest jednocześnie adresem docelowym komunikatu. Żądanie *szczegółowe*, zwane także *specyficznym dla grupy i źródeł*, kierowane jest, podobnie jak żądanie ukierunkowane, do konkretnej grupy multicast, lecz ponadto zawiera wykaz konkretnych źródeł, dozwolonych lub zabronionych, dla tej grupy.

Żądania ukierunkowane i szczegółowe wysyłane są przez routery w celu ostatecznego upewnienia się przez nie o zasadności podejmowanych akcji — np. uznania, że określona grupa multicast przestała istnieć i kierowane na jej adres komunikaty można odtąd zwyczajnie ignorować. Ignorowanie takie może mieć również podłoże bardziej subtelne: jeżeli mianowicie określony host źródłowy znajduje się na liście zabronionych źródeł u każdego hosta danej grupy multicast, można ignorować komunikaty multicast wysyłane przez ów host do tejże grupy.

Z protokołami IGMP i MLD wiąże się konieczność pogodzenia jeszcze jednej pary sprzecznych okoliczności. Z jednej strony, komunikaty tych protokołów mają znaczenie krytyczne dla zapewnienia prawidłowości forwardowania pakietów, z drugiej jednak, są to protokoły bez gwarancji niezawodności, a więc wspomniane komunikaty mogą w sieci najzwyczajniej ginąć — przykładowo brak odpowiedzi na żądanie ukierunkowane może świadczyć o zakończeniu żywota przez grupę multicast, lecz równie dobrze może być konsekwencją zagubienia raportu wysłanego przez host (hosty). W celu zniwelowania skutków zawodności rzeczonych protokołów ich komunikaty są wielokrotnie *retransmitowane* — tym tematem zajmiemy się szczegółowo w punkcie 9.4.5.

9.4.3. Przykłady

Na rysunku 9.12 widzimy efekt śledzenia pakietów związanych z protokołami IGMPv2, IGMPv3, MLDvl i MLDv2 działającymi w tej samej podsieci. Śledzenie obejmuje 16 pakietów (na rysunku widoczne jest 10), z których pierwszy jest żądaniem pochodzącym z węzła (routera) identyfikowanego lokalnym dla łącza adresem `fe80::204:5aff:fe9f:9e80` (w obu wersjach MLD format żądania jest identyczny). Ten sam węzeł formułuje jednocześnie żądania protokołu IGMP i identyfikuje się adresem IPv4 `10.0.0.1`.

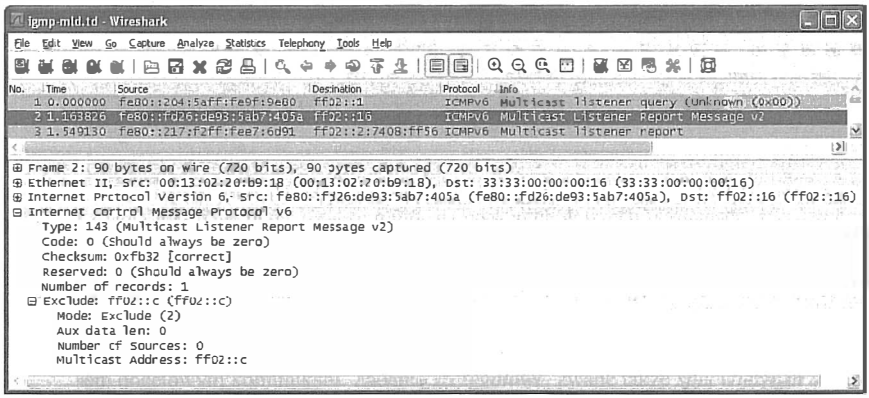


Rysunek 9.12. *Pakiety protokołów IGMPv2, IGMPv3, MLDv1 i MLDv2 działających w tej samej podsieci. Wyróżniony pakiet jest żądaniem MLD*

Wspomniane żądanie MLD (pakiet 1.) wysyłane jest na adres grupowy wszystkich węzłów (`ff02::1`); źródłowym adresem MAC jest `00:04:5a:9f:9e:80`, docelowym `33:33:00:00:00:01` — to ilustracja opisanej wcześniej zasady konwersji adresów IP na adresy MAC (dla przypomnienia — w IPv6 prefiks `33:33` rozpoczyna sprzętowe adresy multicast, tak jak na rysunku 9.3). W polu *Limit przeskoków* znajduje się wartość 1, bo komunikaty MLD nie powinny być forwardowane poza lokalną sieć. *Rozmiar ładunku użytecznego* wynosi 36 bajtów — składają się nań: 8-bitowa opcja *Router Alert*, 4-bajtowy nagłówek ICMPv6 i 24 bajty „właściwych" danych MLD. Pola *Typ*, *Kod*, *Suma kontrolna* i *Maksymalne opóźnienie odpowiedzi* zajmują łącznie 8 bajtów ze wspomnianych 24, na 16 kolejnych znajduje się przedmiotowy adres IPv6 multicast, w tym przypadku 0 (mamy do czynienia z żądaniem generalnym). Dwa kolejne bajty to znacznik *S* oraz pola

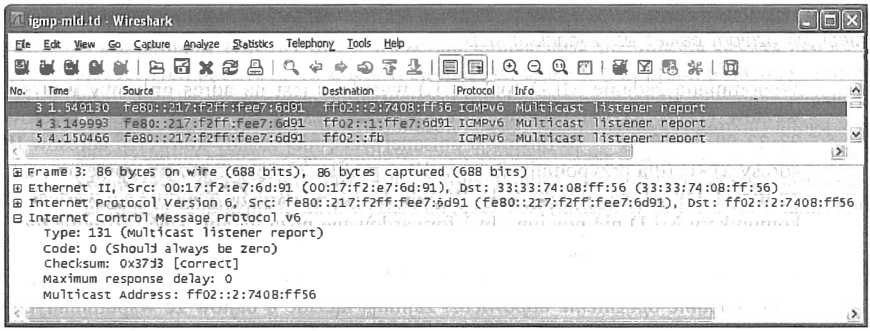
QRV i *QQIC*, w dwóch końcowych bajtach znajduje się licznik pozycji na liście adresów źródłowych, tym razem równy 0 (lista jest pusta). *Maksymalne opóźnienie odpowiedzi* określono na 10 sekund, wartość zmiennej *QRV* wynosi 2, a żądania rozsyłane są w odstępie *QQIC* równym 125 sekund — są to wartości domyślne w protokole MLD.

Drugi pakiet, uwidoczniony szczegółowo na rysunku 9.13, to raport MLDv2 wyrażający zainteresowanie hosta grupą multicast o adresie `ff02::c` — to lokalny dla łącza adres charakterystyczny dla protokołu SSDP. Zamiar dołączenia do grupy wyrażony jest przez pojedynczy rekord grupowy, zawierający pustą listę źródeł w trybie eliminującym.



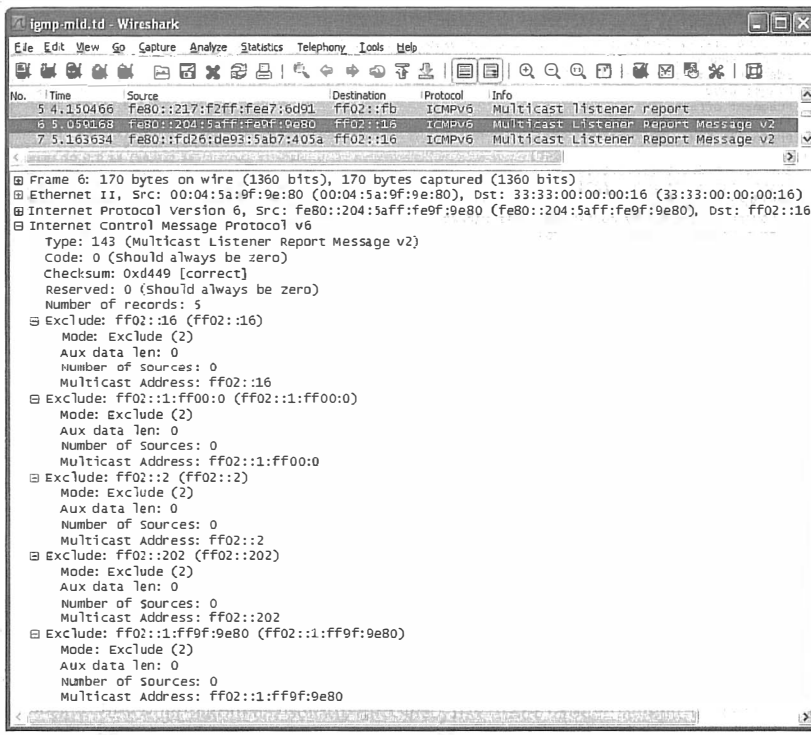
Rysunek 9.13. *Raport MLDv2 wyrażający zainteresowanie grupą multicast protokołu SSDP (identyfikowaną za pomocą adresu grupowego ff02::c). Rekord grupowy zawiera pustą listę źródeł w trybie eliminującym*

Trzy następne pakiety, o numerach 3, 4 i 5, to raporty protokołu MLDv1, wciąż — jak widać — używanego; na rysunku 9.14 uwidoczniono tylko pierwszy z wymienionych pakietów, pozostałe różnią się od niego wyłącznie adresami docelowymi IPv6. Podobnie jak w wersji MLDv2, każdy z raportów wykorzystuje tę samą strukturę nagłówków IPv6 (podstawowego i rozszerzających), lecz adresem docelowym jest adres odnośnej grupy multicast — `ff02::2:7408:ff56`; odpowiadającym mu adresem MAC jest `33:33:74:08:ff:56`.



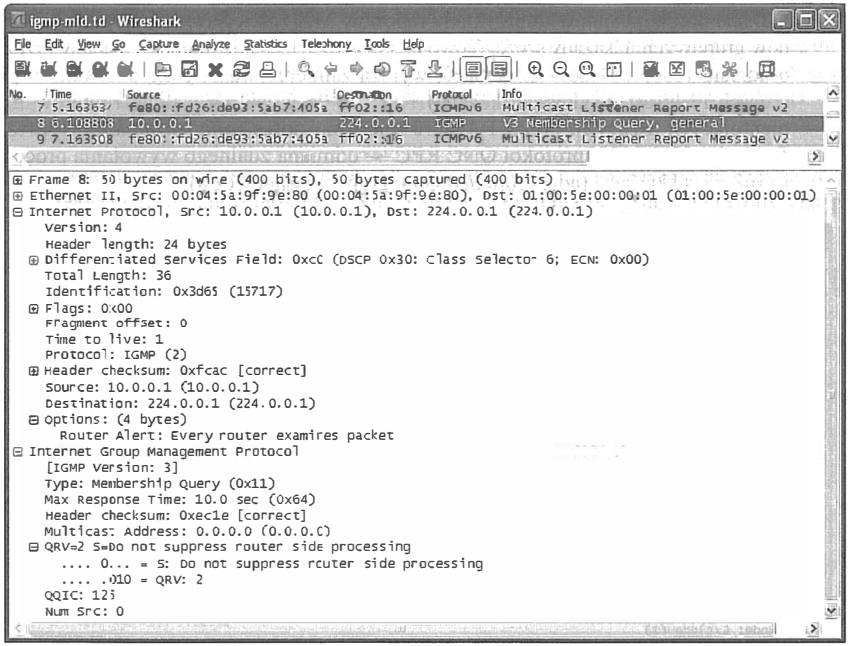
Rysunek 9.14. *Raport MLDv1 jako sygnał zainteresowania hosta grupą multicast o adresie ff02::2:7408:ff56, stanowiącym także adres docelowy*

Kolejna porcja pakietów to przykłady raportów MLDv2 wykorzystujących wiele rekordów grupowych. Ukazany szczegółowo na rysunku 9.15 pakiet numer 6, nadany przez host o adresie `fe80::204:5aff:fe9f:9e80`, wyraża zainteresowanie uczestnictwem w pięciu grupach multicast, o adresach (kolejno) `ff02::16` (to grupa wszystkich routerów implementujących MLDv2), `ff02::1:ff00:0` (pierwszy adres *solicited-node*), `ff02::2` (*All Routers*), `ff02::202` (protokół ONC RPC — odmiana zdalnego wywołania procedury) i `ff02::1:ff9f:9e80` (adres *solicited node* grupy własnej węzła).



Rysunek 9.15. *Raport MLDv2 wyrażający zainteresowanie uczestnictwem w pięciu grupach multicast. Każda z tych grup reprezentowana jest przez rekord grupowy z pustą listą źródeł w trybie eliminującym*

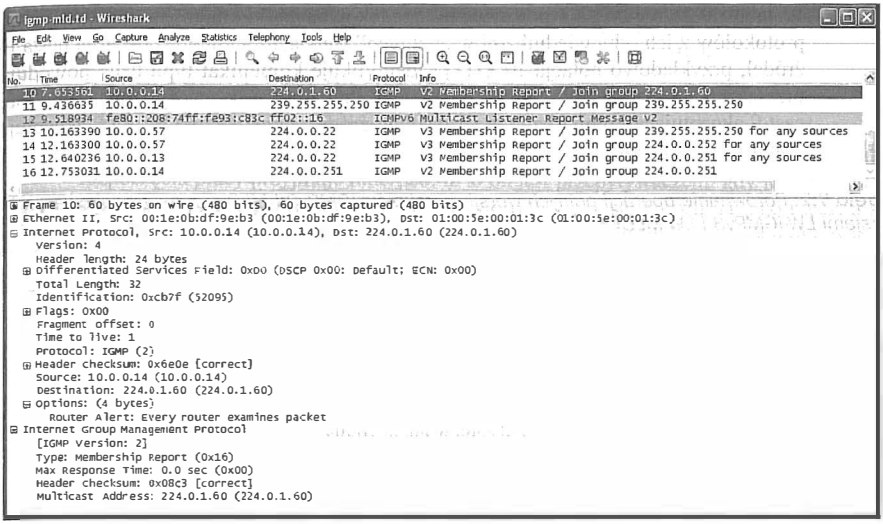
Kolejne pakiety pochodzą od protokołu IGMPv3. Widoczny na rysunku 9.16 pakiet numer 8 to żądanie generalne pochodzące od węzła `10.0.0.1`, wysyłane na adres grupowy `224.0.0.1` (*All Nodes*), odwzorowany na adres MAC `01:00:5e:00:00:01`. *Czas życia* (TTL) ustawiony jest na 1 — komunikaty IGMP nie są forwardowane przez routery. Nagłówek IPv4 ma rozmiar 24 bajty — do standardowych 20 bajtów dochodzi 4-bajtowa opcja *Router Alert*. Jak w każdym żądaniu generalnym, przedmiotowy adres multicast jest zerowy; maksymalne opóźnienie odpowiedzi oraz odstęp między kolejnymi żądaniami mają domyślne wartości (odpowiednio) 10 sekund i 125 sekund. Pakiet numer 9 — niewidoczny w szczegółach, lecz podobny do pakietów 7. i 2. — to raport MLDv2, wyrażający zainteresowanie grupą multicast o adresie `ff02::1:3`, charakterystycznym dla protokołu LLMNR.



Rysunek 9.16. *Generalne żądanie IGMPv3, wysyłane na adres grupowy 224.0.0.1 (All Nodes). W polu DSF nagłówka IPv4 znajduje się wartość DSCP 0x30, oznaczająca klasę sterowania międzysiecią (6), nagłówek ten zawiera także opcję Router Alert*

Siedem ostatnich pakietów objętych śledzeniem można zobaczyć na rysunku 9.17. Pakiet 10. to raport IGMPv2 wysyłany przez drukarkę sieciową o adresie `10.0.0.14` na adres grupowy `224.0.1.60`, przyporządkowany usłudze odnajdowania sprzętu w produktach firmy Hewlett-Packard. Podobnie jak w przypadku MLDv1, raporty IGMPv2 wysyłane są na adres docelowy tożsamy z adresem grupy multicast, której dotyczą. Wartość 1 w polu *Czas życia* (TTL) oznacza zakaz forwardowania raportu przez routery: do standardowych 20 bajtów nagłówka IPv4 dochodzą 4 bajty opcji *Router Alert* oraz 8 bajtów składających się na treść raportu IGMP.

Następne pakiety, nieukazane szczegółowo, podobne są do pakietów wcześniej prezentowanych. Pakiet 11. wysłany został przez wspomnianą drukarkę `10.0.0.14` jako raport wyrażający zainteresowanie grupą multicast `239.255.255.250` (związaną z UPnP). Pakiet 12. to raport MLDv2, za pośrednictwem którego host `fe80::208:74ff:fe93:c83c` wyraża zainteresowanie grupami `ff02::202` (ONC RPC) i `ff02::1:ff93:f83c` (i własny adres *solicited node* hosta). Pakiety 13. i 14. wysłane zostały przez host `10.0.0.57` w związku z zainteresowaniem grupami (odpowiednio) `239.255.255.250` i `224.0.0.252` (LLMNR). Dwa ostatnie pakiety to raporty (odpowiednio) IGMPv3 i IGMPv2 pochodzące od węzłów `10.0.0.13` i `10.0.0.14` zamierzających dołączyć do grupy `224.0.0.251` (mDNS, patrz rozdział 11.).



Rysunek 9.17. *Pakiet 10. oraz sześć kolejnych pakietów to mieszanka raportów IGMPv2 i IGMPv3 (z wyjątkiem pakietu 12.). Komunikaty IGMPv2 nie zawierają listy dopuszczalnych lub zabronionych źródeł*

9.4.4. Protokoły LW-IGMPv3 i LW-MLDv2

Routery multicast, na podstawie otrzymywanych raportów, budują aktualną informację o stanie grup multicast i ich relacji z określonymi źródłami nadawcami komunikatów multicast. Informacja ta jest niezbędna do prawidłowego zaopatrywania grup multicast w adresowane do nich komunikaty, jest także pomocna w różnego rodzaju optymalizacjach: przykładowo routery multicast mogą zaniechać forwardowania komunikatów multicast pochodzących ze źródła, które u każdego potencjalnego odbiorcy wykazywane jest jako źródło zabronione. Mimo iż teoretycznie optymalizacja taka dawać może wymierne korzyści, to jednak w praktyce — jak pokazuje wieloletnie doświadczenie — aplikacje nader rzadko korzystają z możliwości wyłączania konkretnych źródeł, skwapliwie stosują natomiast selektywne specyfikowanie źródeł akceptowanych, szczególnie w przypadku SSM. Ergo — rezygnacja z implementowania w routerach trybu eliminującego nie byłaby dla programistów jakoś szczególnie dotkliwa, natomiast uprościłaby znacznie oprogramowanie tychże routerów, ponieważ nie byłoby konieczności utrzymywania list źródeł zabronionych.

Na bazie tej przesłanki skonstruowane zostały zredukowane („lekkie" — *lightweight*) wersje opisywanych protokołów, nazwane *Lightweight IGMPv3* (w skrócie LW-IGMPv3) i *Lightweight MLDv2* (w skrócie LW-MLDv2), obie opisywane w dokumencie [RFC5790]. Redukcja polega tu na praktycznej rezygnacji z trybu eliminującego — jedyną dopuszczalną w tym trybie postacią listy źródeł jest lista pusta, co odpowiada klasycznemu przypadkowi dołączania do grupy (jak np. na rysunku 9.13). Zredukowane wersje protokołów posługują się dokładnie tymi samymi komunikatami, co ich pierwowzory, obie też umożliwiają realizację zarówno ASM, jak i SSM.

W tabeli 9.2 znajduje się porównanie „zredukowanych" komunikatów wymienionych protokołów z ich odpowiednikami w „pełnych" wersjach. Symbol {} oznacza pustą listę źródeł, przykładowo notacja TO_EX({}) identyfikuje komunikat typu 0x04 powodujący przełączenie w tryb eliminujący z pustą listą. Przez (*, G) oznaczyliśmy grupę multicast G akceptującą komunikaty ze wszystkich źródeł, przez (S, G) grupę G akceptującą komunikaty pochodzące ze źródła S.

Tabela 9.2. *Porównanie operacji pełnych wersji protokołów IGMPv3 i MLDv2 z ich zredukowanymi wersjami LW-IGMPv3 i LW-MLDv2*

Wersja pełna	Wersja zredukowana	Komunikat wysyłany jako
IS_EX({})	TO_EX({})	Raport wyrażający dołączenie do (*, G), w odpowiedzi na żądanie
IS_EX(S)	-	Raport, w odpowiedzi na żądanie, wyrażający dołączenie do G, z traktowaniem źródła S jako zabronionego
IS_IN(S)	ALLOW(S)	Raport, w odpowiedzi na żądanie, wyrażający dołączenie do G, z akceptowaniem źródła S
ALLOW(S)	ALLOW(S)	Raport spontaniczny o dołączeniu do grupy (S, G)
BLOCK(S)	BLOCK(S)	Raport spontaniczny o opuszczeniu grupy (S, G)
TO_IN(S)	TO_IN(S)	Zmiana trybu eliminującego na akceptujący z dołączeniem do (S, G)
TO_IN({})	TO_IN({})	Opuszczenie (*, G)
TO_EX(S)	-	Zmiana trybu akceptującego na eliminujący z opuszczeniem (S, G)
TO_EX({})	TO_EX({})	Dołączenie do (*, G)

Widoczny jest brak trybów wykorzystujących niepustą listę źródeł w trybie eliminującym oraz zastąpienie rekordów bieżącego stanu (IS_EX i IS_IN) rekordami zmiany stanu. Co prawda, od routerów kompatybilnych z LW-IGMPv3 i LW-MLDv2 wymaga się honorowania rekordów IS_EX i IS_IN, jednakże w pierwszym z nich ignorowana jest lista źródeł, drugi natomiast równoważny jest rekordowi ALLOW.

9.4.5. Niezawodność IGMP i MLD

Prawidłowość funkcjonowania protokołów IGMP i MLD — i generalnie multicastingu — może być zakłócana przez wiele czynników; protokoły te zdolne są radzić sobie z dwoma najbardziej prawdopodobnymi — gubieniem komunikatów i awariami routerów.

Zabezpieczeniem na wypadek awarii routera jest redundancja sprzętowa, czyli równoległe działanie kilku routerów multicast na tym samym łączu. Jak wcześniej wyjaśnialiśmy, aktywnym routerem przepytującym (krótko: przepytywaczem[3]) jest wówczas ten z nich, który ma najmniejszy (liczbowo) adres IP; jest on odpowiedzialny za wysyłanie generalnych i specyficznych żądań. Pozostałe ze wspomnianych routerów — „zapasowe" — monitorują komunikaty protokołów, gotowe do wejścia w rolę przepytywacza w przypadku awarii dotychczasowego. Oprócz śledzenia żądań wysyłanych przez przepyty-

[3] W dokumentach RFC używane jest określenie *querier* — *przyp. tłum.*

wacz i otrzymywanych przez niego odpowiedzi koordynują one także wzajemnie swe informacje konfiguracyjne (przede wszystkim wskazania zegarów).

Pierwszym ze wspomnianych działań koordynacyjnych jest **elekcja przepytywacza**. Jak przed chwilą wspomnieliśmy, wszystkie routery multicast przyłączone do tego samego łącza znają nawzajem wysyłane przez siebie żądania. Każdy router multicast po rozpoczęciu pracy przyjmuje milcząco, iż to on jest przepytywaczem; gdy jednak odbierze żądanie od swego partnera o niższym adresie IP, zrzeka się tej roli na jego rzecz, a sam przechodzi w stan czuwania (*standby*). W ten oto sposób zwycięzcą przepytywaczem zostaje router o najniższym adresie IP; pozostałe znajdują się w stanie czuwania i jednocześnie bacznie obserwują okresowo swego aktywnego lidera. Gdy w założonym interwale czasowym (zwanym w oryginale *other-querier present timer*) nie sformułuje on żadnego żądania, istnieje podejrzenie, że uległ awarii. Rozpoczyna się wówczas na nowo procedura wyłaniania laureata o najniższym adresie IP.

Przepytywacz wysyła okresowo żądania generalne, w odstępach określonych przez konfigurowany parametr zwany **interwałem żądań** (*query interval*); routery zapasowe znają wartość tego interwału, dzięki czemu uaktywnienie któregoś z nich (w przypadku awarii przepytywacza) nie spowoduje zaburzenia rytmiczności wysyłania żądań.

Na podstawie otrzymywanych raportów przepytywacz może podjąć decyzję o zaprzestaniu forwardowania komunikatów kierowanych na określony adres multicast, przypuszczając, że grupa identyfikowana przez ten adres już nie istnieje. Ciężar gatunkowy takiej decyzji wymaga jednak uprzedniego upewnienia się o zasadności wspomnianego przypuszczenia — brak odpowiedzi ze strony rzeczonej grupy może równie dobrze być konsekwencją gubienia komunikatów. Na adres wspomnianej grupy wysłana zostaje zatem seria żądań, w odstępach czasu określonych przez parametr o nazwie *Last Member Query Time*, w skrócie LMQT, zwykle krótszy od wspomnianego wcześniej interwału żądań; wielkość tego interwału ma, jak łatwo zauważyć, zasadniczy wpływ na szybkość reagowania protokołu na zniknięcie grupy multicast. Konsekwentny brak odpowiedzi na wszystkie żądania skłania router do uznania, iż rzeczona grupa rzeczywiście zniknęła. Ten klarowny mechanizm komplikuje się jednak w sytuacji, gdy w trakcie opisanego testu wystąpi awaria dotychczasowego przepytywacza; zostanie on zastąpiony przez inny, wybrany w procedurze elekcji, co jednak nie zmienia faktu, iż prawdopodobnie zagubionych zostało kilka komunikatów.

Podstawowym środkiem przeciwdziałania konsekwencjom gubienia komunikatów jest retransmitowanie niektórych rodzajów komunikatów (konkretnie: żądań specyficznych oraz komunikatów z rekordami zmiany stanu) pewną niewielką liczbę razy, określoną przez parametr *QRV*. Routery zapasowe współdzielą tę wartość z aktywnym przepytywaczem, co jest kolejnym elementem strategii zachowania spójności między nimi. Gdy retransmisja komunikatu faktycznie dochodzi do skutku, odstęp czasowy między kolejnymi emisjami raportu ma wartość losową, nie większą niż parametr konfiguracyjny o nazwie **interwał niezamówionych raportów** (*unsolicited report interval*), zaś żądania retransmitowane są w odstępach czasu wynikających z LMQT. Łącza bardziej podatne na utratę komunikatów (np. łącza bezprzewodowe) muszą z konieczności używać większej wartości QRV, z korzyścią dla niezawodności, lecz za cenę generowania większego ruchu.

W celu ułatwienia utrzymania synchronizacji między routerem przepytywaczem wysyłającym żądania specyficzne a routerami zapasowymi wprowadzono do żądań IGMP/MLD bit *S*. Przepytywacz wielokrotnie wysyła każde żądanie specyficzne — liczba retransmisji określona jest przez licznik *QRV*. Gdy po wysłaniu pierwszego żądania nie nadejdzie odpowiedź (raport) w standardowym interwale *QQIT*, przepytywacz zmniejsza odstęp między kolejnymi retransmisjami do wartości LMQT; nadejście wspomnianego raportu powoduje przywrócenie standardowej wartości interwału retransmisji (retransmisje realizowane są nadal, mimo nadejścia odpowiedzi).

Routery zapasowe obserwują działania aktywnego przepytywacza i naśladują opisane zachowanie w postaci zmniejszania licznika retransmisji przy braku odpowiedzi, ale tylko wtedy, gdy wspomniany bit *S* ma wartość 0. Aktywny przepytywacz zeruje ten bit w pierwszym egzemplarzu wysyłanego żądania i ustawia w kolejnych retransmisjach, co chroni routery zapasowe przed zmniejszeniem interwału retransmisji. Ochrona taka jest konieczna, bo routery zapasowe mogłyby przeoczyć raport docierający do aktywnego przepytywacza i nie zresetować swych interwałów retransmisji do standardowego poziomu.

9.4.6. Zmienne i liczniki protokołów IGMP i MLD

Protokoły IGMP i MLD są protokołami funkcjonującymi w oparciu o informację typu „miękkiego stanu" (*soft state*), czyli informację uważaną za aktualną jedynie przez pewien odcinek czasu, po upływie którego musi ona zostać odświeżona albo skasowana. Częstotliwość tego odświeżania wynika z kilku parametrów konfiguracyjnych, których znaczenie opisujemy w tabeli 9.3. Parametry te obejmują ponadto licznik (opisywanej przed chwilą) retransmisji komunikatów, oznaczony w tabeli przez RV. Jak wynika z drugiej kolumny wspomnianej tabeli, niektóre z parametrów są wielkościami niezależnie konfigurowalnymi, wartość niektórych parametrów jest natomiast funkcją innych parametrów — tę drugą grupę oznaczyliśmy komentarzem „niezmienialny".

9.4.7. Podsłuchiwanie IGMP/MLD w warstwie 2.

Korzyści, jakie daje multicasting w porównaniu z broadcastingiem, można dodatkowo zwiększyć, wyposażając przełączniki działające na poziomie warstwy 2. w możliwość śledzenia i interpretowania komunikatów IGMP i MLD, plasujących się w warstwie 3. Funkcja ta, nazwana **podsłuchiwaniem IGMP/MLD** (*IGMP/MLD snooping*) i opisana w dokumencie [RFC4541], faktycznie implementowana jest przez wielu producentów przełączników. Bez niej ruch multicast realizowany jest na poziomie warstwy 2. w postaci rozgłaszania ramek we wszystkich gałęziach drzewa rozpinającego wyznaczanego przez przełączniki, co (z przyczyn wcześniej wyjaśnionych) wiąże się zwykle z marnotrawieniem zasobów sieci. Przełączniki uwrażliwione na wspomniane podsłuchiwanie (zwane skrótowo *IGS*, od *IGMP-snooping Switch*) potrafią eliminować zbędny ruch IGMP/MLD w odniesieniu do poszczególnych interfejsów na takiej samej zasadzie, jak normalnie robią to routery multicast.

Ta klarowna idea napotyka jednak na pewne komplikacje implementacyjne, związane z (opisywaną wcześniej) osobliwością protokołów IGMPv1, IGMPv2 oraz MLDv1. Otóż, jak pamiętamy, dopiero wersje IGMPv3 i MLDv2 narzuciły na każdy host obowiązek odpowiadania (w postaci raportu) na wszystkie żądania dotyczące grupy multicast, którą

Tabela 9.3. *Wartości interwałów czasowych i innych parametrów regulujących funkcjonowanie protokołów IGMP i MLD. Niektóre parametry są funkcją innych i jako takie nie podlegają bezpośredniemu konfigurowaniu*

Nazwa i znaczenie	Wartości domyślne i ograniczenia
`RV` (*Robustness Variable*) — zmienna niezawodności związana z retransmisją komunikatów: liczba retransmisji równa jest RV – 1	2; nie może być zerowa, powinna być większa od 1
`QI` (*Query Interval*) — odstęp czasowy między kolejnymi żądaniami generalnymi wysyłanymi przez aktywny przepytywacz	125 sekund
`QRI` (*Query Response Interval*) — maksymalny czas oczekiwania przepytywacza na odpowiedź; kodowany w polu *Maksymalne opóźnienie odpowiedzi*	10 sekund
`GMI` (*Group Membership Interval*) w IGMP, `MALI` (*Multicast Address Listening Interval*) w MLD — minimalny czas, jaki musi upłynąć od momentu wysłania żądania specyficznego, by router multicast miał prawo potraktować brak raportu jako świadectwo zniknięcia grupy multicast	`RV` · `QI` + `QRI` (niezmienialny)
`OQPI` (*Other Querier Present Interval*) w IGMP, `OQPT` (*Other Querier Present Timeout*) w MLD — minimalny czas, jaki musi upłynąć od wysłania ostatniego żądania przez aktywny przepytywacz, by router zapasowy mógł podejrzewać jego awarię	`RV` · `QI` + 0,5 · `QRI` (niezmienialny)
`SQI` (*Startup Query Interval*) — interwał między kolejnymi żądaniami generalnymi wysyłanymi przez router rozpoczynający pracę w roli przepytywacza	0,25 · `QI`
`SQC` (*Startup Query Count*) — liczba żądań generalnych wysyłanych w odstępach `SQI` przez router rozpoczynający pracę w roli przepytywacza	`RV`
`LMQI` (*Last Member Query Interval*) w IGMP, `LLQI` (*Last Listener Query Interval*) w MLD — maksymalny czas oczekiwania na raport stanowiący odpowiedź na żądanie specyficzne; kodowany w polu *Maksymalne opóźnienie odpowiedzi*	1 sekunda
`LMQC` (*Last Member Query Count*) w IGMP, `LLQC` (*Last Listener Query Count*) w MLD — liczba retransmisji żądania specyficznego, po wykonaniu której router multicast ma prawo potraktować brak raportu jako świadectwo zniknięcia grupy multicast	`RV`
`URI` (*Unsolicited Report Interval*) — odstęp czasowy między retransmisjami spontanicznych raportów	1 sekunda
`OVQPT` (*Older Version Querier Present Timeout*) — interwał czasowy, który musi upłynąć, by host po przetworzeniu ostatniego otrzymanego żądania IGMPv1/IGMPv2/MLDv1 powrócił do trybu IGMPv3/MLDv2	`RV` · `QI` + `QRI` (niezmienialny)
`OVHPI` (*Older Version Host Present Interval*) w IGMP, `OVHPT` (*Older Version Host Present Timeout*) w MLD — odstęp czasowy, który musi upłynąć, by przepytywacz po otrzymaniu ostatniego raportu IGMPv1/IGMPv2/MLDv1 powrócił do trybu IGMPv3/MLDv2	`RV` · `QI` + `QRI` (niezmienialny)

host jest zainteresowany (lub w której już uczestniczy); w wersjach poprzednich wystarczająca była odpowiedź z jednego hosta związanego z określoną grupą — pozostałe hosty, widząc akcję swego partnera, mogły czuć się z rzeczonego obowiązku zwolnione. I tu właśnie kryje się wspomniana komplikacja: gdy IGS nie stwierdzi nadejścia raportu na danym interfejsie, może niesłusznie uznać, że ten interfejs nie jest związany z daną grupą multicast, w konsekwencji czego niektóre hosty (należące do segmentu LAN lub VLAN przyłączonego do tegoż interfejsu) nie otrzymają należnych im komunikatów. Skutecznym rozwiązaniem tego problemu może być zablokowanie możliwości wzajemnego informowania się uczestników grupy o wysłanych raportach — istotnie, tak właśnie robią przełączniki kompatybilne z IGMPv1/IGMPv2/MLDv1, forwardując raporty tychże protokołów jedynie do najbliższego routera multicast (zamiast standardowego ich forwardowania przez wszystkie interfejsy). Odnajdywanie routerów multicast jest znacznie ułatwione dzięki protokołowi MRD, opisywanemu w punkcie 8.4.8.

Kolejna komplikacja implementacyjna bierze się z zasadniczej różnicy między protokołami IGMP a MLD: ten pierwszy jest niezależnym protokołem, a jego PDU enkapsulowane są w datagramach IPv4; komunikaty MLD są natomiast w istocie komunikatami ICMPv6, enkapsulowanymi w datagramach IPv6. Przełączniki implementujące podsłuchiwanie MLD muszą więc prawidłowo rozpoznawać komunikaty ICMPv6 i zapewniać prawidłowe ich forwardowanie w celu wypełniania ich standardowych funkcji (opisywanych w rozdziale 8.), a przy tym taktować w sposób specjalny komunikaty MLD.

Na potrzeby dalszej optymalizacji multicastingu IP na poziomie urządzeń warstwy 2. opracowano kilka niestandardowych, firmowych protokołów. Wśród nich wymienić należy przede wszystkim protokoły RGMP (*Router-port Group Management Protocol*) autorstwa firmy Cisco. Protokół ten rozszerza funkcje wysyłania raportów także na routery multicast, dzięki czemu zyskują one dodatkową informację ułatwiającą optymalizację forwardowania.

9.5. Ataki wykorzystujące IGMP i MLD

Jako że IGMP i MLD to protokoły sygnalizacyjne, odpowiedzialne za sterowanie ruchem w ramach multicastingu, stanowią one potencjalny obiekt lub narzędzie ataków typu DoS polegających na wyczerpującym angażowaniu zasobów systemu i jego mocy obliczeniowej. Ponadto, jak w przypadku wszelkiego oprogramowania, rozmaite luki ukrywające się w błędnych implementacjach stanowią wdzięczny obiekt zainteresowania pomysłowych hakerów.

Intensywny strumień spontanicznych raportów subskrybujących uczestnictwo w grupach multicast o dużym natężeniu ruchu doprowadza w krótkim czasie do wykorzystania dostępnej przepustowości, czyli skutecznego ataku DoS. Bardziej wyrafinowaną formą tego ataku jest generowanie żądań opatrywanych małym (liczbowo) adresem IP, dzięki czemu atakujący ma szansę wejść w rolę aktywnego przepytywacza (na zasadzie opisanej wcześniej procedury elekcyjnej) i następnie narzucać sieci restrykcyjne warunki pracy w postaci krótkich czasów dopuszczalnego opóźnienia, krótkich odstępów między żądaniami i dużych liczników retransmisji. W rezultacie hosty pochłonięte zostają intensywnym generowaniem raportów i na wykonywanie użytecznej pracy nie starcza już czasu i przepustowości.

Błędy w implementacjach protokołów, umiejętnie wykorzystywane, mogą stanowić znakomite furtki do paraliżowania i zawieszania pracy routerów. Wiele implementacji źle radzi sobie np. z pofragmentowanymi komunikatami IGMP, a szczególnie pofragmentowanymi w podstępny sposób. Od niedawna obserwuje się także (udane) próby wymuszania zdalnego wykonywania błędnego kodu, za pomocą specjalnie spreparowanych pakietów IGMP lub MLD zawierających informację związaną z SSM.

Jednak, ogólnie rzecz biorąc, konsekwencje błędów w implementacjach IGMP i MLD są relatywnie mniej groźne w porównaniu z innymi protokołami, ponieważ multicasting jest zwykle mechanizmem o zasięgu lokalnym; haker nieposiadający dostępu do łącza prowadzącego do danej sieci LAN ma ograniczoną możliwość oddziaływania na tę sieć za pośrednictwem protokołów IGMP i MLD.

9.6. Podsumowanie

Broadcasting, zwany także rozgłaszaniem, to w ogólnym znaczeniu dostarczanie kopii pakietu do wszystkich węzłów sieci. W kontekście protokołów grupy TCP/IP oznacza to rozsyłanie pakietów do wszystkich hostów w sieci (najczęściej lokalnej) lub podsieci. Multicasting to rozsyłanie informacji jedynie do wybranych węzłów sieci — w kontekście protokołów grupy TCP/IP do tych węzłów, które wyrażą zainteresowanie daną kategorią informacji, subskrybując w tym celu swe członkostwo w grupie multicast. W większości przypadków multicasting jest rozwiązaniem bardziej efektywnym od broadcastingu, bo generuje zwykle mniejszy ruch i pochłania mniej zasobów obliczeniowych. Płaconą za to ceną jest jednak większa złożoność koncepcyjna i implementacyjna multicastingu. Obie te techniki ułatwiają dostarczanie tej samej informacji do wielu miejsc przeznaczenia, uwalniając nadawcę od wysyłania jej do każdego adresata z osobna, każdorazowo z nawiązywaniem odrębnego połączenia. Drugim zadaniem multicastingu jest odnajdywanie serwerów rezydujących w nieznanej (pod)sieci. Broadcasting istniał od samych początków IPv4, multicasting pojawił się nieco później. Projektanci IPv6 zrezygnowali całkowicie z broadcastingu, rozbudowując znacznie możliwości multicastingu i czyniąc go mechanizmem krytycznym dla całego protokołu.

W protokole IPv4 zdefiniowano dwa typy adresów broadcast: adres ograniczonego rozgłaszania `255.255.255.255` oraz grupę adresów rozgłaszania ukierunkowanego. Każdy z adresów tej drugiej kategorii rozpoczyna się prefiksem podsieci, po którym następuje numer hosta złożony z samych bitów jedynkowych. Wybór interfejsu wykorzystywanego do wysyłania ruchu broadcast zależny jest od konkretnego systemu operacyjnego: zazwyczaj rozgłaszanie ograniczone wykonywane jest przez interfejs główny (*primary*), natomiast interfejs dla rozgłaszania ukierunkowanego i multicastingu wybierany jest na podstawie tablicy forwardowania utrzymywanej przez host.

W modelu multicastingu IP procesy zainteresowane otrzymywaniem określonych pakietów przyłączają się do określonej grupy multicast, identyfikowanej za pomocą odpowiedniego adresu grupowego, w kontekście określonego podzbioru interfejsów. W sieciach obsługujących multicasting na poziomie warstwy łącza danych w standardzie IEEE (czyli np. w sieciach Ethernet) sprzętowy adres multicast tworzony jest jako konkatenacja prefiksu `01:00:5e`, bitu `0` i 23 najmniej znaczących bitów adresu IPv4; w protokole IPv6 sprzętowy adres multicast jest natomiast konkatenacją prefiksu `33:33` i 32 najmniej zna-

czących bitów adresu IPv6. Tak określone mapowanie adresu IP na adres sprzętowy nie jest jednoznaczne — danemu adresowi sprzętowemu multicast odpowiada wiele adresów IP, dlatego niezbędne jest filtrowanie niechcianych pakietów przez oprogramowanie hosta na podstawie analizy ich rzeczywistych adresów IP.

W protokole IPv4 multicasting obsługiwany jest przez odrębny protokół IGMP, w IPv6 funkcję tę spełnia protokół MLD, faktycznie stanowiący część protokołu ICMPv6. Mimo iż IGMP i MLD są protokołami warstwy 3, dla większej optymalizacji multicastingu wyposaża się niektóre przełączniki działające w warstwie 2. w zdolność rozpoznawania i interpretowania komunikatów tych protokołów — co nazywane jest „podsłuchiwaniem" — *IGMP/MLD snooping* — i umożliwia dodatkową eliminację niepotrzebnego ruchu na poziomie warstwy łącza danych.

Ze względu na krytyczne znaczenie komunikatów IGMP/MLD z jednej strony, a możliwość ich gubienia w sieci z drugiej, protokoły te dysponują mechanizmami uodporniania na opisane zagrożenie (oraz awarie sprzętu) w postaci retransmitowania istotnych komunikatów i okresowego odświeżania istotnych informacji, na zasadzie „miękkiego stanu" (*soft state*). Mechanizmy te sterowane są przez zespół stoperów i parametrów konfiguracyjnych.

IGMP i MLD są protokołami sygnalizacyjnymi i jako takie stanowią obiekty potencjalnych ataków przeciążeniowych, powodowanych intensywnymi strumieniami żądań i raportów. Rozmaite błędy w implementacjach tych protokołów mogą także stanowić furtkę do zdalnego uruchamiania pasożytniczego kodu. Szczególnie wrażliwy jest pod tym względem protokół MLDv2 — jako jeszcze stosunkowo mało rozpowszechniony, rozpoznany jest w niewielkim stopniu pod kątem luk tkwiących w jego implementacjach i możliwości ich eksploatowania. Na szczęście, działanie protokołów multicastingu zamyka się zwykle w granicach sieci lokalnej i prowadzącego do niej łącza, więc zarówno możliwości ingerencji w funkcjonowanie tych protokołów, jak i potencjalne konsekwencje tej ingerencji mają zakres bardzo ograniczony.

9.7. Bibliografia

[CK11] S. Cheshire, M. Krochmal, *Multicast DNS*, Internet draft-cheshirednsext-multicastdns, w przygotowaniu, luty 2011.

[EGW02] B. Edwards, L. Giuliano, B. Wright, *Interdomain Multicast Routing: Practical Juniper Networks and Cisco Systems Solutions* (Addison-Wesley, 2002).

[GCLG99] Y. Goland, T. Cai, P. Leach, Y. Gu, *Simple Service Discovery Protocol/1.0 Operating without an Arbiter*, Internet draft-cai-ssdp-v1-03.txt (expired), październik 1999.

[OUI36] *http://standards.ieee.org/develop/regauth/oui36/oui36.txt*

[RFC1112] S. Deering, *Host Extensions for IP Multicasting*, Internet RFC 1112/STD 0005, sierpień 1989.

[RFC1122] R. Braden (red.), *Requirements for Internet Hosts*, Internet RFC 1122/STD 0003, październik 1989.

[RFC2644] D. Senie, *Changing the Default for Directed Broadcasts in Routers*, Internet RFC 2644/BCP 0034, sierpień 1999.

[RFC3376] B. Cain, S. Deering, I. Kouvelas, B. Fenner, A. Thyagarajan, *Internet Group Management Protocol, Version 3*, Internet RFC 3376, październik 2002.

[RFC3488] I. Wu, T. Eckert, *Cisco Systems Router-port Group Management Protocol (RGMP)*, Internet RFC 3488 (informational), luty 2003.

[RFC3590] B. Haberman, *Source Address Selection for the Multicast Listener Discovery (MLD) Protocol*, Internet RFC 3590, wrzesień 2003.

[RFC3810] R. Vida, L. Costa (red.), *Multicast Listener Discovery Version 2 (MLDv2) for IPv6*, Internet RFC 3810, czerwiec 2004.

[RFC4541] M. Christensen, K. Kimball, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, Internet RFC 4541 (informational), maj 2006.

[RFC4601] B. Fenner, M. Handley, H. Holbrook, I. Kouvelas, *Protocol Independent Multicast—Sparse Mode (PIM-SM): Protocol Specification (Revised)*, Internet RFC 4601, sierpień 2006.

[RFC4604] H. Holbrook, B. Cain, B. Haberman, *Using Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Protocol Version 2 (MLDv2) for Source-Specific Multicast*, Internet RFC 4604, sierpień 2006.

[RFC4607] H. Holbrook, B. Cain, *Source-Specific Multicast for IP*, Internet RFC 4607, sierpień 2006.

[RFC4795] B. Aboba, D. Thaler, L. Esibov, *Link-Local Multicast Name Resolution (LLMNR)*, Internet RFC 4795 (informational), styczeń 2007.

[RFC5015] M. Handley, I. Kouvelas, T. Speakman, L. Vicisano, *Bidirectional Protocol Independent Multicast (BIDIR-PIM)*, Internet RFC 5015, październik 2007.

[RFC5214] F. Templin, T. Gleeson, D. Thaler, *Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)*, Internet RFC 5214 (informational), marzec 2008.

[RFC5579] F. Templin (red.), *Transmission of IPv4 Packets over Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) Interfaces*, Internet RFC 5579 (informational), luty 2010.

[RFC5790] H. Liu, W. Cao, H. Asaeda, *Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols*, Internet RFC 5790, luty 2010.

Rozdział 10. Protokół datagramów użytkownika (UDP) oraz fragmentacja IP

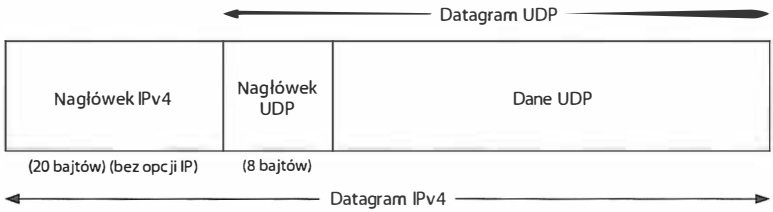
10.1. Wprowadzenie

UDP jest nieskomplikowanym, operującym datagramami protokołem transportowym, który zapewnia transmisję danych w ramach pojedynczych komunikatów. Nie uwzględnia mechanizmów korekcji błędów, kontroli kolejności wiadomości, eliminowania duplikatów, sterowania przepływem lub ograniczania przeciążeń. Może natomiast obejmować detekcję błędów przy użyciu sum kontrolnych dodawanych i analizowanych przez jednostki końcowe w warstwie transportowej. Funkcje samego protokołu są więc ograniczone do minimum, a główne zadania związane z wysyłaniem i przetwarzaniem pakietów pozostają w gestii aplikacji. Oznacza to, że zapewnienie poprawnego dostarczenia danych i zagwarantowanie odpowiedniej kolejności datagramów należy do obowiązków twórcy aplikacji. Zazwyczaj każda operacja wysłania danych za pomocą protokołu UDP powoduje utworzenie dokładnie jednego datagramu UDP, który następnie wymusza utworzenie jednego pakietu IP. Rozwiązanie to odbiega od sposobu działania protokołów strumieniowych, takich jak TCP (patrz rozdział 15.), w których ilość danych generowanych przez aplikację nie przekłada się bezpośrednio na zawartość poszczególnych datagramów IP i na komunikaty przetwarzane przez odbiorcę.

Oficjalna specyfikacja protokołu UDP została zapisana w dokumencie [RFC0768] i obowiązuje jako standard od ponad 30 lat (bez istotniejszych zmian). Zgodnie z wcześniejszym stwierdzeniem mechanizm UDP nie obejmuje żadnych procedur korekcji błędów. Oznacza to, że wysyła datagramy przekazywane przez aplikację do warstwy IP bez gwarancji, że dotrą one do wskazanego celu. Ponadto nie uwzględnia żadnego mechanizmu, który wyeliminowałby negatywny wpływ dużego natężenia ruchu UDP na pracę innych użytkowników sieci. Można by więc sądzić, że brak gwarancji niezawodności transmisji i ochrony danych sprawiają, że protokół UDP jest bezużyteczny. Nie jest to jednak prawdą. Z uwagi na bezpołączeniowy charakter mechanizm UDP wnosi mniejszy narzut transmisyjny niż inne protokoły sieciowe. Poza tym multiemisja i transmisja rozgłoszeniowa są znacznie łatwiejsze do zrealizowania właśnie w przypadku użycia protokołów

bezpołączeniowych, takich jak UDP (więcej informacji na ten temat znajduje się w rozdziale 9.). W niektórych zastosowaniach bardzo przydatna może się również okazać możliwość zdefiniowania własnego systemu retransmisji na poziomie aplikacji (przykładem jest tutaj opracowanie [CT90]).

Zasada przenoszenia datagramu UDP w pakiecie IPv4 została zilustrowana na rysunku 10.1. Analogiczny mechanizm znajduje zastosowanie w systemach IPv6, choć są w nim nieznaczne odstępstwa, które zostały opisane w podrozdziale 10.5. Do oznaczenia segmentu UDP w pakiecie IPv4 służy wartość 17, która jest zapisywana w polu *Protokół* nagłówka IP. Ta sama wartość jest wykorzystywana w polu *Następny nagłówek* datagramu IPv6. W dalszej części rozdziału zostały opisane zasady postępowania w przypadku przekroczenia przez datagram UDP rozmiaru określonego za pomocą parametru MTU i konieczności podzielenia go na kilka pakietów IP.

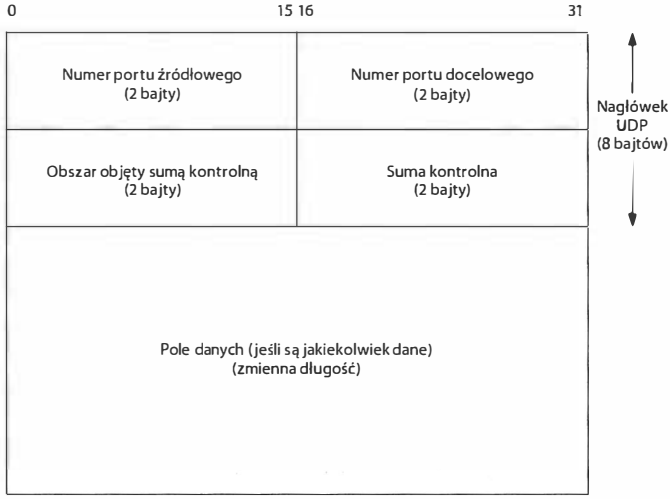


Rysunek 10.1. *Przenoszenie datagramu UDP w pojedynczym datagramie IPv4 (typowy przypadek; bez opcji IPv4). Enkapsulacja w protokole IPv6 jest analogiczna — nagłówek UDP jest zapisywany na końcu sekwencji nagłówków IPv6*

10.2. Nagłówek UDP

Struktura datagramu UDP obejmująca pole danych i nagłówek (którego rozmiar zawsze wynosi 8 bajtów) została przedstawiona na rysunku 10.2.

Numery portów pełnią funkcję **skrzynek na listy**, umożliwiając modułowi obsługi protokołu identyfikację procesów nadawczych i odbiorczych (więcej informacji na ich temat znajduje się w rozdziale 1.). Mają czysto **wirtualny** charakter — nie są powiązane z żadnymi fizycznymi komponentami komputera. W protokole UDP porty są 16-bitowymi liczbami dodatnimi. Wartość portu źródłowego jest opcjonalna i może zostać ustawiona na 0, jeśli aplikacja nie wymaga odpowiedzi ze strony odbiorcy. Numery portów docelowych umożliwiają protokołom TCP, UDP i SCTP [RFC4960] wyodrębnienie odpowiednich komunikatów z danych dostarczanych przez protokół IP. Z kolei wyodrębnienie datagramów poszczególnych protokołów transportowych na poziomie warstwy IP następuje na podstawie wartości pola *Protokół* zapisanej w nagłówku IPv4 lub pola *Następny nagłówek* pakietu IPv6. Oznacza to, że numery portów mogą być wykorzystywane niezależnie w ramach każdego z protokołów transportowych — porty TCP są przetwarzane jedynie przez protokół TCP, porty UDP jedynie przez protokół UDP itd. Wynika z tego, że dwa niezależne serwery mogą posługiwać się tymi samymi numerami portów i adresami IP, jeśli wykorzystują różne protokoły transportowe.



Rysunek 10.2. *Nagłówek i pole danych protokołu UDP. Pole sumy kontrolnej jest wyliczane przez jednostki końcowe z uwzględnieniem pseudonagłówka UDP, który obejmuje adresy IP (źródłowy i docelowy) nagłówka IP. Oznacza to, że wszelkie modyfikacje tych pól (np. w operacji NAT) wymagają wyznaczenia nowej wartości sumy kontrolnej*

Mimo opisanej niezależności protokołów, usługom internetowym pracującym w protokołach TCP i UDP przypisuje się zazwyczaj jednakowe numery portów w ramach każdego z protokołów transportowych. Taki sposób postępowania wynika jedynie ze względów praktycznych i nie jest narzucany przez specyfikację samych protokołów. Więcej informacji na temat przydziału numerów portów znajduje się w opracowaniu [IPORT].

Przedstawione na rysunku 10.2 pole *Długość* odpowiada rozmiarowi nagłówka i pola danych wyrażonemu w bajtach. Minimalną wartością pola jest 8, poza przypadkiem wykorzystania segmentu UDP w jumbogramie IPv6 (patrz podrozdział 10.5). Możliwe jest również wysłanie datagramu UDP bez danych (choć taka operacja należy do rzadkości). Warto zwrócić uwagę na fakt, że pole *Długość* w nagłówku UDP jest nadmiarowe — w nagłówku IPv4 (opisanym w rozdziale 5.) jest zapisany całkowity rozmiar datagramu, a nagłówek IPv6 przechowuje informację o wielkości pola danych. Rozmiar segmentu UDP przenoszony w protokole IPv4 jest więc różnicą między długością datagramu IPv4 i długością nagłówka IPv4. Rozmiar segmentu UDP dostarczanego za pomocą protokołu IPv6 odpowiada natomiast wartości pola *Długość pola danych* z nagłówka IPv6 pomniejszonej o rozmiar ewentualnych nagłówków rozszerzeń (o ile nie są stosowane jumbogramy). W obu przypadkach wartość pola *Długość* nagłówka UDP powinna odpowiadać wartości wyliczonej na podstawie informacji pozyskanych z pakietu IP.

10.3. Suma kontrolna

Suma kontrolna UDP jest pierwszą omawianą wartością kontrolną protokołu transportowego, która jest wyliczana przez jednostki końcowe (analogiczna suma kontrolna znajduje zastosowanie w protokole ICMP, ale protokół ten nie jest mechanizmem transportowym). Obejmuje nagłówek UDP, dane UDP oraz pseudonagłówek (opisany w dalszej części podrozdziału). Jest obliczana po stronie nadawcy i weryfikowana w urządzeniu docelowym.

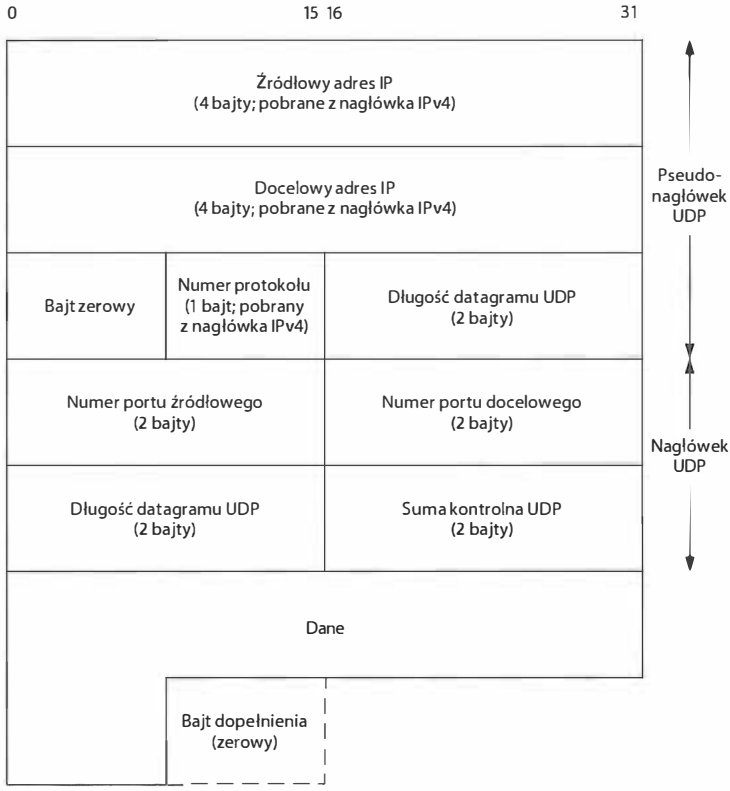
Wartość sumy kontrolnej nie ulega zmianie podczas transportu datagramu (poza przypadkiem translacji adresów sieciowych [NAT], opisanym w rozdziale 7.). Zgodnie z zamieszczonymi wcześniej informacjami suma kontrolna protokołu IPv4 obejmuje jedynie nagłówek (nie uwzględnia danych zawartych w pakiecie IP) i jest zmieniana przez każdy węzeł na trasie (z powodu dekrementacji wartości *TTL* w czasie przekazywania pakietu). Protokoły transportowe (np. TCP i UDP) wykorzystują sumy kontrolne do zabezpieczenia nagłówków *i* danych. W mechanizmie UDP operacja generowania sumy kontrolnej jest opcjonalna (ale zalecana), natomiast w innych protokołach ma charakter obowiązkowy. Obliczanie sumy kontrolnej jest również konieczne w rozwiązaniach IPv6, ponieważ nie ma tego typu wartości na poziomie warstwy IP. Aby zapewnić bezbłędne dostarczanie informacji do aplikacji, protokoły transportowe muszą wyznaczać sumy kontrolne lub korzystać z innych mechanizmów detekcji błędów, które są uwzględniane przed dostarczeniem danych do aplikacji odbiorczej.

Choć zasada obliczania opisywanych wartości jest analogiczna do opisanej w rozdziale 5. w odniesieniu do internetowych sum kontrolnych (wyznaczenie dopełnienia do jedności z sumy dopełnień do jedności 16-bitowego słowa), różni się nieznacznie w dwóch aspektach. Po pierwsze, rozmiar datagramu UDP może odpowiadać nieparzystej liczbie bajtów, a algorytm obliczania sumy kontrolnej dodaje słowa 16-bitowe (liczba bajtów zawsze jest parzysta). W przypadku protokołu UDP konieczne jest więc dodanie wirtualnego bajta dopełnienia (złożonego z samych zer) do datagramu o nieparzystej liczbie bajtów (tylko na potrzeby obliczenia i sprawdzenia wartości kontrolnej). Sam bajt dopełnienia nie jest transmitowany przez sieć, dlatego jest określany jako wirtualny.

Druga różnica wynika z uwzględnienia w mechanizmie UDP (a także w TCP) 12-bajtowego **pseudonagłówka** wyznaczonego na podstawie wybranych pól nagłówka IPv4 lub 40-bajtowego pseudonagłówka bazującego na polach nagłówka IPv6. Wspomniany pseudonagłówek również ma charakter wirtualny i służy jedynie do wyliczenia sumy kontrolnej (zarówno po stronie nadawcy, jak i po stronie odbiorcy). Nie jest wysyłany wraz z danymi. Do jego wyznaczenia wykorzystuje się adresy IP źródłowy i docelowy oraz pole *Protokół* lub *Następny nagłówek* (które w protokole UDP ma wartość 17) z nagłówka IP. Dzięki temu urządzenie docelowe może na poziomie warstwy transportowej sprawdzić, czy dane zostały dostarczone do właściwego odbiorcy (czy warstwa IP nie zaakceptowała błędnie zaadresowanego pakietu oraz czy nie został dostarczony datagram przeznaczony dla innego protokołu transportowego). Dane uwzględniane w sumie kontrolnej zostały przedstawione na rysunku 10.3. Zaprezentowano na nim sam pseudonagłówek oraz zasadniczy nagłówek UDP i pole danych protokołu UDP.

Wnikliwy czytelnik zauważy, że działanie opisanego mechanizmu stanowi **naruszenie zasady niezależności warstw.** Protokół UDP (element warstwy transportowej) bezpośrednio przetwarza bity „należące" do protokołu IP (elementu warstwy sieciowej). Odstępstwo to nie ma jednak istotnego znaczenia w przypadku implementowania mechanizmów obsługi protokołu, ponieważ informacje warstwy IP są zazwyczaj dostępne podczas przekazywania danych do (lub z) warstwy UDP. Znacznie większym problemem jest natomiast wykonanie operacji NAT (opisanej w rozdziale 7.), szczególnie wtedy, gdy datagramy UDP podlegają fragmentacji.

Na rysunku 10.3 zaprezentowany został datagram o nieparzystej liczbie bajtów, wymagający dopełnienia bajtem zerowym przed rozpoczęciem obliczania sumy kontrolnej. Warto zwrócić uwagę na to, że rozmiar segmentu UDP występuje dwukrotnie w obszarze objętym mechanizmem kontrolnym. Jeśli wynikiem operacji jest wartość 0x0000, zostaje



Rysunek 10.3. *Pola wykorzystywane do obliczania sumy kontrolnej datagramu UDP przenoszonego przez protokół IPv4. Obejmują pseudonagłówek, nagłówek UDP oraz dane. Jeśli rozmiar pola danych nie odpowiada parzystej liczbie bajtów, na potrzebny obliczeń jest uzupełniany bajtem złożonym z samych zer logicznych. Pseudonagłówek i bajt dopełnienia nie są transmitowane przez sieć*

ona zapisana jako słowo złożone z samych jedynek logicznych (`0xFFFF`), co jest odpowiednikiem zera w arytmetyce uzupełnienia do jedności (więcej informacji na ten temat znajduje się w rozdziale 5.). Odebranie sumy kontrolnej o wartości `0x0000` oznacza, że nadawca w ogóle jej nie obliczył. Jeśli jednak wartość *została* wyznaczona, ale nie zgadza się z wynikiem analogicznej operacji wykonanej po stronie odbiorcy, datagram UDP zostaje usunięty. Nie wiąże się to z generowaniem jakichkolwiek komunikatów o błędach, choć wystąpienie opisanego zdarzenia może zostać odnotowane w danych statystycznych (podobnie jak w przypadku niezgodności sumy kontrolnej nagłówka IPv4).

Mimo że zgodnie ze specyfikacją UDP wyznaczanie sum kontrolnych jest opcjonalne, dokument [RFC1122] narzuca obowiązek domyślnego ich generowania przez wszystkie stacje korzystające z protokołu UDP. W latach 80. ubiegłego wieku część producentów systemów informatycznych wyłączała funkcję obliczania wartości kontrolnych w celu zwiększenia wydajności kodu obsługującego sieciowy system plików firmy SUN (NFS — *Network File System*). I choć dzięki mechanizmowi CRC implementowanemu na po-

ziomie warstwy drugiej (który jest znacznie silniejszy niż internetowe sumy kontrolne; patrz rozdział 3.) w większości przypadków nie stanowi to problemu, domyślne rezygnowanie z zabezpieczenia jest uznawane za działanie niewłaściwe (i naruszające zalecenia RFC). Doświadczenia z transmisją danych w Internecie w początkowej fazie jego rozwoju dowodzą, że podczas przekazywania datagramów przez routery może się zdarzyć wszystko. Choć trudno w to uwierzyć, w sieci pracowały urządzenia, których błędy w oprogramowaniu lub w warstwie sprzętowej powodowały zmianę bitów w przesyłanych datagramach. W przypadku rezygnacji z sum kontrolnych weryfikowanych po stronie jednostek końcowych błędy te pozostawały niezauważone. Ponadto wiele starszych protokołów warstwy łącza danych (np. SLIP) nie zapewniało żadnych mechanizmów weryfikacji sum kontrolnych na poziomie drugiej warstwy modelu OSI, umożliwiając wprowadzanie zmian w pakietach IP, które nie były zabezpieczane przez algorytmy wdrażane w wyższych warstwach stosu protokołów.

Zgodnie z zaleceniem [RFC1122] funkcje obliczania sum kontrolnych protokołu UDP powinny być włączone domyślnie. Jednocześnie odbiorca jest zobowiązany do zweryfikowania wartości kontrolnej, jeśli została ona wyznaczona przez nadawcę (tj. wartość sumy kontrolnej jest różna od zera).

Ze struktury pseudonagłówka wynika, że podczas wykonywania operacji NAT na datagramie UDP przenoszonym przez protokół IPv4 konieczne jest zmodyfikowanie nie tylko sumy kontrolnej protokołu IP, ale również wartości zabezpieczającej pseudonagłówek UDP. Obowiązek ten wynika ze zmian wprowadzanych w adresach IP i (opcjonalnie) w numerach portów. Funkcje translacji adresów sieciowych z założenia naruszają więc zasadę niezależności warstw przez wprowadzanie modyfikacji na kilku poziomach przetwarzania pakietów. Oczywiście, nie ma innej możliwości zaimplementowania tego mechanizmu, skoro operowanie pseudonagłówkiem samo w sobie jest naruszeniem wspomnianej zasady. Zasady wykonywania translacji adresów sieciowych w odniesieniu do ruchu UDP zostały zdefiniowane w dokumencie [RFC4787]. Ich krótkie podsumowanie znajduję się również w rozdziale 7.

Ostatnio obserwuje się zainteresowanie pewnym „złagodzeniem" reguł generowania sum kontrolnych w aplikacjach, które są w pewnym stopniu nienarażone na błędy (np. aplikacje multimedialne). Rozpoczęła się więc dyskusja na temat tego, czy **częściowe sumy kontrolne** są wartościowymi elementami komunikacji. Obejmują one jedynie fragment definiowanego przez aplikację pola danych. Więcej informacji na ich temat znajduje się w podrozdziale 10.6, w omówieniu protokołu UDP-Lite.

10.4. Przykłady

Analiza protokołu UDP została przeprowadzona z wykorzystaniem programu `sock` [SOCK], który posłużył do generowania datagramów UDP przechwytywanych za pomocą narzędzia `tcpdump`. Celem pierwszego testu jest uruchomienie serwera na porcie usługi Discard (porcie o numerze 9). Drugi test polega na wyłączeniu serwera i zaprezentowaniu sposobu informowania klienta o niedostępności usługi. Ze względów bezpieczeństwa w standardowej konfiguracji systemu dostępnych jest niewiele usług UDP. Komunikat przedstawiony w drugim przykładzie nie jest więc niczym wyjątkowym.

```
Linux% sock -v -u -i 10.0.0.3 discard
connected on 10.0.0.5.46274 to 10.0.0.3
wrote 1024 bytes
...                                                    Powtórzone 1023 razy

Linux% sock -v -u -i 10.0.0.3 discard
connected on 10.0.0.5.46294 to 10.0.0.3
wrote 1 bytes
write returned -1, expected 1024: Connection refused   Połączenie odrzucone
```

Polecenie uruchamiające program `sock` zostało uzupełnione o opcję `-v` odpowiedzialną za wyświetlanie szczegółowych informacji (w tym numerów portów), opcję `-u` oznaczającą użycie protokołu UDP zamiast standardowego TCP oraz opcję `-i`, która zapewnia przesłanie danych zamiast odczytywania standardowego strumienia wejściowego i zapisywania informacji w standardowym strumieniu wyjściowym. Działanie programu sprowadza się do wysłania domyślnej liczby (1024) datagramów do jednostki docelowej o adresie IP `10.0.0.3`. Do przetwarzania wygenerowanych segmentów został wykorzystany serwer pracujący na porcie usługi Discard. Przechwycenie przenoszonego ruchu wymaga wykonania następującej instrukcji w systemie komputera, który ma dostęp do przekazywanych danych:

```
Linux# tcpdump -n -p -s 1500 -vvv host 10.0.0.3 and \( udp or icmp \)
```

Przedstawione polecenie odpowiada za wyświetlenie informacji o wszelkich formach komunikacji w protokołach UDP i ICMP pomiędzy dwoma komputerami (ewentualnie również o innych rodzajach ruchu, które nie zostały tutaj pokazane). Opcja `-s 1500` informuje program `tcpdump` o obowiązku gromadzenia do 1500 bajtów z przechwytywanych pakietów (w omawianym przykładzie wysyłane pakiety o rozmiarze przekraczającym 1024 bajty). Opcja `-vvv` włącza funkcję generowania szczegółowych informacji wyjściowych. Z kolei funkcja `-n` wyłącza mechanizm odwzorowywania adresów IP na adresy domenowe, a opcja `-p` zapobiega włączeniu trybu zbiorczego (*promiscous*) na danym interfejsie. Wynik wykonania instrukcji powinien być zbliżony do zaprezentowanego na listingu 10.1 (niektóre wiersze listingu zostały przełamane w celu zwiększenia czytelności zestawienia).

Listing 10.1. *Wynik działania narzędzia tcpdump przedstawiający pakiety wygenerowane przez program sock (w czasie pracy serwera)*

```
1 22:52:53.102838 10.0.0.5.46274 > 10.0.0.3.9:
                  [udp sum ok] udp 1024 (DF) (ttl 64, id 24462, len 1052)
2 22:52:53.102964 10.0.0.5.46274 > 10.0.0.3.9:
                  [udp sum ok] udp 1024 (DF) (ttl 64, id 24463, len 1052)
3 22:52:53.103091 10.0.0.5.46274 > 10.0.0.3.9:
                  [udp sum ok] udp 1024 (DF) (ttl 64, id 24464, len 1052)
4 22:52:53.103215 10.0.0.5.46274 > 10.0.0.3.9:
                  [udp sum ok] udp 1024 (DF) (ttl 64, id 24465, len 1052)
... Powtórzone 1020 razy ...
```

Z listingu wynika, że z jednostki o adresie `10.0.0.5` zostały wysłane datagramy UDP/IPv4 o rozmiarze 1052 bajtów (1024 bajty pola danych segmentu UDP, 8 bajtów nagłówka UDP oraz 20 bajtów nagłówka IPv4). Numer portu źródłowego to `46274`, a docelowego `9` (port usługi Discard). Przerwa między pakietami wynosiła średnio 100 µs. Ponadto wiadomo,

że sumy kontrolne UDP zostały wyznaczone i są poprawne (sprawdzone przez program tcpdump). Ustawiono bit **zakazu fragmentowania** (DF — *Don't Fragment*). Parametr *TTL* ma wartość 64, a wartość identyfikatora pakietu jest zwiększana o jeden wraz z transmisją każdego datagramu. Nie został wygenerowany żaden pakiet ICMP. Wydaje się, że wszystkie dane poprawnie dotarły do celu, choć z powodu braku potwierdzeń nie można mieć stuprocentowej pewności. W rozdziale 13. został opisany inny ważny protokół transportowy — TCP — który przed rozpoczęciem transmisji informacji uruchamia procedurę uzgadniania połączenia z jednostką zdalną i przesyła potwierdzenia informujące o poprawnym odebraniu kolejnych porcji danych.

W drugim przykładzie program sock został uruchomiony z identycznymi parametrami, ale dostarczanie datagramów nie powiodło się z powodu wyłączenia usługi Discard. Przebieg transmisji odzwierciedla listing 10.2 (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.2. *Wynik działania narzędzia tcpdump — informacja o wysłaniu komunikatu ICMP o niedostępności celu (niedostępności portu)*

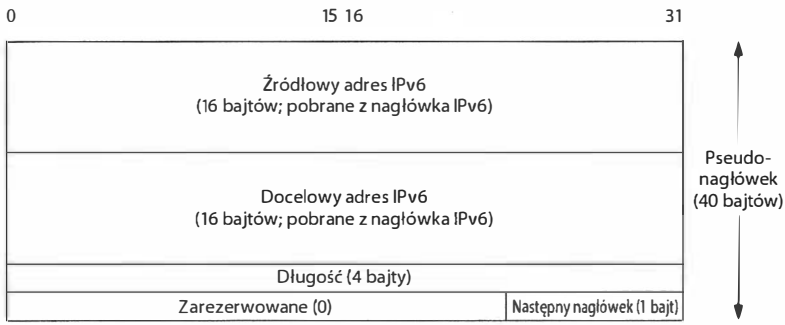
```
1 22:55:07.223094 10.0.0.5.46294 > 10.0.0.3.9:
                  [udp sum ok] udp 1024 (DF) (ttl 64, id 37874, len 1052)
2 22:55:07.223134 10.0.0.3 > 10.0.0.5: icmp:
                  10.0.0.3 udp port 9 unreachable for
                     10.0.0.5.46294 > 10.0.0.3.9:
                           udp 1024 (DF) (ttl 64, id 37874, len 1052)
                  [tos 0xc0] (ttl 255, id 63302, len 576)
```

W zaprezentowanym przykładzie można zaobserwować nieco inne działanie programu. W tym przypadku tylko jeden datagram UDP został przesłany, a w odpowiedzi nadawca otrzymał komunikat ICMP. Wszystkie parametry polecenia są takie same jak w pierwszym ćwiczeniu, ale nie funkcjonuje serwer, który mógłby odbierać datagramy. Mechanizm obsługi UDP generuje więc komunikat ICMPv4 Destination Unreachable (Port Unreachable) informujący o niedostępności portu docelowego (więcej informacji na temat protokołu ICMP znajduje się w rozdziale 8.). Przesyłane powiadomienie zawiera 556 pierwszych bajtów z pierwotnego (niedostarczonego) datagramu. Jeśli komunikat ICMP nie zostanie usunięty z sieci przez którekolwiek z urządzeń pośredniczących (przypadkowo lub celowo, np. przez zaporę sieciową), aplikacja nadawcza otrzymuje informację o niedostępności odbiorcy i wyświetla komunikat o błędzie. W analizowanym przykładzie odpowiada za to wiersz `write returned -1`. Powiadomienie o problemie obejmuje dostatecznie dużo danych, aby jednostka nadawcza mogła ustalić, który z portów jest niedostępny. Warto również zwrócić uwagę na to, że w kolejnych operacjach numer portu źródłowego jest zwiększany. W pierwszym przykładzie miał wartość 46274, a w drugim 46294. Zgodnie z informacją zawartą w rozdziale 1. numery portów klienckich są przydzielane w zakresie od 49152 do 65535. Działanie mechanizmu przydziału portów okazuje się więc niezgodne z tymi założeniami.

W systemach Linux zmiana zakresu portów lokalnych nie stanowi żadnej trudności. Sprowadza się do zmiany zawartości pliku */proc/sys/net/ipv4/ip_local_port_range*. W systemach Windows Vista i późniejszych trzeba do tego celu wykorzystać polecenie `netsh` [KB929851]. Zestawienie obowiązujących numerów portów znajduje się w opracowaniu [IPORT].

10.5. Datagramy UDP w sieciach IPv6

Prostota mechanizmu UDP pozwala na przeniesienie go z sieci IPv4 do sieci IPv6 bez większych modyfikacji. Największa różnica między poszczególnymi wersjami protokołu jest zauważalna podczas budowania pseudonagłówka, który musi uwzględniać 128-bitowy adres protokołu IPv6. Warto jednak pamiętać, że istnieje jeszcze jedna, nieco mniej oczywista różnica. W nagłówku protokołu IPv6 nie występuje suma kontrolna. Jeśli zatem w działaniu modułu UDP zostanie wyłączone obliczanie sum kontrolnych, **żaden mechanizm nie zapewni detekcji błędów** w adresowaniu IP. Z tego wynika, że podczas wykorzystywania protokołu UDP w sieciach IPv6 należy obowiązkowo włączyć generowanie sum kontrolnych pseudonagłówka (zgodnie z sekcją 8. dokumentu [RFC2460]). Struktura wspomnianego pseudonagłówka jest jednakowa w protokołach UDP i TCP, i została przedstawiona na rysunku 10.4. Pole *Długość* zostało w tym przypadku rozszerzone w porównaniu z swoim poprzednikiem do 32 bitów. Jak wiadomo, w przypadku zastosowania protokołu UDP ma ono charakter nadmiarowy. Nie jest jednak powielane w pakietach przenoszących segmenty TCP (zarówno TCP/IPv4, jak i TCP/IPv6), o czym informujemy w rozdziale 13. Zostało więc zachowane również w protokole UDP/IPv6.



Rysunek 10.4. *Pseudonagłówek UDP (i TCP) używany w protokole IPv6 (RFC2460]). Obejmuje adresy IPv6 źródłowy i docelowy oraz 32-bitowe pole długości datagramu. Podczas przenoszenia segmentu UDP w pakiecie IPv6 obliczanie sumy kontrolnej jest obowiązkowe, ponieważ w samym nagłówku IPv6 nie występuje żadna wartość kontrolna. Pole Następny nagłówek jest kopiowane z ostatniego elementu w sekwencji nagłówków IPv6*

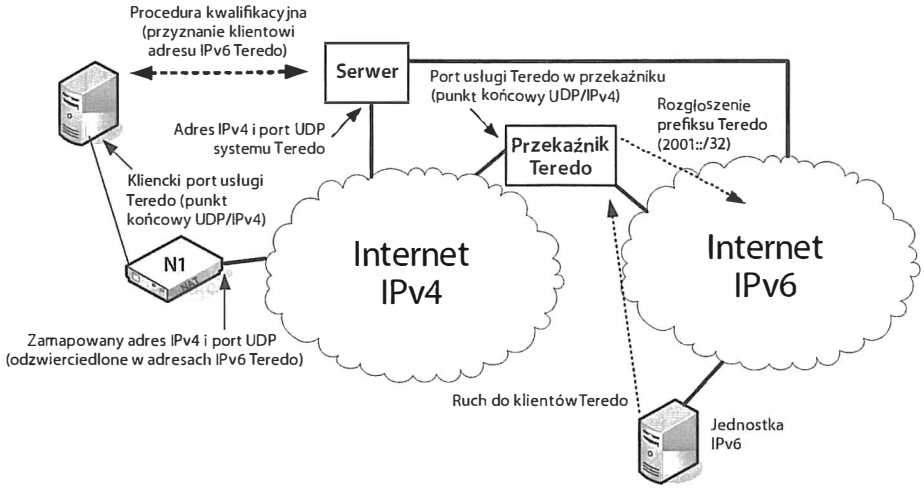
Na sposób formowania datagramów UDP wpływają również dwa założenia dotyczące rozmiaru pakietów IPv6. Po pierwsze, w standardzie IPv6 minimalna wartość parametru MTU wynosi 1280 bajtów (a nie jak w IPv4 576 bajtów). Po drugie, protokół IPv6 obsługuje **jumbogramy** (pakiety o rozmiarze większym niż 65 535 bajtów). Analizując nagłówek IPv6 i towarzyszące mu opcje (opisane w rozdziale 5.), nietrudno zauważyć, że w jumbogramach do przechowywania informacji o długości pola danych służą 32 bity. Oznacza to, że ilość danych przenoszonych w datagramie UDP/IPv6 może być naprawdę bardzo duża. Z tego faktu wynika jednak pewien problem, który został opisany w dokumencie [RFC2675]. Pole *Długość* w nagłówku UDP jest zapisywane na 16 bitach. Dlatego datagramy o rozmiarach przekraczających 65 535 bajtów zapisywane w pakietach IPv6 muszą mieć ustawione pole *Długość pola danych* na 0. Niemniej pole *Długość* występujące w pseudonagłówku jest dostatecznie duże, aby przechować rzeczywisty rozmiar obszaru danych (32 bity). Obliczając wartość pola w jumbogramie IPv6, należy więc

zsumować rozmiar nagłówka UDP i rozmiar pola danych. Z kolei sprawdzenie pola po stronie odbiorczej wymaga ustalenia rozmiaru datagramu UDP (nagłówka z danymi) przez odjęcie od wartości zdefiniowanej w opcji *Jumbo Payload* (definiującej jumbogram) rozmiaru wszystkich nagłówków rozszerzeń IPv6 (zazwyczaj sprowadza się do odjęcia 40 bajtów nagłówka IPv6 od całkowitej długości datagramu). W mało prawdopodobnym przypadku, gdy pole *Długość* w nagłówku UDP ma wartość 0 i nie jest wykorzystywana opcja *Jumbo Payload*, rozmiar datagramu UDP można ustalić na podstawie niezerowej wartości *Długość pola danych* pakietu IPv6 (więcej informacji na ten temat znajduje się w sekcji 4. dokumentu [RFC2675]).

10.5.1. Teredo — tunelowanie datagramów IPv6 w sieciach IPv4

Ogólnoświatowe wdrożenie protokołu IPv6 okazało się znacznie trudniejsze niż początkowo przypuszczano. Trudności te stały się z kolei przyczyną sporego bałaganu spowodowanego przez wprowadzenie wielu (teoretycznie chwilowych) **mechanizmów przejścia** [RFC4213] [RFC5969]. Jednym ze wspomnianych mechanizmów jest system **6to4** [RFC3056], w którym pakiety IPv6 (generowane przez stacje końcowe) mogą być przenoszone w pakietach IPv4 w klasycznej infrastrukturze sieciowej. Jednak największą wadą mechanizmu 6to4 jest podatność na te same niedogodności wynikające ze stosowania operacji NAT, na które narażone są inne aplikacje internetowe. Rozwiązanie nie zyskało popularności również ze względu na problemy ze skalowaniem [RFC6343]. I choć opracowano stosowne metody eliminacji ograniczeń (takie jak ICE; patrz rozdział 7.), mechanizm ten został zastąpiony przez specjalny protokół o nazwie **Teredo** (określany początkowo jako „shipworm" [świdrak okrętowiec], ale z uwagi na skojarzenia z robakiem internetowym [*worm*] nazwany ostatecznie Teredo; nazwę oparto na łacińskiej nazwie tego małża) [RFC4380] [RFC5991] [RFC6081]. Rozwiązanie cieszy się dużą popularnością przede wszystkim dlatego, że zostało włączone do najnowszych wersji systemu Microsoft Windows.

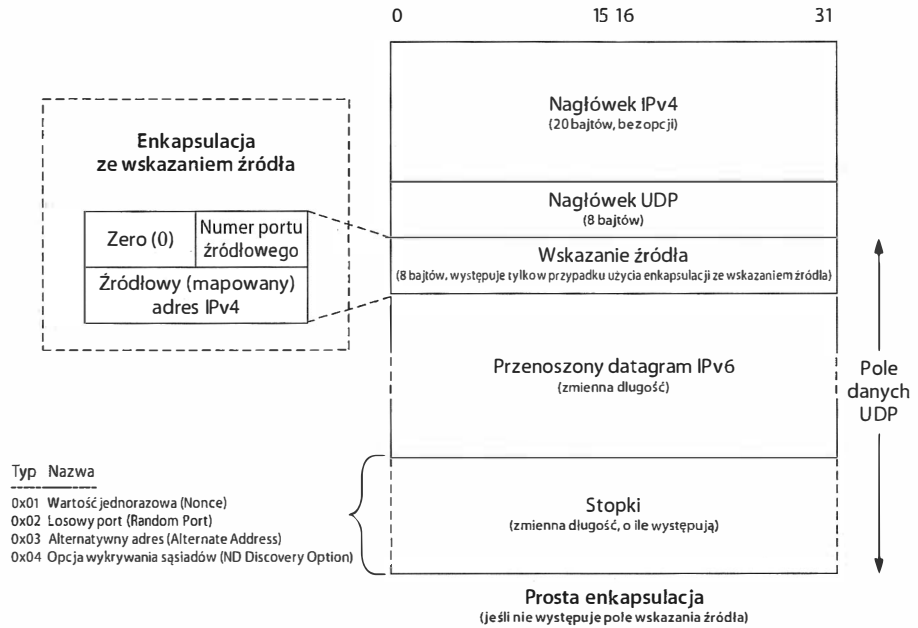
Mechanizm Teredo (nazywany również **tunelowaniem Teredo**) transportuje datagramy IPv6 w polu danych segmentów UDP/IPv4, co umożliwia wykorzystanie systemów, które nie są obsługują połączeń IPv6. Technika ta została przedstawiona na rysunku 10.5. **Klientami** Teredo są stacje obsługujące protokoły IPv4 i IPv6 z uruchomionym interfejsem tunelowania Teredo. Interfejsom tunelowania przypisywany jest specjalny adres o prefiksie IPv6 2001::/32, który podlega „procedurze kwalifikacyjnej" opisanej w następnym akapicie. **Serwery** Teredo pełnią funkcję zbliżoną do serwerów STUN (patrz rozdział 7.). Umożliwiają ustanawianie bezpośrednich tuneli dla odpowiednio enkapsulowanych pakietów IPv6 w sieciach z jednostkami NAT. **Przekaźniki** Teredo (*Teredo relay*) realizują zadania analogiczne do serwerów TURN. Dlatego podczas przenoszenia ruchu wielu użytkowników sieci mogą potrzebować znacznej mocy obliczeniowej. Serwery muszą dysponować wszystkimi funkcjami przekaźników. Odwrotna zależność nie jest jednak prawdziwa. Wykorzystanie przekaźnika Teredo w komunikacji IPv6 jest operacją „ostatniej szansy". Jednostki sieciowe unikają tunelowania Teredo, jeśli istnieją inne możliwości dostarczenia pakietów IPv6 (np. transmisja bezpośrednia lub skorzystanie z systemu 6to4).



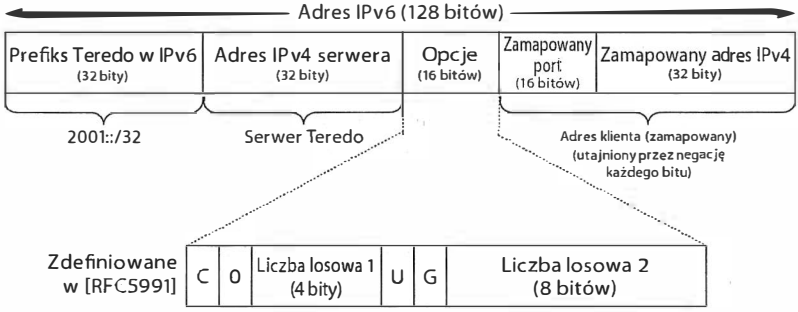
Rysunek 10.5. *System przekazywania ruchu IPv6 w sieciach IPv4 Teredo zapisuje datagramy IPv6 (wraz z opcjonalnymi stopkami) w polu danych segmentów UDP/IPv4. Serwer Teredo umożliwia klientom uzyskanie adresu IPv6 po określeniu sposobu mapowania adresów i numerów portów. Przekaźniki Teredo zapewniają przekazywanie ruchu między klientami Teredo, 6to4 oraz standardowymi jednostkami IPv6*

Działanie systemu przedstawionego na rysunku 10.5 wymaga przypisania klientowi Teredo nazwy lub adresu IPv4 serwera Teredo oraz numeru portu UDP tego serwera (zazwyczaj 3544). Ponieważ system ten został opracowany przez firmę Microsoft, jeden z serwerów Teredo został udostępniony pod adresem *teredo.ipv6.microsoft.com*. Gdy klient jest gotowy do pobrania adresu, rozpoczyna **procedurę kwalifikacyjną**. Wysyła pakiety ICMPv6 RS (więcej informacji na ich temat znajduje się w rozdziale 8.) z jednego z adresów IPv6 łącza lokalnego (*link-local*), wykorzystując port usługi Teredo (agenta odpowiedzialnego za pakowanie pakietów IPv6 w segmenty UDP/IPv4 oraz ich rozpakowywanie). Format enkapsulacji odpowiada enkapsulacji ze wskazaniem źródła (*origin indication*), przedstawionej na rysunku 10.6.

Odpowiedziami na te pakiet są komunikaty ICMPv6 RA, które również odpowiadają enkapsulacji ze wskazaniem źródła (przedstawionej na rysunku 10.6). Odpowiedź RA zawiera opcję informacji o prefiksie z odpowiednim prefiksem Teredo (więcej informacji na ten temat znajduje się w rozdziale 2.). Format ze wskazaniem źródła dostarcza klientowi informacji na temat własnych mapowanych wartości adresu i portu. Źródłowy adres komunikatu RA jest użytecznym adresem IPv6 łącza lokalnego serwera. Adresem docelowym jest adres IPv6 łącza lokalnego klienta, występujący w komunikacie RS. Jeśli operacja zostanie zrealizowana poprawnie, procedura kwalifikacji się kończy i klient może utworzyć adres IPv6 mechanizmu Teredo, posługując się prefiksem oraz informacjami na temat źródła dostarczonymi przez serwer. Adres Teredo składa się z kilku parametrów. Jego struktura została przedstawiona na rysunku 10.7.



Rysunek 10.6. *Formaty prostej enkapsulacji oraz enkapsulacji ze wskazaniem źródła wykorzystywane w systemie Teredo. Format enkapsulacji ze wskazaniem źródła zakłada wstawienie pomiędzy nagłówek UDP i przenoszony datagram IPv6 adresu IPv4 oraz numeru portu. Dane te są wykorzystywane przez stacje klienckie Teredo podczas formowania adresu IPv6 (odpowiadają mapowanym adresom i numerom portów). Adresy i numery portów są „ukrywane" przez negowanie kolejnych bitów w celu uniemożliwienia usłudze NAT wprowadzania zmian w ich treści. Możliwe jest dołączanie dodatkowych stopek w formacie TLV, które odpowiadają za definiowanie różnych rozszerzeń mechanizmu Teredo (np. obsługi symetrycznej translacji NAT)*



Rysunek 10.7. *W jednostkach klienckich Teredo używa się adresów IPv6 o prefiksie 2001::/32. Kolejne bity odzwierciedlają adres IPv4 serwera Teredo i rodzaj usługi NAT (znacznik 16-bitowy). Po nich następuje ciąg losowych bitów utrudniających przeprowadzenie ataku z odgadywaniem adresu. Ostatnia część to 16-bitowy numer portu klienckiego oraz 32-bitowy adres IPv4 stacji klienckiej. Wartości te podlegają „utajnieniu"*

Adres Teredo (przedstawiony na rysunku 10.7) składa się z prefiksu Teredo (2001::/32), adresu IPv4 serwera Teredo, 16-bitowego pola *Opcji* opisanych w kolejnym akapicie oraz mapowanego numeru portu oraz mapowanego adresu IPv4. Dwie ostatnie wartości są informacjami o adresie klienta rejestrowanymi przez serwer Teredo i są zazwyczaj wyznaczane przez ostatnią operację NAT po stronie klienckiej. Bity adresu i numeru portu są negowane, aby uniemożliwić przepisanie ich przez jednostkę NAT.

Pole *Opcje* (16-bitowe) jest wykorzystywane do informowania stacji o rodzaju operacji NAT wykonanej podczas procedury kwalifikacyjnej. Niektóre usługi NAT (nazywane niegdyś symetrycznymi usługami NAT — definiującymi mapowanie na podstawie adresów lub adresów i numerów portów wraz z filtrowaniem na podstawie adresów lub adresów i numerów portów) współdziałają z systemem Teredo tylko w przypadku uwzględnienia rozszerzeń Teredo (opisanych w dalszej części podrozdziału). Jednak w większości sieci domowych wykonywane są operacje NAT (określane jako „NAT stożkowy" — *cone NAT* — w których mapowanie i filtrowanie jest niezależne od punktów końcowych) działające bez wspomnianych rozszerzeń. Pierwotnie bit *C* informował o wykonaniu operacji NAT w wersji *cone NAT*. Jednak obecnie jego użycie jest uznawane za niepotrzebne i zaleca się ustawianie wartości 0 (stacje klienckie ignorują tę wartość, natomiast serwery wykrywają na jej podstawie jednostki starszego typu). Kolejny bit jest na stałe ustawiony na 0. Bity *U* (uniwersalny) i *G* (grupy) są przeznaczonego do przyszłego wykorzystania i obecnie również są ustawiane na zero. Pola *Liczba losowa 1* i *Liczba losowa 2* przechowują wartości losowe wygenerowane zgodnie z zaleceniem [RFC5991]. Ich zadanie polega na utrudnieniu odgadnięcia adresu (a tym samym wyeliminowaniu ataków polegających na generowaniu pakietów z losowymi wartościami adresu).

Wyznaczenie adresu Teredo umożliwia zakwalifikowanemu klientowi rozpoczęcie generowania ruchu IPv6. Skutki niepoprawnego zakończenia procedury kwalifikacji oraz zasady bezpiecznego kwalifikowania klientów zostały opisane w dokumencie [RFC4380]. Ogólnie rzecz ujmując, klient Teredo może komunikować się z innym klientem przyłączonym do tego samego łącza, z innym klientem działającym w Internecie IPv4 lub stacją funkcjonującą w Internecie IPv6. W każdym przypadku zapewnia dla mechanizmu IPv6 ND alternatywę bazującą na protokołach UDP/IPv4. W komunikacji z klientami przyłączonymi do tego samego łącza używany jest protokół wykrywania stacji wykonujący multiemisję IPv4. Jego adres to 224.0.0.253. Do ustalenia, czy stacje docelowe przyłączone są do tego samego łącza, wykorzystuje się specjalne „pakiety-bańki" (*bubble*) (pakiety pozbawione danych w polu danych). Są to pakiety IPv6 o najmniejszym dopuszczalnym rozmiarze sformatowane zgodnie z zasadami prostej enkapsulacji, przedstawionymi na rysunku 10.6. Występujące w nagłówku IPv6 pole *Docelowy adres IP* odpowiada stacji, z którą prowadzona jest komunikacja. W pakiecie nie są natomiast zapisane żadne dane oraz rozszerzenia (pole *Następny nagłówek* ma wartość 0x3b oznaczającą brak kolejnego nagłówka). W przypadku klientów przyłączonych do Internetu IPv4 wymiana danych wymaga zastosowania adresów IPv6 Teredo, które odzwierciedlają adres IPv4 i numer portu. Klient nie ma więc problemów z wysłaniem pakietu Teredo do zdalnej jednostki NAT. Współdziałanie z restrykcyjnymi jednostkami NAT wymaga od mechanizmu Teredo wysłania pakietu-bańki, który „przebije" się przez NAT i spowoduje wyznaczenie mapowania NAT dla protokołu UDP (więcej informacji na ten temat znajduje się w rozdziale 7. i w dokumencie [RFC6081]).

Zanim zakwalifikowany klient wyśle pakiet przeznaczony do jednostki IPv6 (tj. do jednostki, która nie używa adresu Teredo), musi sprawdzić, czy dysponuje informacjami na temat przekaźnika pośredniczącego w transmisji ze stacją docelową. Jeśli tak, pakiet jest wysyłany w formacie opisanym jako prosta enkapsulacja. Jeśli nie, przygotowuje żądanie ICMPv6 Echo Request zawierające dużą liczbę losową (np. 64-bitową) i wysyła je do jednostki docelowej IPv6 przez serwer Teredo. Serwer przekazuje pakiet do stacji docelowej. Jednostka odbierająca żądanie odczytuje z nadchodzącego datagramu źródłowy adres IPv6, który odpowiada adresowi Teredo klienta. Przygotowuje wówczas odpowiedź Echo Reply, którą dostarcza do najbliższego przekaźnika Teredo. Zadanie przekaźnika polega natomiast na przesłaniu odpowiedzi do wskazanego klienta. Stacja odbiorcza analizuje adres IPv4 przekaźnika i zapisuje w pamięci podręcznej. Dzięki temu kolejne pakiety adresowane do określonej stacji IPv6 może dostarczać do przekaźnika, którego adres został wcześniej ustalony.

Zgodnie ze specyfikacją [RFC6081] mechanizm Teredo obsługuje wiele opcjonalnych rozszerzeń, z których część umożliwia pracę z zastosowaniem symetrycznych operacji NAT. Wspomniane rozszerzenia odpowiadają za zmianę domyślnego sposobu działania protokołu i są opisywane jako: obsługa symetrycznego NAT-u (SNS — *Symmetric NAT Support*), symetryczny NAT z obsługą UPnP (UP — *UPnP-Enabled Symmetric NAT*), symetryczny NAT z rezerwacją portów (PP — *Port-Preserving Symmetric NAT*), symetryczny NAT z użyciem kolejnych portów (SP — *Sequential Port-Symmetric NAT*), hairpinning (HP) oraz zmniejszenie obciążenia serwera (SLR — *Server Load Reduction*). Większość rozszerzeń może być wykorzystywana niezależnie. Wyjątek stanowią mechanizmy UP i PP, które zależą od rozszerzenia SNS. Poszczególne odmiany usługi NAT, które mogą być wykorzystywane w połączeniu z wymienionymi rozszerzeniami, zostały opisane w tabeli zawartej w sekcji 3. dokumentu [RFC6081]).

Działanie rozszerzeń wymaga dołączenia do komunikatu Teredo co najmniej jednej **stopki**. Zapis informacji ma formę tripletów TLV o składni podobnej do opcji ICMPv6 ND (przedstawionej na rysunku 8.41). Każdy wpis składa się z 8-bitowego pola *Typu* oraz 8-bitowego pola *Długości*. Dwa najbardziej znaczące bity pola *Typ* opisują sposób przetwarzania stopki w przypadku, w którym jednostka docelowa nie obsługuje danego rozszerzenia. Kombinacja 01 nakazuje odrzucenie pakietu. Wszystkie pozostałe wartości informują, że niezrozumiałą stopkę należy pominąć i przystąpić do analizowania kolejnej, zgodnie z porządkiem przetwarzania. Zatwierdzona lista typów wartości znajduje się w dokumencie [TTYPES] sporządzonym przez organizację IANA. Definicje obowiązujące w czasie pisania książki zostały przedstawione w tabeli 10.1.

Stopka Nonce przenosi 32-bitową wartość losową, która jest zmieniana w każdym komunikacie. Stanowi więc zabezpieczenie przed powtórnym dostarczeniem tego samego pakietu (więcej informacji na temat ataków z powtarzaniem pakietów zostało zamieszczonych w rozdziale 18.). Rozwiązanie to jest wykorzystywane do zabezpieczania par HP lub SNS (adresów IPv4 i numerów portów). Każda para zajmuje 6 bajtów, a pojedyncza stopka może pomieścić maksymalnie cztery takie pary. Zadaniem par jest identyfikowanie stacji końcowych UDP/IPv4, które mogą być wykorzystywane w kontakcie z serwerem przez różne jednostki klienckie Teredo pracujące po tej samej stronie jednostki NAT. Mechanizm ten jest stosowany w rozszerzeniu HP.

Tabela 10.1. *Stopki Teredo są zapisywane z polem danych IPv6 w datagramach UDP/IPv4. Definicja każdej z nich obejmuje informację o typie wartości, nazwie oraz zakresie zastosowania. W niektórych rozwiązaniach rozmiar pola wartości jest stały*

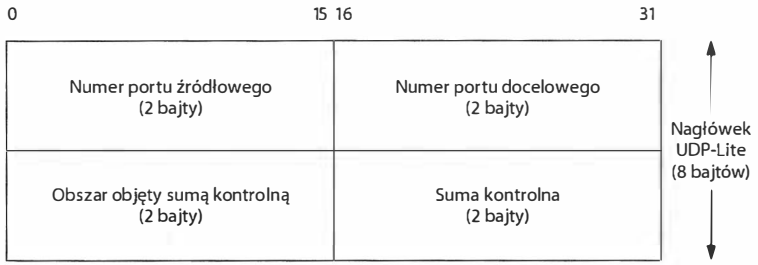
Typ	Długość	Nazwa	Użycie	Uwagi
0x00	Zarezerwowana	Niezdefiniowana	Niezdefiniowane	Niezdefiniowane
0x01	0x04	Nonce	SNS, UP, PP, SP, HP	32-bitowa wartość jednorazowa zabezpieczająca pakiet przed ponownym przesłaniem (patrz rozdział 18.).
0x02	Zarezerwowana	Niezdefiniowana	Niezdefiniowane	Niezdefiniowane
0x03	[8, 26]	Alternate Address	HP	Dodatkowa wartość adresu i (lub) portu wykorzystywana przez jednostki klienckie Teredo przesłonięte przez tę samą jednostkę NAT.
0x04	0x04	ND Option	SLR	Umożliwia odświeżenie mapowania NAT przez bezpośrednie przesłanie pakietu-bańki (przenoszącego komunikat NS).
0x05	0x02	Random Port	PP	Mapowany port nadawcy.

Stopka ND Option przenosi jeden bajt, który odpowiada komunikatom TeredoDiscovery Solicitation (0x00) lub TeredoDiscoveryAdvertisement (0x01). Pierwszy z wymienionych komunikatów wymusza na odbiorcy bezpośrednie przesłanie pakietu-bańki (wymusza bezpośrednią komunikację między klientami Teredo) odpowiadającego drugiemu formatowi wiadomości. Pakiet TeredoDiscoveryAdvertisement jest odpowiedzią na to żądanie. Opisywana stopka jest używana w rozszerzeniach SLR, które pozwalają na wykorzystanie komunikatów NS/NA transportowanych w pakietach-bańkach do odświeżania stanu usługi NAT, bez potrzeby przesyłania analogicznych pakietów za pośrednictwem serwerów. Ostatnia ze stopek — Random Port — przenosi 16-bitowy numer portu UDP. Nadawca stopki zgaduje, jaki port zostanie mu przydzielony przez usługę NAT i informuje o tym fakcie jednostkę docelową. Mechanizm ten jest używany w rozszerzeniu PP (patrz sekcja 6.3 dokumentu [RFC6081]).

10.6. UDP-Lite

Niektóre aplikacje nie są wrażliwe na występowanie błędów w wysyłanych lub odbieranych informacjach. Często bazują więc na protokole UDP, który pozwala na wyeliminowanie dodatkowego narzutu transmisyjnego związanego z ustanawianiem połączenia oraz może być wykorzystywany w komunikacji rozgłoszeniowej i multiemisji. Niestety, standard UDP umożliwia jedynie stosowanie sum kontrolnych odnoszących się do całego pola danych lub do pominięcia wartości kontrolnej w całości. Protokół o nazwie **UDP-Lite** (lub **UDPLite**) [RFC3828] jest rozwiązaniem tego problemu. Wykorzystuje się w nim zmodyfikowaną wartość kontrolną, która może odnosić się tylko do części pola danych

datagramu UDP. Protokołowi UDP-Lite przypisano oddzielne wartości pól *Protokół* IPv4 oraz *Następny Nagłówek* IPv6 (136). Należy więc traktować go jak niezależny protokół transportowy. W mechanizmie UDP-Lite nadmiarowe pole *Długość* nagłówka UDP zostało zastąpione polem *Obszar objęty sumą kontrolną* (*Checksum Covarage*), zgodnie z rysunkiem 10.8.



Rysunek 10.8. *Protokół UDP-Lite uwzględnia pole obszaru objętego sumą kontrolną, które przechowuje liczbę bajtów (liczonych od pierwszego bajta nagłówka UDP) uwzględnionych w wartości kontrolnej. Minimalna wartość parametru wynosi 0 i oznacza, że cały datagram podlega weryfikacji. Wartości od 1 do 7 są niedozwolone, ponieważ nagłówek segmentu zawsze jest objęty sumą kontrolną. Protokołowi UDP-Lite przypisano numer 136 (odmienny od wartości 17 charakterystycznej dla mechanizmu UDP). Ta sama wartość identyfikatora jest stosowana w polu Następny nagłówek protokołu IPv6*

Przedstawione na rysunku 10.8 pole *Obszar objęty sumą kontrolną* informuje o liczbie bajtów (liczonych od pierwszego bajta nagłówka UDP), które są uwzględniane w wartości kontrolnej. Najmniejsza dopuszczalna wartość pola wynosi 8 (poza szczególna wartością, jaką jest 0), ponieważ sam nagłówek UDP-Lite zawsze musi być objęty działaniem mechanizmów kontrolnych. Wartość 0 informuje o obowiązku weryfikacji sumy kontrolnej w odniesieniu do całego segmentu, podobnie jak w tradycyjnym protokole UDP. Z uwagi na ograniczenie długości pola danych, wynikające z rozmiaru pola *Obszar objęty sumą kontrolną*, mechanizm nie zawsze sprawdza się w przetwarzaniu jumbogramów, bo maksymalna liczba bajtów uwzględnianych w sumie kontrolnej nie może przekroczyć 64 k, chyba że weryfikacja dotyczy całego datagramu (tzn. w polu nagłówka jest wówczas zapisana wartość 0). Do obsługi protokołu UDP-Lite służy specjalna opcja interfejsu API gniazd (`IPPROTO_UDPLITE`), podobnie jak do określania obszaru objętego weryfikacją (są to opcje `SOL_UDPLITE`, `UDPLITE_SEND_CSCOV` oraz `UDPLITE_RECV_CSCOV` funkcji `setsockopt`).

10.7. Fragmentacja

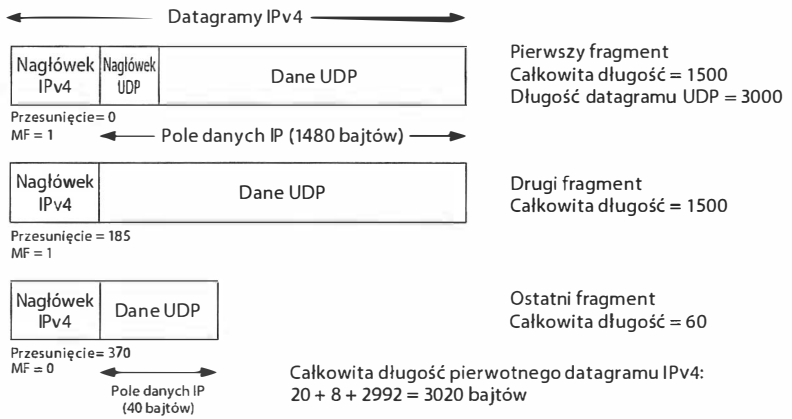
Zgodnie z informacjami zawartymi w rozdziale 3. mechanizmy ramkowania działające na poziomie warstwy łącza danych nakładają pewne ograniczenie na rozmiar wysyłanych ramek. Dlatego zagwarantowanie niezależności datagramów IP od szczegółów implementacyjnych modułu warstwy łącza danych wymaga zastosowania **fragmentacji** i **odtwarzania** pakietów IP. Za każdym razem, gdy warstwa IP otrzymuje datagram przeznaczony do wysłania, wyznacza lokalny interfejs, za pomocą którego pakiet zostanie wyemitowany (dzięki analizie tablicy przekazywania; patrz rozdział 5.), i ustala wartość parametru MTU. Jeśli rozmiar pakietu jest większy od wartości MTU, konieczne jest wykonanie fragmentacji. Zadanie to może zostać zrealizowane zarówno w stacji nadawczej, jak i w każdym węźle pośredniczącym w transmisji danych. Oznacza to, że fragmenty

datagramu mogą same podlegać fragmentacji. W połączeniach IPv6 mechanizm ten działa nieco inaczej, ponieważ operacja fragmentacji może być wykonywana jedynie w stacji źródłowej. Przykład takiego postępowania został przedstawiony w rozdziale 5.

Odtworzenie podzielonego pakietu IP następuje dopiero w jednostce docelowej. Wynika to z dwóch względów, z których drugi jest bardziej przekonujący niż pierwszy. Po pierwsze, brak obowiązku odtwarzania pakietów na trasie eliminuje konieczność implementowania tej funkcji w oprogramowaniu (lub sprzęcie) routerów. Po drugie, poszczególne fragmenty datagramu mogą być przesyłane do wskazanego miejsca docelowego różnymi trasami. Wynika z tego, że żaden z routerów uczestniczących w przekazywaniu pakietów nie może odtworzyć pierwotnego datagramu, ponieważ przetwarza jedynie pewien podzbiór wszystkich fragmentów. Biorąc pod uwagę wydajność dzisiejszych routerów, pierwszy z wymienionych powodów nie jest szczególnie przekonujący, tym bardziej, że większość urządzeń tego typu pełni jednocześnie funkcje jednostek końcowych w różnych formach komunikacji (np. podczas konfigurowania). Niemniej drugi z wymienionych powodów rozwiewa wszelkie wątpliwości.

10.7.1. Przykład — fragmentacja datagramów UDP/IPv4

Jeśli konieczne jest uniknięcie fragmentacji na poziomie warstwy IP, aplikacja korzystająca z protokołu UDP musi być wyposażona w odpowiednie mechanizmy zabezpieczające. Do podziału pakietu dochodzi wtedy, gdy rozmiar wynikowego datagramu jest większy od wartości MTU danego łącza. Działanie to może z kolei negatywnie wpłynąć na wydajność komunikacji, ponieważ *utrata jednego z fragmentów jest równoznaczna z utratą całego datagramu*. Na rysunku 10.9 przedstawiono przypadek podziału datagramu o rozmiarze 3020 bajtów na kilka pakietów IPv4.



Rysunek 10.9. *Pojedynczy datagram UDP o 2992 bajtach zapisanych w polu danych UDP jest dzielony na trzy pakiety UDP/IPv4 (bez opcji). Nagłówek UDP (zawierający numery źródłowego i docelowego portu) znajduje się jedynie w pierwszym fragmencie (co znacznie utrudnia pracę zapór sieciowych i usług NAT). Do obsługi mechanizmu fragmentacji służą pola Identyfikacja, Przesunięcie fragmentu oraz znacznik nagłówka IPv4 o nazwie MF (More Fragments) oznaczający „więcej fragmentów"*

Z rysunku 10.9 wynika, że datagram UDP przenosi 2992 bajty danych aplikacji (w polu danych UDP) oraz 8 bajtów nagłówka. W rezultacie pole *Całkowita długość* nagłówka IPv4 ma wartość 3020 bajtów (ponieważ nagłówek IPv4 sam ma rozmiar 20 bajtów). Podzielenie datagramu na trzy pakiety spowodowało dodanie 40 bajtów (po 20 bajtów nagłówka każdego nowo utworzonego pakietu). Zatem całkowita liczba przesłanych bajtów zwiększyła się do 3060, czyli o 1,3%. Wartość pola *Identyfikacja* (definiowana przez pierwotnego nadawcę) jest kopiowana do każdego z generowanych fragmentów i stanowi wyróżnik pozwalający na scalenie datagramu po stronie odbiorczej. Pole *Przesunięcie fragmentu* wyznacza przesunięcie pierwszego bajta pola danych względem początku oryginalnego datagramu IPv4 (wartość jest wyrażona w jednostkach odpowiadających ośmiu bajtom). Pierwszy fragment ma więc zerowe przesunięcie. Wartość pola w drugim pakiecie wynosi 185 (185 * 8 = 1480). Liczba 1480 odpowiada rozmiarowi pierwszego fragmentu pomniejszonemu o rozmiar nagłówka IPv4. Ta sama zasada dotyczy trzeciego fragmentu. Bit MF informuje o tym, czy należy się spodziewać kolejnych fragmentów. Pakiet ostatniego fragmentu ma ten bit ustawiony na zero. Odbiór bitu MF o wartości 0 oznacza, że proces odtwarzania pakietu może obliczyć rozmiar pierwotnego datagramu jako sumę wartości *Przesunięcie fragmentu* (pomnożoną przez 8) oraz *Całkowita długość* (pomniejszoną o rozmiar nagłówka IPv4). Ponieważ pole *Przesunięcie fragmentu* jest wyznaczane względem pierwotnego datagramu, proces odtwarzania datagramu może prawidłowo obsłużyć również pakiety docierające w niewłaściwej kolejności. Podział datagramu na fragmenty wiąże się jednocześnie z zapisaniem w polu *Całkowita długość* każdego z fragmentów jedynie rozmiaru danego fragmentu.

Choć operacja fragmentacji wydaje się nieskomplikowana, jedna z jej cech sprawia, że bywa bardzo kłopotliwa w praktyce — jeśli jeden z fragmentów zostanie utracony, tracony jest cały datagram. Przyczyną jest brak mechanizmów kontrolnych na poziomie samego protokołu IP. Rozwiązania, takie jak wyznaczenie czasu oczekiwania i retransmisje, są domeną wyższych warstw stosu protokołów (na poziomie warstwy transportowej oczekiwanie i retransmisje są wykorzystywane jedynie w protokole TCP; w UDP nie ma takich mechanizmów). W przypadku utraty fragmentu segmentu TCP moduł TCP retransmituje cały segment, który odpowiada pojedynczemu datagramowi IP. Nie ma jednak możliwości retransmitowania fragmentu datagramu. Fragmentacja może np. zostać wykonana przez jeden z routerów pośredniczących w transporcie danych. Stacja końcowa nie ma jednak możliwości stwierdzenia, która jednostka wykonała podział datagramu. Z tego powodu fragmentacja jest operacją, której należy unikać. Argumenty za tym zostały przedstawione w dokumencie [KM87].

Doprowadzanie do podziału datagramu IP przy użyciu protokołu UDP nie stanowi większego wyzwania (w dalszej części książki opisano przykład świadczący o tym, że wygenerowanie segmentu TCP o dostatecznie dużym rozmiarze, by doprowadził do fragmentacji, jest wyjątkowo trudnym zadaniem). Wystarczy posłużyć się programem sock i zwiększać stopniowo rozmiar datagramu, aż do wystąpienia jego podziału. W sieciach Ethernet maksymalny rozmiar pola danych wynosi zazwyczaj 1500 bajtów (więcej informacji na ten temat znajduje się w rozdziale 3.). Oznacza to, że przekazanie z aplikacji bloku o rozmiarze nie większym niż 1472 pozwala uniknąć fragmentacji (przy założeniu, że rozmiar nagłówka IPv4 to 20 bajtów, a rozmiar nagłówka UDP to 8 bajtów)[1]. W poniższym przykła-

[1] Założenie jest poprawne, jeśli w nagłówku IPv4 nie zostały zdefiniowane żadne opcje. Dodanie opcji może spowodować rozszerzenie nagłówka IPv4 z 20 bajtów nawet do 60 bajtów.

dzie program sock został uruchomiony z wykorzystaniem bloków danych o rozmiarach 1471, 1472, 1473 oraz 1474 bajtów. Należy się więc spodziewać przynajmniej dwóch przypadków fragmentacji:

```
Linux% sock -u -i -n1 -w1471 10.0.0.3 discard
Linux% sock -u -i -n1 -w1472 10.0.0.3 discard
Linux% sock -u -i -n1 -w1473 10.0.0.3 discard
Linux% sock -u -i -n1 -w1474 10.0.0.3 discard
```

Na listingu 10.3 przedstawiono wynik zarejestrowany przez program tcpdump (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.3. *Fragmentacja datagramów UDP w łączu ethernetowym o parametrze MTU wynoszącym 1500 bajtów*

```
1 23:42:43.562452 10.0.0.5.46530 > 10.0.0.3.9:
                  udp 1471 (DF) (ttl 64, id 61350, len 1499)
2 23:42:50.267424 10.0.0.5.46531 > 10.0.0.3.9:
                  udp 1472 (DF) (ttl 64, id 62020, len 1500)
3 23:42:57.814555 10.0.0.5 > 10.0.0.3:
                  udp (frag 37671:1@1480) (ttl 64, len 21)
4 23:42:57.814715 10.0.0.5.46532 > 10.0.0.3.9:
                  udp 1473 (frag 37671:1480@0+) (ttl 64, len 1500)
5 23:43:04.368677 10.0.0.5 > 10.0.0.3:
                  udp (frag 37672:2@1480) (ttl 64, len 22)
6 23:43:04.368838 10.0.0.5.46535 > 10.0.0.3.9:
                  udp 1474 (frag 37672:1480@0+) (ttl 64, len 1500)
```

Pierwsze dwa z zarejestrowanych datagramów UDP (pakiety 1. i 2.) mieszczą się w pojedynczych ramkach ethernetowych (o standardowym formacie DIX [ethernetowym]) bez konieczności podziału. W trzecim przypadku rozmiar datagramu IPv4 wynosi 1501 bajtów (co odpowiada 1473 bajtom dostarczonym z warstwy aplikacji), a to oznacza, że trzeba wykonać fragmentację datagramu (pakiety 3. i 4.). Podobnie datagram powstający po zapisie 1474 bajtów danych ma rozmiar 1502 bajtów, więc jego transmisja jest możliwa dopiero po podziale na dwa fragmenty (pakiety 5. i 6.).

Polecenie tcpdump, wyświetlając informacje na temat przechwyconych datagramów, prezentuje kilka interesujących danych. Po pierwsze, elementy frag 37671 (pakiety 3. i 4.) oraz frag 37672 (pakiety 5. i 6.) odpowiadają wartościom pól *Identyfikacja* zapisanym w nagłówkach IPv4. Następna liczba w części dotyczącej fragmentacji (wartość między dwukropkiem a znakiem @ w pakietach 4. i 6.) odzwierciedla rozmiar pakietu IPv4 po usunięciu nagłówka IPv4. Pierwszy fragment obu datagramów zawiera 1480 bajtów danych (8 bajtów nagłówka UDP oraz 1472 bajty danych użytkownika; po dodaniu 20 bajtów pozbawionego opcji nagłówka IPv4 otrzymujemy pakiet o dokładnie 1500 bajtach). Drugi fragment pierwszego podzielonego datagramu (pakiet 3.) składa się z jednego bajta (jest to pozostały bajt użytkownika). Natomiast drugi fragment drugiego z dzielonych datagramów przenosi 2 bajty danych. Fragmentacja zakłada operowanie danymi w jednostkach odpowiadających 8 bajtom. Ograniczenia to nie dotyczy jedynie ostatniego fragmentu. W omawianym przykładzie przetwarzane są bloki o rozmiarze 1480 bajtów, ponieważ liczba ta jest wielokrotnością liczby 8 (warto porównać ten przykład z przykładem przedstawionym w rozdziale 5., z którego wynika, że parametr MTU o wartości 1500 bajtów nie pozwala na optymalne wykorzystanie fragmentacji IPv6 w sieciach Ethernet).

Liczba zapisana za znakiem @ odpowiada przesunięciu danych fragmentu względem początku datagramu. Pierwszy fragment każdego kolejnego datagramu ma zerowe przesunięcie (pakiety 4. i 6.). Natomiast kolejne fragmenty datagramów są przesunięte o 1480 bajtów (pakiety 3. i 5.). Znak + występujący za wartością przesunięcia oznacza, że istnieją kolejne fragmenty składowe danego datagramu. Znak ten odzwierciedla więc wartość bitu *MF* trzybitowego pola *Opcji* w nagłówku IPv4.

W przeprowadzonym doświadczeniu zaskakujący może być fakt dostarczenia w pierwszej kolejności fragmentów o większej wartości przesunięcia — **przed** pierwszymi fragmentami. Zamiana kolejności jest celowym działaniem nadawcy, ponieważ taki sposób jest korzystniejszy dla odbiorcy. Dostarczenie najpierw ostatniego fragmentu pozwala zaalokować po stronie zdalnej odpowiedni bufor na odtworzenie całego datagramu. Jeśli kod odtwarzania datagramów jest dostatecznie wydajny, zamiana kolejności nie stanowi dla niego żadnego problemu. Oczywiście, istnieją również rozwiązania, które działałyby efektywniej, gdyby informacje na temat wyższych warstw stosu (np. numery portów UDP) były dostarczane na początku [KEWG96].

Pakiety 3. i 5. (kolejne fragmenty datagramu) są pozbawione numerów portów UDP (źródłowego i docelowego). Aby narzędzie `tcpdump` mogło wyświetlić wspomniane numery, konieczne było odtworzenie podzielonego datagramu (numery portów są zapisane w nagłówku UDP zapisanym w pierwszym fragmencie).

10.7.2. Maksymalny czas odtwarzania datagramu

W chwili odebrania pierwszego fragmentu datagramu moduł IP uruchamia zegar odtwarzania datagramu. Zabezpieczenie czasowe jest konieczne, ponieważ niedostarczenie kolejnych fragmentów mogłoby spowodować wyczerpanie pamięci przeznaczonej do buforowania pakietów (zgodnie z listingiem 10.4), a tym samym mogłoby ułatwić przeprowadzenie ataku na daną jednostkę sieciową. Kolejny przykład został przygotowany za pomocą specjalnego programu, który formuje i wysyła jedynie dwa pierwsze fragmenty komunikatu ICMPv4 Echo Request i przerywa swoje działanie. Kolejne fragmenty nie są generowane. Reakcja stacji odbiorczej została przedstawiona na listingu 10.4 (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.4. *Przekroczenie czasu odtwarzania datagramu IPv4*

```
1 17:35:59.609387 10.0.0.5 > 10.0.0.3:
     icmp: echo request (frag 28519:380@0+) (ttl 255, len 400)
2 17:36:19.617272 10.0.0.5 > 10.0.0.3:
     icmp (frag 28519:380@376+) (ttl 255, len 400)
3 17:36:29.602373 10.0.0.3 > 10.0.0.5:
     icmp: ip reassembly time exceeded for 10.0.0.5 > 10.0.0.3:
           icmp: echo request (frag 28519:380@0+) (ttl 255, len 400)
           [tos 0xc0](ttl 64, id 38816, len 424)
```

Z analizy listingu wynika, że pierwszy fragment (w czasie i w kolejności generowania) ma całkowitą długość 400 bajtów. Drugi fragment został dostarczony 20 s po pierwszym. Natomiast ostatni fragment nie został nigdy dostarczony do stacji odbiorczej. Po upływie 30 s od momentu dostarczenia pierwszego fragmentu system docelowy wygenerował komunikat ICMPv4 Time Exceeded (przekroczenie czasu; kod 1), informując nadawcę,

że datagram został odrzucony, i dołączając kopię pierwszego fragmentu. Typowa wartość czasu odtwarzania wynosi 30 lub 60 s. Jak wynika z doświadczenia, zegar jest uruchamiany po odebraniu pierwszego fragmentu i nie jest zerowany wraz z odbiorem kolejnych porcji danych. Wyznacza więc pewien dopuszczalny przedział czasowy emisji kolejnych fragmentów jednego datagramu.

W przeszłości większość implementacji modułu IP bazujących na berkeleyowskich rozwiązaniach uniksowych nigdy nie generowała tego typu błędów. Mimo że zegar *był uruchamiany* i pakiety *były odrzucane* po upływie wyznaczonego czasu, systemy odbiorcze nie odsyłały komunikatu ICMP. Ponadto niektóre implementacje opisywanego mechanizmu nie generują komunikatu ICMP, jeśli nie zostanie odebrany *pierwszy fragment* (tj. ten z polem *Przesunięcie fragmentu* o wartości 0). Powodem jest to, że odbiorca komunikatu o błędzie nie jest w stanie określić, który z procesów użytkownika wysłał odrzucony datagram (z uwagi na brak nagłówka warstwy transportowej). Zakłada się natomiast, że protokoły wyższych warstw ostatecznie zarejestrują upływ dopuszczalnego czasu reakcji i zapewnią wykonanie retransmisji, jeśli będzie niezbędna.

10.8. Ustalanie parametru MTU trasy w protokole UDP

Przeanalizujmy interakcje między aplikacją korzystającą z protokołu UDP a mechanizmem ustalania wartości MTU na trasie transportu danych (PMTUD — *Path MTU Discovery Mechanism*) [RFC1191]. W protokołach takich jak UDP, w których aplikacja wywołująca zazwyczaj ma wpływ na rozmiar generowanego datagramu, możliwość wyznaczenia odpowiedniego rozmiaru datagramu bywa bardzo użyteczna, ponieważ pozwala na uniknięcie fragmentacji. Standardowy mechanizm PMTUD bazuje na komunikatach PTB protokołu ICMP (więcej informacji na ich temat znajduje się w rozdziale 8.), które odpowiadają za wyznaczenie maksymalnego rozmiaru pakietu (niepowodującego fragmentacji) na trasie między dwiema jednostkami. Wspomniane komunikaty są zazwyczaj przetwarzane poniżej warstwy UDP i nie angażują aplikacji. Dlatego skorzystanie z mechanizmu sprowadza się zwykle do wywołania odpowiedniej funkcji API, która zwraca szacowaną wartość MTU na trasie do każdej z jednostek docelowych. Warstwa IP może również wykonać operację PMTUD niezależnie od aplikacji. Uzyskane dane są wówczas buforowane i unieważniane, jeśli odwołania do określonej stacji docelowej nie zostaną zaktualizowane w odpowiednim czasie.

10.8.1. Przykład

Do przeprowadzenia kolejnego testu został wykorzystany program `sock`. Za jego pomocą utworzono datagram UDP skutkujący powstaniem datagramu IPv4 o rozmiarze 1501 bajtów. Zarówno lokalny system, jak i sieć LAN obsługują dane o rozmiarze większym niż 1500 bajtów. Jednak parametr MTU łącza internetowego ma mniejszą wartość. Celem zadania jest przesłanie trzech komunikatów UDP do usługi `echo` (port UDP o numerze 7) w krótkich odstępach czasu.

```
Linux% sock -u -i -n 3 -w1473 www.cs.berkeley.edu echo
```

Przebieg pakietu zarejestrowany za pomocą programu `tcpdump` został przedstawiony na listingu 10.6 (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.6. *Komunikaty PTB protokołu ICMP zarejestrowane przez program tcpdump. W danych wynikowych zawarta jest sugerowana wartość MTU*

```
1 14:42:18.359366 IP (tos 0x0, ttl 64, id 18331, offset 0, flags [DF],
    proto UDP (17), length 1501)
    12.46.129.28.33954 > 128.32.244.172.7: UDP, length 1473

2 14:42:18.359384 IP (tos 0x0, ttl 64, id 18332, offset 0, flags [DF],
    proto UDP (17), length 1501)
    12.46.129.28.33954 > 128.32.244.172.7: UDP, length 1473

3 14:42:18.359402 IP (tos 0x0, ttl 64, id 18333, offset 0, flags [DF],
    proto UDP (17), length 1501)
    12.46.129.28.33954 > 128.32.244.172.7: UDP, length 1473

4 14:42:18.360156 IP (tos 0x0, ttl 255, id 23457, offset 0,
    flags [none], proto ICMP (1), length 56)
    12.46.129.1 > 12.46.129.28: ICMP
    128.32.244.172 unreachable - need to frag (mtu 1500), length 36
    IP (tos 0x0, ttl 63, id 18331, offset 0, flags [DF],
    proto UDP (17), length 1501)
    12.46.129.28.33954 > 128.32.244.172.7: UDP, length 1473
```

Na listingu 10.6 został pokazane trzy datagramy UDP. Pole danych każdego z nich ma rozmiar 1473 bajtów. Wynikowe datagramy IPv4 mają więc rozmiar 1501 bajtów (bez fragmentowania). W każdym datagramie ustawiony bit *DF* zapobiega fragmentacji (domyślne ustawienie testowanego systemu), więc gdy pakiet dociera do routera (adres IP routera to 12.46.129.1), wysyłany jest komunikat PTB protokołu ICMPv4 z sugestią, aby obniżyć wartość MTU do 1500 bajtów. W dostarczanych powiadomieniach ICMPv4 zawarte są nagłówki UDP/IPv4 (oraz 8 pierwszych bajtów danych) problematycznych datagramów. W analizowanym przykładzie program sock generował datagramy z tak dużą częstotliwością, że działanie polecenia zakończyło się przed dostarczeniem i przetworzeniem któregokolwiek z komunikatów ICMP.

Większość dostawców usług internetowych wyznacza minimalną wartość parametru MTU na poziomie 1500 bajtów. Niektórzy dostawcy korzystają jednak z mechanizmu PPPoE do przydzielania adresu i zarządzania połączeniami, co z kolei powoduje obniżenie wartości MTU do 1492 bajtów, bo nagłówek PPPoE zajmuje 6 bajtów (patrz rozdział 3.), a następujący po nim nagłówek PPP składa się z 2 bajtów. Na datagram zostają więc 1500 - 6 - 2 = 1492 bajty.

Jeśli odwołania zostaną skierowane do innej jednostki docelowej (dla której nie ma historii MTU), a pomiędzy poszczególnymi zapisami zostanie wprowadzona dodatkowa przerwa, sposób wykonania zadania ulegnie istotnej zmianie. Dodanie 2-sekundowej przerwy w generowaniu komunikatów sprowadza się do uwzględnienia w wywołaniu polecenia sock opcji -p 2. Oto przykład dwukrotnego wykonania tej samej instrukcji:

```
Linux% sock -u -i -n 3 -w1473 -p 2 www.wisc.edu echo
write returned -1, expected 1473: Message too long
Linux% sock -u -i -n 3 -w1473 -p 2 www.wisc.edu echo
```

Informacje zarejestrowane przez program tcpdump w czasie tej operacji zostały zamieszczone na listingu 10.7 (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.7. *Przykład ustalenia parametru MTU na interfejsie o wartości MTU równej 3000 bajtów dostosowującym swoje działanie do MTU trasy, które wynosi 1500 bajtów*

```
1 17:22:16.331023 IP (tos 0x0, ttl 64, id 58648, offset 0, flags [DF],
    proto: UDP (17), length: 1501)
    12.46.129.28.33955 > 144.92.9.185.7: UDP, length 1473

2 17:22:16.331581 IP (tos 0x0, ttl 255, id 38518, offset 0,
    flags [none], proto: ICMP (1), length: 56)
    12.46.129.1 > 12.46.129.28: ICMP
    144.92.9.185 unreachable - need to frag (mtu 1500), length 36
      IP (tos 0x0, ttl 63, id 58648, offset 0, flags [DF],
      proto: UDP (17), length: 1501)
      12.46.129.28.33955 > 144.92.9.185.7: UDP, length 1473

3 17:22:24.284866 IP (tos 0x0, ttl 64, id 53776, offset 0, flags [+],
    proto: UDP (17), length: 1500)
    12.46.129.28.33955 > 144.92.9.185.7: UDP, length 1473

4 17:22:24.284873 IP (tos 0x0, ttl 64, id 53776, offset 1480,
    flags [none], proto: UDP (17), length: 21)
    12.46.129.28 > 144.92.9.185: udp

5 17:22:26.293554 IP (tos 0x0, ttl 64, id 53777, offset 0, flags [+],
    proto: UDP (17), length: 1500)
    12.46.129.28.33955 > 144.92.9.185.7: UDP, length 1473

6 17:22:26.293559 IP (tos 0x0, ttl 64, id 53777, offset 1480,
    flags [none], proto: UDP (17), length: 21)
    12.46.129.28 > 144.92.9.185: udp

7 17:22:28.301469 IP (tos 0x0, ttl 64, id 53778, offset 0, flags [+],
    proto: UDP (17), length: 1500)
    12.46.129.28.33955 > 144.92.9.185.7: UDP, length 1473

8 17:22:28.301474 IP (tos 0x0, ttl 64, id 53778, offset 1480,
    flags [none], proto: UDP (17), length: 21)
    12.46.129.28 > 144.92.9.185: udp
```

Z zestawienia zaprezentowanego na listingu 10.7 wynika, że przyczyną błędu zgłoszonego w czasie pierwszego wywołania polecenia było dostarczenie komunikatu PTB w protokole ICMPv4. Dodatkowy czas pomiędzy kolejnymi transmisjami umożliwił dostarczenie komunikatu PTB do jednostki nadawczej, a także przekazanie i przetworzenie informacji o błędzie w jednostce nadawczej. Co ciekawe, po ponownym uruchomieniu program posługuje się parametrem MTU o wartości 1500 i wysyła trzy datagramy z zastosowaniem techniki fragmentacji (pakiety 3., 5. i 7. odpowiadają pierwszym fragmentom każdego z trzech datagramów). Po 15 minutach dane na temat MTU trasy zostają uznane za przedawnione (sytuacja ta nie została pokazana w przykładzie) i kolejny datagram jest wysyłany bez fragmentacji. Powoduje to dostarczenie następnego komunikatu PTB ICMPv4 i ponowienie całego procesu.

Zgodnie z zaleceniem [RFC1191] wartość MTU trasy (PMTU — *Path MTU*) ustalona za pomocą mechanizmu wykrywania MTU trasy (PMTUD — *Path MTU Discovery*) powinna być uznawana za przedawnioną po upływie 10 minut. Wyznaczenie wartości PMTU bywa niekiedy utrudnione ze względu na filtrowanie ruchu w zaporach i bramach sieciowych,

co negatywnie wpływa na działanie algorytmu PMTUD. Z tego względu wiele systemów umożliwia wyłączenie mechanizmu PMTUD lub dostosowanie go do używanych interfejsów. Wyłączenie opisywanego mechanizmu w systemie Linux sprowadza się do zapisania jedynki w pliku */proc/sys/net/ipv4/ip_no_pmtu_disc*. W systemie Windows to samo zadanie można wykonać, wpisując wartość 0 w rejestrze o kluczu *HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Tcpip\Parameters\EnablePMTUDiscovery*. Opracowano również rozwiązanie alternatywne [RFC4821], w którym operacja PMTUD nie zależy od pakietów ICMP. Jego omówienie znajduje się w rozdziale 15., poświęconym protokołowi TCP.

10.9. Zależność między fragmentacją IP i procesem ARP/ND

Korzystanie z protokołu UDP pozwala na zaobserwowanie zależności między fragmentacją datagramów IP i działaniem mechanizmu ARP. Jak wiadomo, procedura ARP służy do odwzorowania adresów IP na adresy MAC jednostek funkcjonujących w tej samej sieci IPv4 (więcej informacji na ten temat znajduje się w rozdziale 4.). Nasuwa się więc pytanie, ile komunikatów ARP musi zostać wygenerowanych w trakcie przesyłania fragmentów datagramu IP oraz ile emisji fragmentów jest wstrzymywanych do czasu zakończenia operacji ARP? Analogiczna zależność jest obserwowana w sieciach IPv6 bazujących na mechanizmie ND. Aby ustalić odpowiedzi na tak sformułowane pytanie, wystarczy wykonać dwa poniższe polecenia w sieci LAN o parametrze MTU równym 1500 bajtów:

```
Linux% sock -u -i -n1 -w8192 10.0.0.20 echo
Linux% sock -u -i -n1 -w8192 10.0.0.3 echo
```

Użyte w instrukcjach opcje wymuszają na programie `sock` wygenerowanie pojedynczego datagramu UDP z 8192 bajtami użytkownika. Taka wielkość pola danych wiąże się z koniecznością podzielenia datagramu na sześć fragmentów, które będzie można przesłać w sieci LAN o MTU wynoszącym 1500 bajtów. Oczywiście, trzeba się upewnić, że przed wykonaniem polecenia bufor ARP jest pusty. W przeciwnym przypadku wymiana żądania i odpowiedzi może się okazać zbędna. Wynik testu został przedstawiony na listingu 10.8 (niektóre wiersze zostały przełamane w celu zwiększenia czytelności).

Listing 10.8. *Procedura ARP i mechanizm fragmentacji w Ethernecie o parametrze MTU równym 1500 bajtów*

```
1 15:45:49.063561 arp who-has 10.0.0.20 tell 10.0.0.5
2 15:45:50.059523 arp who-has 10.0.0.20 tell 10.0.0.5
3 15:45:51.059505 arp who-has 10.0.0.20 tell 10.0.0.5
---
4 15:46:08.555725 arp who-has 10.0.0.3 tell 10.0.0.5
5 15:46:08.555973 arp reply 10.0.0.3 is-at 0:0:c0:c2:9b:26
6 15:46:08.555992 10.0.0.5 > 10.0.0.3:
     udp (frag 27358:1480@2960+) (ttl 64, len 1500)
7 15:46:08.555998 10.0.0.5 > 10.0.0.3:
     udp (frag 27358:1480@1480+) (ttl 64, len 1500)
8 15:46:08.556004 10.0.0.5.32808 > 10.0.0.3.7:
     udp 8192 (frag 27358:1480@0+) (ttl 64, len 1500)
```

Podczas wykonywania pierwszego testu komputer o adresie 10.0.0.20 nie pracował w sieci. Z tego powodu nie zarejestrowano żadnej odpowiedzi ARP. Z wierszy od 1. do 3. wynika jednak, że żądania ARP były wysyłane w 1-sekundowych odstępach. Ponieważ po trzech próbach jednostka zdalna nie odpowiedziała, procedura ARP została zakończona niepowodzeniem. W drugim teście odpowiedź ARP została dostarczona do stacji inicjującej operację po 250 μs, a 20 μs później komputer wysłał pierwszy fragment. Kolejne fragmenty zostały wyemitowane w bardzo krótkich odstępach czasowych, wynoszących około 6 μs. Należy pamiętać, że w systemie Linux (w którym doświadczenie zostało przeprowadzone) ostatni fragment jest wysyłany jako pierwszy.

W przeszłości interakcje między mechanizmem fragmentacji a modułem ARP były bardzo problematyczne. Przykładowo w niektórych implementacjach żądania ARP były generowane przed każdym fragmentem. W innych z kolei jedynie pierwszy fragment podlegał buforowaniu przed zakończeniem procedury ARP (co prowadziło do utraty datagramu, ponieważ wszystkie fragmenty oprócz pierwszego były odrzucane). Pierwszy z problemów opisano w dokumencie [RFC1122]. W tym samym zaleceniu wprowadzono zakaz implementowania rozwiązań, które prowadziłyby do wspomnianego zalewania sieci żądaniami ARP. Maksymalną częstotliwość wysyłania takich pakietów określono na poziomie 1 sekundy. Druga z niedogodności również została opisana w dokumencie [RFC1122], ale wzmianka na temat rozwiązania problemu ogranicza się jedynie do stwierdzenia, że warstwa łącza danych „POWINNA zachować (a nie odrzucić) przynajmniej jeden (ostatni) pakiet z każdego zbioru pakietów przeznaczonych dla jednostki o danym nieodwzorowanym adresie IP i przesłać ten pakiet po odwzorowaniu adresu". Takie rozwiązanie może jednak prowadzić do niepotrzebnej utraty pakietów. Dlatego w poszczególnych implementacjach mechanizmu zapewnia się dostępność dłuższej kolejki na pakiety oczekujące na wykonanie procedury ARP.

10.10. Maksymalny rozmiar datagramu UDP

Teoretycznie maksymalny rozmiar datagramu IPv4 wynosi 65 535 bajtów. Wynika to z 16-bitowej długości pola *Całkowita długość* wchodzącego w skład nagłówka IPv4 (patrz rozdział 5.). Sam nagłówek IPv4 bez dodatkowych opcji zajmuje 20 bajtów. Z kolei nagłówek UDP jest zapisywany na 8 bajtach. Na dane przenoszone w datagramie UDP pozostaje więc 65 507 bajtów. W protokole IPv6 16-bitowe pole *Długość pola danych* umożliwia przenoszenie datagramu UDP z 65 527 bajtami użytkownika (8 z 65 535 bajtów pola danych IPv6 zajmuje nagłówek UDP), o ile nie są wykorzystywane jumbogramy. Jednak segmenty o tak dużych rozmiarach mogą nie docierać do odbiorców. Istnieją dwa powody takiego stanu rzeczy. Po pierwsze, niektóre moduły obsługi protokołu w systemie lokalnym mają pewne ograniczenia. Po drugie, aplikacja zdalna bywa nieprzygotowana do przyjmowania tak dużych datagramów.

10.10.1. Ograniczenia implementacyjne

Moduł obsługi protokołu udostępnia aplikacjom interfejs programistyczny (API), za którego pomocą wybierany jest pewien domyślny rozmiar bufora nadawczego i bufora odbiorczego. W niektórych implementacjach wartości domyślne są mniejsze niż maksymalny rozmiar datagramu IP, a w części rozwiązań nie są w ogóle obsługiwane datagramy o rozmiarze większym niż kilkadziesiąt kilobajtów (choć nie jest to powszechny problem).

Interfejs API gniazd [UNP3] oferuje funkcje, które pozwalają aplikacjom na sprawdzenie oraz ustawienie określonego rozmiaru buforów nadawczego i odbiorczego. Dla gniazd UDP rozmiar ten jest bezpośrednio związany z maksymalną wielkością datagramu UDP, który aplikacja może zapisać lub odczytać. Typowymi wartościami omawianych parametrów są 8 192 bajty oraz 65 535 bajtów. Można je jednak zmienić, wywołując funkcję `setsockopt()`.

Zgodnie z informacjami zamieszczonymi w rozdziale 5. stacja robocza jest zobowiązana do zapewnienia dostatecznie dużego bufora, aby odbierać datagramy IPv4 o rozmiarze co najmniej 576 bajtów. W wielu aplikacjach UDP na etapie projektowania nakładane jest ograniczenie do 512 bajtów danych użytkownika (co skutkuje utworzeniem datagramów IPv4 o rozmiarze 576 bajtów). Rozwiązania tego typu są stosowane m.in. w serwerach DNS (patrz rozdział 11.) oraz serwerach DHCP (patrz rozdział 6.).

10.10.2. Obcinanie datagramów

To, że stacja nadawcza jest w stanie emitować i przyjmować datagramy UDP/IP o danym (dużym) rozmiarze, wcale nie oznacza, że jednostka odbiorcza jest przygotowana do odczytu danych o takim samym rozmiarze. Interfejs programistyczny protokołu UDP pozwala twórcom aplikacji na wyznaczenie maksymalnej liczby bajtów, która jest pobierana w każdej operacji odczytu. Co się stanie, gdy odebrany datagram przekroczy tę wartość?

W większości przypadków odpowiedź jest taka sama — interfejs API *obetnie* datagram, odrzucając te bajty danych, które wykraczają poza określony bufor aplikacji odbiorczej. Ostateczny sposób postępowania zależy jednak od konkretnej implementacji mechanizmu. Niektóre systemy zwracają nieodczytaną część datagramu w kolejnych operacjach odczytu. Inne informują proces wywołujący o ilości odrzuconych danych (albo o tym, że *pewne* dane zostały odrzucone, ale ich ilość jest nieznana).

W systemie Linux do ustalenia, ile danych zostało odrzuconych, wystarczy ustawić opcję `MSG_TRUNC` w interfejsie API gniazd. Z kolei w systemie HP-UX znacznik `MSG_TRUNC` jest ustawiany, gdy w wyniku operacji odczytu zwracane są okrojone dane. Interfejs API gniazd w wersjach poprzedzających SVR4 (włącznie z systemem Solaris 2.x) w ogóle *nie obcina* datagramów. Wszelkie nadmiarowe dane są zwracane w kolejnych odczytach. Aplikacja nie jest jednak powiadamiana o tym, że pobranie pojedynczego datagramu wymaga wykonania wielu operacji odczytu.

Analizując ruch TCP, trzeba pamiętać, że ma on formę ciągłego strumienia bajtów dostarczanych do aplikacji bez podziału na komunikaty. Aplikacja może więc odczytać dowolną ilość danych, o ile dane te są dostępne (jeśli nie, aplikacja czeka na ich dostarczenie).

10.11. Budowa serwera UDP

Protokół UDP ma pewne cechy, które wpływają na sposób projektowania i implementowania aplikacji sieciowych na nim bazujących [RFC5405]. Programy serwerowe zazwyczaj współdziałają z systemem operacyjnym i korzystają z mechanizmów umożliwiających obsługę wielu żądań klienckich w tym samym czasie. Projektowanie i tworzenie aplikacji klienckich jest łatwiejsze, dlatego nie zostało omówione w książce.

W czasie typowej komunikacji klient-serwer jednostka kliencka rozpoczyna swoje działanie, natychmiast odwołuje się do pojedynczego serwera i kończy pracę. Z kolei serwery uruchamiają się i przechodzą w tryb uśpienia, oczekując na nadsyłane żądania. Odebranie datagramu klienckiego powoduje „obudzenie" serwera, sprawdzenie otrzymanych informacji i przekazanie do dalszego przetwarzania. Dalszy opis nie dotyczy jednak szczegółowych aspektów programowania aplikacji klienckich i serwerowych (zostały one opisane w dokumencie [UNP3]), lecz cech protokołu UDP, które wpływają na przygotowanie i wdrożenie usług bazujących na mechanizmie UDP (omówienie zasad budowy serwera TCP zostało zamieszczone w rozdziale 13.). Mimo że część cech zależy w pewnym stopniu od konkretnej implementacji mechanizmu, większość opisywanych tutaj rozwiązań ma charakter uniwersalny.

10.11.1. Adresy IP i numery portów UDP

Dane odbierane przez serwer UDP od klienta mają formę datagramu UDP, w którym nagłówek IP zawiera adresy IP źródłowy i docelowy, a nagłówek UDP przenosi informacje o numerach portów docelowego i źródłowego. Do aplikacji dostarczany jest jednak komunikat pozbawiony nagłówków IP i UDP. System operacyjny musi w inny sposób poinformować aplikację, kto przesłał komunikat (jaki jest adres IP źródła oraz jaki jest numer portu źródłowego), jeśli jest to niezbędne do sformowania odpowiedzi. Dzięki tej funkcji serwer UDP może obsługiwać jednocześnie wiele odwołań klienckich.

Niektóre serwery muszą rejestrować *odbiorców* datagramów, a dokładniej, odpowiadające im docelowe adresy IP. Choć mogłoby się wydawać, że wystarczy w tym celu przeanalizować odebrany datagram, w praktyce zadanie nie zawsze jest tak trywialne. Przy użyciu takich rozwiązań jak zwielokrotnianie interfejsów, tworzenie aliasów adresów IP czy wykorzystywanie wielu zakresów IPv6 pojedyncza jednostka może dysponować wieloma adresami IP, a serwer może odbierać datagramy z każdego z nich (co zdarza się dość często). Jeśli serwer uzależnia wykonanie zadania od wartości docelowego adresu IP, musi mieć dostęp do informacji na temat takich adresów. Ponadto niektóre usługi działają w różny sposób, zależnie od tego, czy adres docelowy jest adresem multiemisji lub adresem rozgłoszeniowym (np. zdefiniowane w dokumencie [RFC1122] wymagania dotyczące jednostki sieciowej zawierają stwierdzenie, że serwer TFTP powinien ignorować datagramy wysyłane na rozgłoszeniowy adres IP).

Jednym z serwerów uzależniających swoje działanie od docelowego adresu IP jest serwer DNS. Informacja o adresie pozwala mu na odpowiednie posortowanie zwracanych wyników odwzorowania. Wzmiankowany sposób działania serwera DNS został szczegółowo opisany w rozdziale 11.

Trzeba więc pamiętać, że jeśli nawet interfejs API zapewnia dostarczenie wszystkich danych zawartych w datagramie warstwy transportowej, efektywne działanie serwera może być uzależnione również od dostępności dodatkowych informacji z innych warstw stosu protokołów (zazwyczaj informacji o adresowaniu). Problem ten nie dotyczy — oczywiście — jedynie mechanizmu UDP, ale ponieważ jest to pierwszy z omawianych protokołów, należy o tym fakcie wspomnieć.

Serwery UDP zaprojektowane do pracy w sieciach IPv4 i IPv6 muszą uwzględniać dwa różne typy adresowania (o różnej długości adresów oraz różnej strukturze danych). Ponadto funkcje odwzorowywania adresów IPv4 na adresy IPv6 mogą pozwalać na wyko-

rzystywanie gniazd IPv6 do obsługi zarówno adresowania IPv4, jak i IPv6. Więcej informacji na ten temat znajduje się w opracowaniu [UNP3].

10.11.2. Ograniczenie użycia lokalnych adresów IP

Podczas powoływania gniazda UDP większość serwerów UDP posługuje się **wieloznacznymi** (*wildcard*) lokalnymi adresami IP. Oznacza to, że nadchodzące datagramy ze wskazanym portem serwera są akceptowane na każdym lokalnym adresie IP (dozwolone jest użycie każdego z lokalnych adresów IP, w tym również adresu pętli zwrotnej). Przykładowo nic nie stoi na przeszkodzie, żeby uruchomić serwer UDP w protokole IPv4 na porcie 777 za pomocą następującego polecenia:

```
Linux% sock -u -s 7777
```

Aby sprawdzić stan gniazda serwerowego, wystarczy wprowadzić instrukcję `netstat` (co pokazujemy na listingu 10.9).

Listing 10.9. *Wynik wykonania polecenia netstat — lista serwerów UDP powiązanych ze wszystkimi adresami IPv4*

```
Linux% netstat -l --udp -n
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address
udp        0      0 *:7777                  0.0.0.0:*
```

Z listingu zostały usunięte wiersze niedotyczące uruchomionego wcześniej serwera. Opcja `-l` odpowiada za wyświetlenie informacji o wszystkich nasłuchujących gniazdach (serwerach). Opcja `--udp` dostarcza danych na temat protokołu UDP. Z kolei opcja `-n` wymusza przedstawienie adresów IP zamiast nazw domenowych stacji.

Choć przedstawione opcje nie są dostępne we wszystkich wersjach polecenia `netstat`, każda wersja programu `netstat` umożliwia przekazanie pewnej kombinacji opcji, które umożliwiają uzyskanie analogicznego zestawienia. W systemach BSD dostępne są np. opcje `-l` i `-p udp`, a w systemach Windows można zamiast nich użyć `-n`, `-a` oraz `-p udp`.

Lokalny adres IP został przedstawiony w formie `*:7777`. Znak gwiazdki reprezentuje właśnie wieloznaczny adres IP. Gdy serwer tworzy punkt końcowy komunikacji, wyznacza jeden z lokalnych adresów IP stacji (może to być również adres rozgłoszeniowy) jako lokalny adres IP punktu końcowego. W wyniku tej operacji do utworzonego gniazda są przekazywane jedynie datagramy UDP zawierające pasujący adres IP. Gdyby podczas uruchamiania programu `sock` został podany adres IP (przed numerem portu), stałby się on lokalnym adresem IP punktu końcowego. Oto przykład stosownego polecenia:

```
Linux% sock -u -s 127.0.0.1 7777
```

Uruchomiony w ten sposób serwer odbiera datagramy jedynie na lokalnym interfejsie pętli zwrotnej (`127.0.0.1`), czyli datagramy generowane przez samą stację. Wynik wykonania polecenia `netstat` zmienia się wówczas zgodnie z listingiem 10.10.

Listing 10.10. *Wynik wykonania polecenia netstat — lista serwerów UDP powiązanych z lokalnym interfejsem pętli zwrotnej*

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address
udp        0      0 127.0.0.1:7777          0.0.0.0:*
```

Próba przesłania do serwera datagramu z innej stacji pracującej w tej samej sieci Ethernet zakończy się zwróceniem komunikatu o niedostępności portu docelowego (ICMPv4 Port Unreachable), a działanie aplikacji nadawczej zostanie przerwane. Serwer nie otrzyma takiego datagramu.

```
Linux% sock -u -v 10.0.0.3 6666
connected on 10.0.0.5.50997 to 10.0.0.3.6666
123
error: Connection refused
```

10.11.3. Wykorzystanie wielu adresów

Ten sam port może być wykorzystywany przez różne serwery posługujące się odmiennymi lokalnymi adresami IP. Zazwyczaj wymaga to jednak poinformowania systemu, że aplikacja akceptuje współdzielenie danego numeru portu.

W interfejsie API gniazd konieczne jest zdefiniowanie opcji SO_REUSEADDR. W testowym programie sock odpowiada za to opcja -A.

Dodatkowe adresy IP można ustawić nawet wtedy, kiedy w systemie została zainstalowana tylko jedna karta sieciowa. Jednostka używana do wykonywania opisywanych testów korzystała z adresu IPv4 10.0.0.3. Niemniej za pomocą poniższych instrukcji możliwe było przypisanie jej dwóch kolejnych adresów:

```
Linux# ip addr add 10.0.2.13 scope host dev eth0
Linux# ip addr add 10.0.2.14 scope host dev eth0
```

Po tej operacji system posługiwał się czterema adresami IPv4 emisji pojedynczej — natywnym adresem IP4, dwoma adresami dodanymi za pomocą przedstawionych poleceń oraz adresem pętli zwrotnej. Nic więc nie stoi na przeszkodzie, aby uruchomić trzy różne egzemplarze serwera UDP na tym samym porcie (posłużą do tego program sock oraz port 8888):

```
Linux% sock -u -s -A 10.0.2.13 8888
Linux% sock -u -s -A 10.0.2.14 8888
Linux% sock -u -s -A 8888
```

Serwery muszą zostać uruchomione z opcją -A, która informuje, że aplikacja zgadza się na wielokrotne użycie tego samego portu. Na listingu 10.11 widoczne są adresy i numery portów serwerów wyświetlane przez polecenie netstat po wykonaniu opisanego zadania.

Listing 10.11. *Serwery UDP pracujące na wskazanych i wieloznacznych adresach IP i na tym samym porcie UDP*

```
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address
udp        0      0 10.0.2.13:8888      0.0.0.0:*
udp        0      0 0.0.0.0:8888        0.0.0.0:*
udp        0      0 10.0.2.14:8888      0.0.0.0:*
```

W rozważanym przykładzie do serwera o wieloznacznym adresie IPv4 będą dostarczane te datagramy, które zostały wysłane na adres `10.0.0.30`, kierowany adres rozgłoszeniowy (np. `10.255.255.255`), adres rozgłoszenia lokalnego (`255.255.255.255`) lub adres pętli zwrotnej (`127.0.0.1`). Pozostałe adresy są powiązane z innymi serwerami.

W przypadku uruchomienia serwera z wieloznacznym adresem lokalnym obowiązują pewne priorytety w odbieraniu informacji. Punkt końcowy o zdefiniowanym adresie IP jest wybierany jako pierwszy. Punkt końcowy o wieloznacznym adresie IP otrzymuje więc wszystkie datagramy, które nie pasują do precyzyjnie określonych adresów.

10.11.4. Ograniczenie zdalnych adresów IP

We wszystkich wierszach wynikowych polecenia `netstat` adres IP jednostki zdalnej (`Foreign Address`) oraz numer portu zdalnego miały wartość `0.0.0.0:*`. Oznacza to, że punk końcowy akceptuje datagramy UDP nadchodzące z dowolnego adresu IP i dowolnego portu. Możliwe jest jednak ograniczenie zakresu jednostek źródłowych. Dzięki niemu gniazdo otrzymuje jedynie datagramy IP wygenerowane przez komputery o ustalonym adresie IPv4 i wskazanym porcie. Ograniczenie to jest nakładane w chwili odebrania komunikatu ze stacji klienckiej, co pozwala na filtrowanie ruchu z innych stacji. Aby określić zdalny adres IPv4 oraz numer portu w trakcie pracy programu `sock`, wystarczy do jego wywołania dodać opcję `-f`.

```
Linux% sock -u -s -f 10.0.0.14.4444 5555
```

W powyższym przykładzie dodanie opcji `-f` spowodowało wyznaczenie zdalnego adresu IPv4 jako `10.0.0.14` oraz zdalnego numeru portu o wartości `4444`. Port serwera to `5555`. Jeśli po uruchomieniu programu wykonane zostanie polecenie `netstat`, na ekranie powinien zostać wyświetlony również lokalny adres IP, który nie został określony wprost (zgodnie z listingiem 10.12).

Listing 10.12. *Ograniczenie zdalnego adresu IP powoduje jednoczesne przypisanie adresu lokalnego*

```
Linux% netstat --udp -n
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address       Foreign Address State
udp        0      0 10.0.0.30:5555 1    0.0.0.14:4444   ESTABLISHED
```

Jest to typowy efekt uboczny określenia zdalnego adresu IP i zdalnego portu — pominięcie adresu lokalnego przy jednoczesnym wskazaniu adresu zdalnego powoduje automatyczne wybranie jednego z adresów lokalnych. Wskazywany jest wówczas ten adres IP, który w procesie routingu musi zostać wybrany do komunikacji z jednostką o podanym

adresie zdalnym. W analizowanym przykładzie użyty został podstawowy adres karty ethernetowej przyłączonej do sieci, w której pracuje jednostka 10.0.0.30. Warto też zwrócić uwagę na to, że wskazanie adresu zdalnego oraz wyznaczenie punku końcowego spowodowały zapisanie w kolumnie State (stan) informacji o ustanowionym powiązaniu (ESTABLISHED).

Trzy możliwe do zdefiniowania wiązania adresu serwera UDP zostały zestawione w tabeli 10.2.

Tabela 10.2. *Sposoby przypisania adresów serwera UDP*

Adres lokalny	Adres zdalny	Opis
lokalny_IP.lokalny_port	*zdalny_IP.zdalny_port*	Ograniczenie do jednego klienta.
lokalny_IP.lokalny_port	*.* (adres wieloznaczny)	Ograniczenie do jednego lokalnego adresu IP i jednego lokalnego portu (z dowolnymi adresami klienckimi).
*.*port_lokalny*	*.* (adres wieloznaczny)	Ograniczenie do jednego portu lokalnego.

We wszystkich przypadkach *lokalny_port* jest portem serwera, a *lokalny_IP* musi być jednym z adresów IP przypisanych systemowi serwera. Kolejność poszczególnych wierszy tabeli odpowiada kolejności wybierania odpowiednich lokalnych punktów końcowych podczas przetwarzania odebranego datagramu. Najbardziej szczegółowe powiązania (pierwszy wiersz) są analizowane w pierwszej kolejności. Najbardziej ogólne (ostatni wiersz — z dwoma adresami wieloznacznymi) jest sprawdzane jako ostatnie.

10.11.5. Wiele serwerów na jednym porcie

Choć nie wynika to z zaleceń RFC, większość implementacji stosu protokołów umożliwia tylko jednej aplikacji używanie określonej pary lokalnego adresu IP i numeru portu UDP w danym czasie. Ograniczenie dotyczy jednak danej **rodziny adresów** (tj. IPv4 lub IPv6). Gdy datagram UDP zostaje dostarczony do komputera na podstawie podanego adresu IP i numeru portu, jego kopia jest przekazywana do pojedynczego punktu końcowego (np. nasłuchującej aplikacji). Adres IP może mieć formę adresu wieloznacznego (zgodnie z wcześniejszymi przykładami), ale obsługa datagramów należy do jednej aplikacji powiązanej z danym adresem (lub adresami). Próba uruchomienia innego serwera z wieloznacznym adresem lokalnym i tym samym numerem portu w ramach tej samej rodziny adresów zakończy się niepowodzeniem.

```
Linux% sock -u -s 12.46.129.3 8888 &
Linux% sock -u -s 12.46.129.3 8888
can't bind local address: Address already in use  //nie można przypisać adresu lokalnego: adres w użyciu
```

W rozwiązaniach wykorzystujących mechanizm multiemisji (patrz rozdział 9.) wiele punktów końcowych może używać tej samej pary lokalnego adresu IP i numeru portu UDP. Niemniej jednak konieczne jest powiadomienie interfejsu API, że aplikacja akceptuje taki sposób działania (we wcześniejszym przykładzie wymagało to dodania opcji -A włączającej opcję SO_REUSEADDR).

W systemie BSD 4.4 współdzielenie portu przez kilka punktów końcowych wymaga ustawienia innej opcji gniazda (SO_REUSEPORT). Poza tym wspomniana opcja musi zostać zdefiniowana we wszystkich punktach końcowych, włącznie z pierwszym przypisanym do danego portu.

Po odebraniu datagramu UDP o docelowym adresie IP właściwym dla multiemisji lub rozgłoszenia kopia datagramu jest dostarczana do każdego punktu końcowego przypisanego do jednego adresu IP i jednego portu (lokalny adres IP punktu końcowego może być również adresem wieloznacznym, pasującym do każdego adresu docelowego pakietu IP). Jeśli jednak do jednostki dotrze datagram IP o adresie **emisji pojedynczej** (tj. ze standardowym adresem IP), tylko kopia datagramu jest dostarczana tylko do *jednego* ze zbioru punktów końcowych. O tym, do którego z punktów końcowych trafi taki datagram, decyduje sposób implementacji mechanizmu UDP w konkretnym systemie. Rozwiązanie to pozwala jednak wielowątkowym i wieloprocesowym serwerom na poprawne działanie bez wielokrotnego wywoływania kodu do obsługi tego samego żądania.

10.11.6. Objęcie dwóch rodzin adresów — IPv4 i IPv6

Kod serwerów można przygotować w taki sposób, aby obejmował nie tylko różne protokoły (przykładem są serwery działające w protokołach TCP i UDP jednocześnie), ale również różne rodziny adresów. Nic nie stoi na przeszkodzie, aby dany serwer UDP odpowiadał na żądania klienckie dostarczane w protokołach IPv4 i IPv6. Choć mogłoby się to wydawać oczywiste (adresy IPv6 są przecież jedynie 128-bitowymi wersjami klasycznych adresów IP), trzeba pamiętać o pewnych niuansach związanych ze współdzieleniem numerów portów. W niektórych systemach przestrzeń numerów portów jest *współdzielona* przez protokoły IPv6 i IPv4 zarówno w mechanizmach UDP, jak i TCP. Oznacza to, że jeśli jedna usługa zostanie przypisana do portu UDP protokołu IPv4, automatycznie otrzymuje ten sam port w przestrzeni portów protokołu IPv6 (i vice versa). Wykorzystanie tego samego portu przez inną usługę staje się więc niemożliwe (jeśli nie została ustawiona wzmiankowana wcześniej opcja SO_REUSEADDR). Ponadto funkcja odwzorowywania adresów IPv4 w adresach IPv6 (patrz rozdział 2.) powoduje, że powiązania z adresami wieloznacznymi skutkują przekazywaniem ruchu IPv4 do mechanizmów obsługi żądań IPv6.

Opisywany sposób działania jest zależny od implementacji określonego mechanizmu. W systemach Linux cała przestrzeń numerów portów jest współdzielona, a powiązania wieloznacznych adresów IPv6 wymuszają analogiczne powiązania w protokole IPv4. W systemach FreeBSD można natomiast użyć opcji IPV6_V6ONLY, która gwarantuje, że wspomniane powiązania zostaną ograniczone do przestrzeni IPv6. Dlatego programiści aplikacji serwerowych powinni dostosować interfejs komunikacji z gniazdami IPv6 do rodzaju docelowego systemu operacyjnego. Powiązania definiowane w języku C zostały opisane w dokumencie [RFC3493].

10.11.7. Brak mechanizmów sterowania przepływem i przeciążeniami

Większość serwerów UDP ma charakter programów **iteracyjnych**. Oznacza to, że pojedynczy wątek (lub proces) serwera obsługuje wszystkie żądania klienckie dostarczane do jednego portu UDP (do zarezerwowanego portu serwera). Zazwyczaj każdemu portowi UDP odpowiada kolejka wejściowa o określonej pojemności. Żądania klienckie napływa-

jące w tym samym czasie podlegają więc buforowaniu przez mechanizm obsługi protokołu UDP. Następnie są przekazywane do aplikacji serwerowej (w następujących po sobie operacjach odczytu) w kolejności zapisywania ich w kolejce (zgodnie z mechanizmem FIFO).

Taki sposób działania może spowodować przepełnienie bufora, które z kolei prowadzi do odrzucania nadchodzących datagramów. Problem może wystąpić nawet w przypadku obsługiwania żądań tylko jednego klienta. Protokół UDP nie uwzględnia żadnych mechanizmów **sterowania przepływem** (serwer nie może w żaden sposób poinformować klienta o konieczności zmniejszenia częstotliwości wysyłania datagramów). Definiuje on komunikację bezpołączeniową pozbawioną jakichkolwiek rozwiązań gwarantujących niezawodność transmisji, w tym również powiadamiania aplikacji o przepełnieniu wejściowej kolejki UDP. Nadmiarowe datagramy są po prostu odrzucane na poziomie warstwy transportowej.

Kolejnym zmartwieniem projektantów systemów jest konieczność stosowania kolejek w routerach IP pośredniczących w wymianie danych między nadawcą i odbiorcą (w środku sieci). Gdy zostaną one zapełnione, komunikaty są odrzucane na zasadach podobnych do obowiązujących w kolejkach UDP. Wystąpienie takiej sytuacji jest nazywane **przeciążeniem**. Przeciążenia są zjawiskami niepożądanymi, ponieważ negatywnie wpływają na cały ruch sieciowy generowany przez użytkowników końcowych przesyłających dane przez przeciążony obszar sieci. Jest to sytuacja nieco inna niż w przypadku przepełnienia wejściowych kolejek protokołu UDP, ponieważ nie dotyczy tylko jednej aplikacji serwerowej. Komunikacja w protokole UDP jest szczególnie narażona na przeciążenia z powodu braku możliwości poinformowania urządzenia nadawczego o wystąpieniu problemu i konieczności zmniejszenia częstotliwości wysyłania pakietów. Z tego powodu rozwiązania UDP klasyfikuje się jako systemy pozbawione mechanizmów **sterowania przeciążeniami**. Sterowanie przeciążeniami jest bardzo złożonym zagadnieniem, które wciąż pozostaje w polu zainteresowania naukowców. Zagadnienie to zostało opisane w rozdziale poświęconym protokołowi TCP (w rozdziale 16.).

10.12. Translacja datagramów UDP/IPv4 i UDP/IPv6

W rozdziale 7. opisaliśmy środowisko odpowiedzialne za translację datagramów IPv4 na IPv6 i odwrotnie. Tematem rozdziału 8. było z kolei zastosowanie tego mechanizmu w protokole ICMP. Gdy datagramy UDP przechodzą przez moduł translatora, tłumaczenie jest realizowane w taki sam sposób, jaki został opisany w rozdziale 7. Pewien problem stanowi jednak generowanie sumy kontrolnej UDP. W datagramach UDP/IPv4 dopuszczalne jest ustawienie zera (wyłączenie obliczania sumy kontrolnej) w polu *Suma kontrolna*. Tymczasem w protokole UDP/IPv6 jest to zabronione. W konsekwencji datagramy o zerowej wartości kontrolnej podlegające przekształceniu z komunikatów IPv4 w komunikaty IPv6 są zastępowane datagramami UDP/IPv6 z pełną sumą kontrolną pseudonagłówka lub są odrzucane w chwili odebrania pakietu. Moduł translatora powinien udostępniać opcję regulującą sposób wykonywania operacji, ponieważ dodatkowe obliczenia związane z generowaniem sum kontrolnych mogą się okazać przeszkodą. Pakiety zawierające niezerową wartość kontrolną wymagają zaktualizowania pola kontrolnego w czasie translacji w każdym z kierunków, jeśli działanie mechanizmu mapowania adresów działa w trybie uwzględniania sum kontrolnych (więcej informacji na ten temat znajduje się w rozdziale 7.).

Kolejnym wyzwaniem okazuje się obsługa fragmentów datagramów. Bezstanowe moduły translacji nie mogą wykonać tłumaczenia fragmentów datagramów UDP/IPv4 o zerowej sumie kontrolnej, ponieważ nie mogą obliczyć odpowiedniej wartości sumy protokołu UDP/IPv6. Muszą więc być odrzucane. Stanowe mechanizmy (np. NAT64) mają możliwość odtworzenia datagramu na podstawie jego fragmentów i obliczenia niezbędnej wartości. Fragmentowane datagramy UDP/IP z wyznaczoną sumą kontrolną są traktowane jak wszystkie inne pakiety niezależnie od kierunku translacji (zgodnie z procedurą opisaną w rozdziale 7.). Duże datagramy UDP/IPv4, które muszą zostać podzielone, aby zmieściły się w datagramie IPv6 o minimalnej wartości MTU, podlegają takiemu samemu przetwarzaniu jak standardowe datagramy IPv4 (są fragmentowane, jeśli trzeba).

10.13. UDP w Internecie

Ustalenie udziału ruchu UDP w ruchu internetowym nie jest łatwe, ponieważ użyteczne dane nie są publicznie dostępne, a opracowania na temat obciążenia sieciowego są różnią się w poszczególnych obszarach sieci. Z badań, takich jak [FKMC03], wynika, że protokół UDP stanowi od 10% do 40% całego ruchu internetowego, a jego udział rośnie wraz z upowszechnianiem się aplikacji typu peer-to-peer [Z09]. Niemniej jednak nadal dominującą formą komunikacji jest TCP.

Według badań [SMC02] fragmentacja datagramów internetowych dotyczy przede wszystkim protokołu UDP (protokół UDP stanowi 68,3% całego fragmentowanego ruchu sieciowego), choć w ogólnym ujęciu bardzo mała część ruchu w ogóle podlega fragmentacji (około 0,3% pakietów, czyli 0,8% bajtów). Z raportu wynika, że najczęściej fragmentowane są datagramy UDP przenoszące dane multimedialne (53% całości, w czym program Microsoft Media Player jest odpowiedzialny za połowę tej wartości) oraz enkapsulowane (tunelowane) pakiety połączeń VPN (około 22%). Ponadto blisko 10% fragmentacji jest wykonywane w **porządku odwróconym** (takie rozwiązanie zostało zaprezentowane w przykładzie, w którym ostatni fragment IP był wysyłany jako pierwszy). Najczęściej obserwowane rozmiary fragmentów to 1500 bajtów (79%), 1484 bajty (18%) oraz 1492 bajty (1%).

Wartość 1500 bajtów parametru MTU wynika ze standardowego użytecznego rozmiaru pola danych ramki ethemetowej. Rozmiar 1484 bajtów został wprowadzony przez firmę Digital Equipment Corporation w urządzeniach GigaSwitch (dzisiaj raczej nieużywanych), które w czasie badań stanowiły istotną część topologii sieci.

Wydaje się, że są dwie zasadnicze przyczyny fragmentacji, czyli niefrasobliwa enkapsulacja oraz brak procedur wykrywania i uwzględnienia wartości MTU trasy w aplikacjach, które przetwarzają komunikaty o dużych rozmiarach. Pierwszy z wymienionych problemów wiąże się z wieloma poziomami enkapsulacji w różnych protokołach, z których każdy dodaje własne nagłówki, co powoduje, że pakiety IP mieszczące się początkowo w 1500-bajtowym polu danych (jest to typowa wartość MTU) stają się zbyt duże (np. w rozwiązaniach przekazujących dane w tunelach VPN). Drugi czynnik jest charakterystyczny dla aplikacji generujących duże pakiety (np. systemów wideo), które muszą zostać podzielone na mniejsze części. Co ciekawe, z badań [SMC02] wynika, że wiele datagramów UDP przekazywanych w pakietach IPv4 z ustawionym bitem DF (odpowiedzialnych prawdopodobnie za ustalanie parametru PMTU) jest enkapsulowanych w pakietach o nieustawionym bicie zakazu fragmentacji (co uniemożliwia wykonanie operacji i pozostawia aplikację bez informacji o problemie).

10.14. Ataki z użyciem protokołu UDP i fragmentacji IP

Większość ataków związanych z działaniem protokołu UDP polega na dążeniu do wyczerpania współdzielonych zasobów (buforów, pojemności łączy itp.) lub wykorzystaniu błędów w budowie stosu protokołów, które prowadzą do zawieszenia systemu lub nieprzewidzianego działania. Obie techniki należą do ogólnej kategorii ataków DoS (osoba atakująca dąży w nich do tego, aby użytkownicy uprawnieni do korzystania z usługi nie mogli się z nią skontaktować). Najprostszy atak DoS z użyciem protokołu UDP polega na generowaniu ruchu o dużym natężeniu z maksymalną częstotliwością. Ponieważ protokół UDP nie ma żadnych mechanizmów sterowania szybkością transmisji, opisane działanie prowadzi do obniżenia wydajności aplikacji współdzielących tę samą trasę transmisji pakietów. Taka sytuacja może być również wynikiem niecelowego działania użytkownika.

Inną, bardziej wyrafinowaną formą ataku DoS z użyciem protokołu UDP jest atak **ze wzmocnieniem**. Działanie osoby atakującej sprowadza się do wysłania niewielkiej liczby datagramów, które wymuszają na innych systemach wygenerowanie większego ruchu. W tzw. **ataku fragglesów** (*fraggle attack*) osoba atakująca podmienia źródłowy adres IP datagramu UDP na adres ofiary i ustawia adres rozgłoszeniowy jako docelowy (np. adres rozgłoszenia skierowanego). Odpowiednio sformowane pakiety UDP są dostarczane do usługi, która po odebraniu datagramu generuje jakąś odpowiedź. Odpowiedzi serwerów są kierowane na adres IP zawarty w polu *Źródłowy adres IP* dostarczonego pakietu UDP, czyli na adres ofiary ataku. W rezultacie opisanego działania komputer ofiary może zostać przeciążony zbyt dużą liczbą odpowiedzi UDP. Istnieje wiele odmian ataków ze wzmocnieniem. Innym przykładem jest połączenie usługi generującej znaki z usługą echa w celu nieustannego przekazywania ruchu między stacjami. Opisane ataki są zbliżone w swoim działaniu do ataku smerfów, charakterystycznego dla protokołu ICMP (więcej informacji na ten temat znajduje się w rozdziale 8.).

Fragmentacja IP również bywa wykorzystywana w różnych formach ataków. Przetwarzanie fragmentów datagramów jest nieco bardziej skomplikowane niż operowanie komunikatami protokołu UDP, większa jest więc również liczba błędów implementacyjnych, których można użyć. Jeden ze sposobów polega na wysyłaniu fragmentów, które nie zawierają żadnych danych. Atak wykorzystuje błąd mechanizmu odtwarzania datagramów IPv4 powodujący zawieszenie niektórych systemów operacyjnych. Inny atak na moduł odtwarzania datagramów nosi nazwę **kropli łzy** (*teardrop*). Polega na starannym przygotowaniu zbioru fragmentów z nakładającymi się wartościami *Przesunięcia fragmentu*, które prowadzą do zawieszenia systemu lub niewłaściwego działania. Jedna z odmian tej techniki zakłada nakładanie fragmentów w taki sposób, aby nadpisały nagłówek UDP z wcześniejszego fragmentu. Nakładanie fragmentów jest zabronione w sieciach IPv6 [RFC5722]. Również atak typu *ping of death* (wykorzystujący zazwyczaj pakiety ICMPv4 Echo Request, choć możliwy do przeprowadzenia także z pakietami UDP) bazuje na mechanizmie odtwarzania datagramów IPv4, które po złożeniu przekraczają dopuszczalny rozmiar. Sama technika ataku nie jest szczególnie skomplikowana. Pole *Przesunięcie fragmentu* może mieć maksymalną wartość 8191, odpowiadającą przesunięciu o 65 528 bajtów. Każdy fragment, którego rozmiar przekracza 7 bajtów, może więc doprowadzić do utworzenia datagramu wynikowego (po scaleniu) o rozmiarze większym niż 65 535 bajtów. Różne sposoby obrony przed tego rodzaju atakami zostały opisane w dokumencie [RFC3128].

10.15. Podsumowanie

UDP nie jest szczególnie skomplikowanym protokołem. Jego oficjalna specyfikacja [RFC0768] zajmuje jedynie trzy strony (włącznie z odniesieniami!). Jedyne realizowane przez niego zadania sprowadzają się do wyznaczania numerów portów i sumy kontrolnej (w warstwie powyżej IP). Nie zapewnia żadnych mechanizmów sterowania przepływem danych, kontrolowania przeciążeń ani korygowania błędów. Uwzględnia jednak detekcję błędów (opcjonalną w protokołach UDP/IPv4, ale obowiązkową w rozwiązaniach UDP/IPv6) oraz zachowanie granic komunikatu. W przykładach zamieszczonych w tym rozdziale protokół UDP został wykorzystany do testowania internetowych sum kontrolnych oraz sposobu działania fragmentacji IP. Przeanalizowany został również jego wpływ na różne aspekty pracy sieciowej — wyznaczanie wartości MTU trasy, projektowanie usług serwerowych oraz udział w ruchu sieciowym przekazywanym w Internecie.

Protokół UDP znajduje zastosowanie przede wszystkich w rozwiązaniach, w których trzeba ograniczyć narzut komunikacyjny związany z ustanawianiem połączenia, w których konieczne jest dostarczanie danych do wielu odbiorców jednocześnie (multiemisja i rozgłoszenia) lub nie są potrzebne relatywnie złożone funkcje protokołu TCP (np. sekwencjonowanie danych, sterowanie przepływem i retransmisja). Z powodu rozwoju aplikacji multimedialnych i komunikacji peer-to-peer stopień jego wykorzystania nieustannie rośnie. Obecnie jest podstawowym protokołem w technologii VoIP [RFC3550] [RFC3261]. Okazuje się także doskonałym mechanizmem przenoszenia ruchu przez jednostki NAT bez wprowadzania dodatkowego narzutu (jedynie 8 bajtów nagłówka UDP). Rozwiązanie to jest stosowane w systemach przejścia do komunikacji IPv6 (Teredo) oraz w mechanizmach przejścia przez NAT z wykorzystaniem techniki STUN (patrz rozdział 7.). Informacje na temat protokołu UDP znajdują się również w rozdziale 18., w części dotyczącej usługi NAT Traversal protokołu IPsec. Dużym obszarem zastosowań protokołu UDP jest również system DNS, opisany w rozdziale 11.

10.16. Bibliografia

[CT90] D. Clark, D. Tennenhouse, *Architectural Considerations for a New Generation of Protocols*, obrady ACM SIGCOMM, 1990.

[FKMC03] M. Fomenkov, K. Keys, D. Moore, K. Claffy, *Longitudinal Study of Internet Traffic in 1998 – 2003*, Raport CAIDA, dostępny pod adresem *http://www.caida.org*, 2003.

[IPORT] *http://www.iana.org/assignments/port-numbers*

[KB929851] Microsoft Support Article ID 929851, *The Default Dynamic Port Range for TCP/IP Has Changed in Windows Vista and in Windows Server 2008*, 19 listopad 2009 (wer. 6.2).

[KEWG96] F. Kaashoek, D. Engler, D. Wallach, G. Ganger, *Server Operating Systems*, obrady SIGOPS European Workshop, 1996.

[KM87] C. Kent, J. Mogul, *Fragmentation Considered Harmful*, „DEC WRL Technical Report" 87/3, 1987.

[RFC0768] J. Postel, *User Datagram Protocol*, Internet RFC 0768/STD 0006, sierpień 1980.

[RFC1122] R. Braden, red., *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1191] J. C. Mogul, S.E. Deering, *Path MTU Discovery*, Internet RFC 1191, listopad 1990.

[RFC2460] S. Deering, R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification*, Internet RFC 2460, grudzień 1998.

[RFC2675] D. Borman, S. Deering, R. Hinden, *IPv6 Jumbograms*, Internet RFC 2675, sierpień 1999.

[RFC3056] B. Carpenter, K. Moore, *Connection of IPv6 Domains via IPv4 Clouds*, Internet RFC 3056, luty 2001.

[RFC3128] I. Miller, *Protection against a Variant of the Tiny Fragment Attack (RFC 1858)*, Internet RFC 3128 (informational), czerwiec 2001.

[RFC3261] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, *SIP: Session Initiation Protocol*, Internet RFC 3261, czerwiec 2002.

[RFC3493] R. Gilligan, S. Thomson, J. Bound, J. McCann, W. Stevens, *Basic Socket Interface Extensions for IPv6*, Internet RFC 3493 (informational), luty 2003.

[RFC3550] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, Internet RFC 3550/STD 0064, lipiec 2003.

[RFC3828] L-A. Larzon, M. Degermark, S. Pink, L-E. Jonsson, red., G. Fairhurst, red., *The Lightweight User Datagram Protocol (UDP-Lite)*, Internet RFC 3828, lipiec 2004.

[RFC4213] E. Nordmark, R. Gilligan, *Basic Transition Mechanisms for IPv6 Hosts and Routers*, Internet RFC 4213, październik 2005.

[RFC4380] C. Huitema, *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)*," Internet RFC 4380, luty 2006.

[RFC4787] F. Audet, red., C. Jennings, *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*, Internet RFC 4787/BCP 0127, styczeń 2007.

[RFC4821] M. Mathis, J. Heffner, *Packetization Layer Path MTU Discovery*, Internet RFC 4821, marzec 2007.

[RFC4960] R. Stewart, red., *Stream Control Transmission Protocol*, Internet RFC 4960, wrzesień 2007.

[RFC5405] L. Eggert, G. Fairhurst, *Unicast UDP Usage Guidelines for Application Designers*, Internet RFC 5405/BCP 0145, listopad 2008.

[RFC5722] S. Krishnan, *Handling of Overlapping IPv6 Fragments*, Internet RFC 5722, grudzień 2009.

[RFC5969] W. Townsley, O. Troan, *IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) — Protocol Specification*, Internet RFC 5969, sierpień 2010.

[RFC5991] D. Thaler, S. Krishnan, J. Hoagland, *Teredo Security Updates*, Internet RFC 5991, wrzesień 2010.

[RFC6081] D. Thaler, *Teredo Extensions*, Internet RFC 6081, styczeń 2011.

[RFC6343] B. Carpenter, *Advisory Guidelines for 6to4 Deployment*, Internet RFC 6343 (informational), sierpień 2011.

[SMC02] C. Shannon, D. Moore, k claffy, *Beyond Folklore: Observations on Fragmented Traffic*, „IEEE/ACM Transactions on Networking", 10(6), grudzień 2002.

[SOCK] *http://www.icir.org/christian/sock.html*

[TTYPES] *http://www.iana.org/assignments/trailer-types*

[UNP3] W. Stevens, B. Fenner, A. Rudoff, *UNIX Network Programming*, Tom 1., Wydanie trzecie (Addison-Wesley, 2004).

[Z09] M. Zhang i in., *Analysis of UDP Traffic Usage on Internet Backbone Links*, obrady 9th Annual International Symposium on Applications and the Internet, 2009.

Rozdział 11. Odwzorowanie nazw i system nazw domenowych (DNS)

11.1. Wprowadzenie

Omawiane dotychczas protokoły umożliwiają działanie rozproszonych aplikacji dzięki identyfikowaniu stacji na podstawie adresu IP. Niestety, adresy IP (szczególnie adresy IPv6) są dość trudne do zapamiętania przez użytkowników. Dlatego w rozwiązaniach internetowych wykorzystuje się **nazwy**, które identyfikują stacje końcowe (klienckie i serwerowe). Aby zapewnić komunikację z użyciem takich protokołów jak TCP i IP, nazwy muszą być jednak odwzorowywane na adresy IP w procedurze **odwzorowania nazw**. W Internecie obowiązuje kilka mechanizmów odwzorowania nazw, ale zdecydowanie najczęściej wykorzystywany jest system rozproszonych baz danych określany jako **system nazw domenowych** (DNS — *Domain Name System*) [MD88]. Jednostki DNS działają jako aplikacje internetowe bazujące na protokołach IPv4 i IPv6. W celu zapewnienia odpowiedniej skalowalności rozwiązania nazwy domenowe są budowane w sposób hierarchiczny. Hierarchiczne są również relacje między serwerami odpowiedzialnymi za odwzorowanie nazw.

System DNS jest w zasadzie rozproszoną sieciową bazą danych o architekturze klient-serwer, wykorzystywaną przez aplikacje TCP/IP do odwzorowywania nazw stacji na ich adresy IP (i odwrotnie). Przy jego użyciu możliwe jest przekazywanie poczty elektronicznej, nazywanie usług sieciowych i wiele innych operacji. Określenie „rozproszona" oznacza, że nie ma w Internecie jednego miejsca, z którego można by pobrać wszystkie informacje na temat systemu. Każda organizacja (wydział uniwersytetu, uczelnia, firma lub oddział firmy) utrzymuje własną bazę danych i udostępnia usługi serwerowe służące innym jednostkom internetowym (klientom) do wyszukiwania informacji. Mechanizm DNS obejmuje protokół pozwalający na komunikację stacji klienckich z serwerami, a także na wymianę danych między serwerami.

Z perspektywy aplikacji dostęp do systemu DNS zapewnia specjalny moduł odwzorowania nazw nazywany **resolverem**. Przed utworzeniem połączenia TCP lub przesłaniem datagramu UDP aplikacja musi zazwyczaj sama przekształcić nazwę jednostki docelowej na odpowiadający jej adres IPv4 lub IPv6, ponieważ protokoły TCP i IP nie korzystają z systemu DNS — operują jedynie adresami IP.

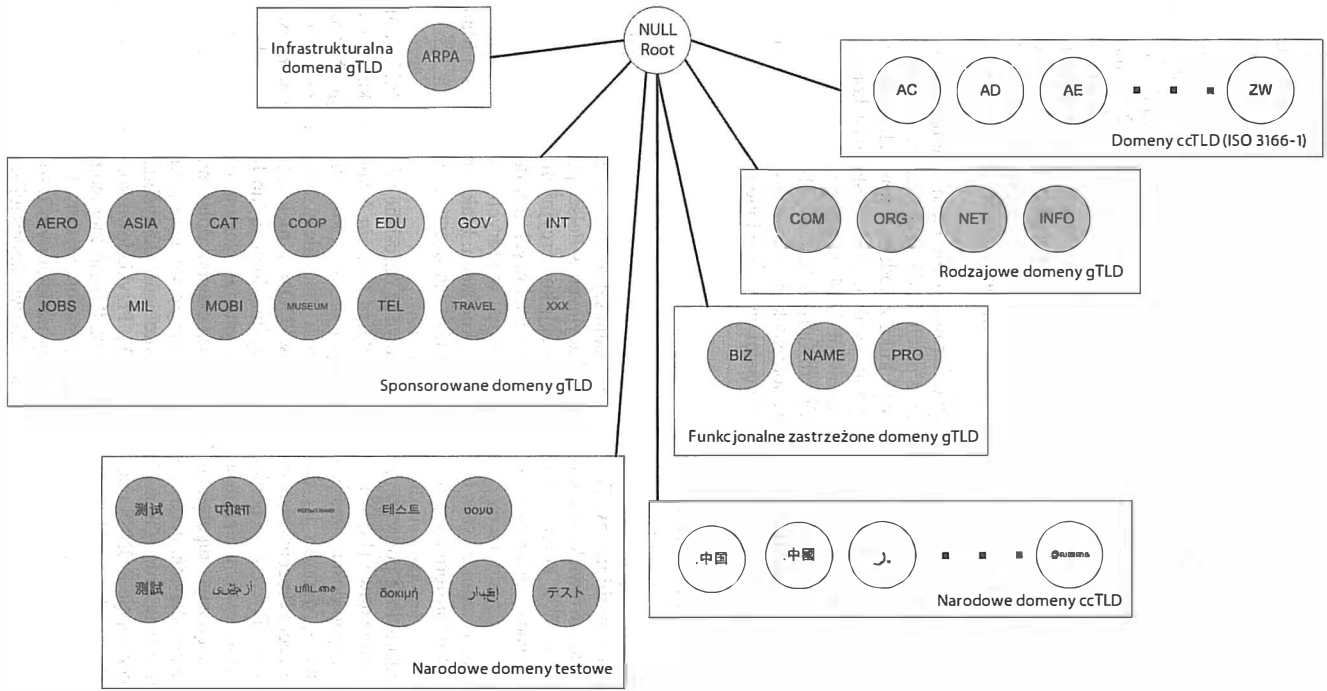
Celem tego rozdziału jest przedstawienie informacji o sposobie definiowania nazw w systemie DNS, zasadach komunikowania się resolverów i serwerów z wykorzystaniem protokołów internetowych (głównie UDP), a także o innych mechanizmach odwzorowywania nazw stosowanych w Internecie. Nie ma w nim szczegółowych opisów konfiguracji serwerów ani opcji dostępnych w aplikacjach resolvera i serwera. Dane te można pobrać z wielu ogólnie dostępnych źródeł (np. z opracowania P. Albitza i C. Liu *DNS and BIND* [AL06] oraz z zalecenia [RFC6168]). W rozdziale 18. znajduje się natomiast szczegółowe omówienie mechanizmu DNSSEC.

11.2. Przestrzeń nazw DNS

Zbiór wszystkich nazw zdefiniowanych w systemie DNS określa się mianem **przestrzeni nazw** DNS. Przestrzeń ta ma charakter hierarchiczny, a definiowane w niej nazwy nie są zależne od wielkości znaków, tak jak nazwy plików i folderów (katalogów) w niektórych systemach plików. Stosowana obecnie przestrzeń nazw ma formę drzewa domen, z korzeniem (węzłem głównym) o niezdefiniowanej nazwie na górze. Pierwszą warstwę węzłów drzewa stanowią tzw. domeny najwyższego poziomu (TLD — *Top Level Domain*), w których skład wchodzą **rodzajowe TLD** (gTLD — *generic TLD*), **krajowe TLD** (ccTLD — *country-code TLD*), **krajowe TLD w zapisie narodowym** (IDN ccTLD — *internationalized country-code TLD*) oraz specjalne **infrastrukturalne TLD** (*infrastructure TLS*) nazywane **ARPA** (ze względów historycznych) [RFC3172]. Domeny te zajmują najwyższy poziom drzewa, zgodnie z rysunkiem 11.1.

W powszechnym użyciu znajdują się domeny z pięciu grup TLD i jednej grupy domen specjalizowanych, skupiającej **narodowe nazwy domenowe** (IDN — *Internationalized Domain Names*)[1]. Historia domen IDN (przykładu umiędzynarodowienia Internetu) jest dość skomplikowana. Jak wiadomo, na świecie jest wiele języków, z których część ma własne zestawy znaków. Choć standard Unicode [U11] obejmuje wszystkie znaki narodowe, wiele z nich, mimo identycznego wyglądu, ma różne kody. Ponadto zapisywany za ich pomocą tekst może być odczytywany w kierunku od prawej do lewej strony lub od lewej do prawej strony, a także (przy połączeniu różnych fragmentów tekstu) w obu kierunkach. Zestawienie tych (i innych) problemów technicznych z zagadnieniami równouprawnienia, prawa międzynarodowego oraz aspektami politycznymi sprawia, że uzyskanie zadowalających wyników staje się bardzo trudne. Więcej informacji na ten temat znajduje się w opracowaniu organizacji IAB poświęconym domenom IDN [RFC4690], opublikowanym w 2006 roku. Bieżące dane można uzyskać w serwisie [IIDN].

[1] Na rysunku 11.1 przedstawiono również 11 testowych domen IDN, które nadal są dostępne.



Rysunek 11.1. *Przestrzeń nazw domenowych rozchodzi się od nienazwanego węzła głównego (oznaczanego też jako root), pod którym znajdują się domeny najwyższego poziomu (TLD) z domenami rodzajowymi (gTLD), domenami krajowymi (ccTLD), domenami krajowymi w zapisie narodowym (IDN ccTLD) oraz specjalnymi infrastrukturalnymi domenami ARPA*

Domeny gTLS zostały przypisane do następujących kategorii: **rodzajowe**, **rodzajowe zastrzeżone** oraz **sponsorowane**. Rodzajowe domeny gTLD (określenie **rodzajowe** występuje dwukrotnie) są przeznaczone do nieograniczonego użycia. Pozostałe (rodzajowe zastrzeżone oraz sponsorowane) zostały zarezerwowane do wykorzystania w celach wynikających z nazw samych domen. Przykładowo domena EDU jest przeznaczona dla instytucji edukacyjnych. Domeny MIL i GOV mogą być używane przez organizacje militarne i rządowe Stanów Zjednoczonych. Natomiast domeny INT należą do instytucji międzynarodowych (np. NATO). W tabeli 11.1 znajduje się zestawienie 22 domen gTLD zaczerpnięte z opracowania [GTLD] aktualnego w połowie 2011 roku. W czasie pisania książki trwały prace nad nowymi domenami gTLD, które mogą istotnie rozszerzyć dotychczasowy zbiór (prawdopodobnie do kilkuset lub nawet ponad tysiąca). Zadania rozszerzenia listy oraz zarządzania zbiorem TLD należą do Internetowej Organizacji ds. Nadawania Nazw i Numerów (ICANN — *Internet Corporation for Assigned Names and Numbers*) [ICANN].

Domeny ccTLD składają się z dwuliterowego kodu kraju zdefiniowanego w standardzie ISO 3166 [ISO3166] lub jednego z pięciu kodów (`uk`, `su`, `ac`, `eu` oraz `tp`), które nie występują w tym zaleceniu. Ponieważ kilka z kodów krajowych może mieć również inne znaczenie, część krajów wykorzystuje je również w działalności komercyjnej, sprzedając domeny krajowe instytucjom biznesowym. Doskonałym przykładem jest adres *www.rnf.fm* obejmujący kod Mikronezji (Sfederowanych Stanów Mikronezji), czyli kod wysp na Oceanie Spokojnym, które oferują domeny krajowe rozgłośniom radiowym. Tworzenie tak niekonwencjonalnych nazw jest czasami nazywane **hakerstwem domenowym**.

11.2.1. Składnia nazw DNS

Nazwy występujące w drzewie DNS poniżej poziomu TLD podlegają dalszemu grupowaniu w **poddomeny**. Technikę tę stosuje się również w odniesieniu do domen ccTLD. Przykładowo większość serwisów edukacyjnych w Polsce posługuje się nazwą zakończoną na `.edu.pl`. Z kolei większość organizacji komercyjnych korzysta z domeny `.com.pl`. W Stanach Zjednoczonych wiele witryn rządowych jest dostępnych pod adresami obejmującymi poddomeny `ci.`*`miasto.stan`*`.us`, w których ciąg *`miasto`* odpowiada nazwie miasta, a fragment *`stan`* to dwuliterowy kod stanu. I tak serwis miasta Manhattan Beach w stanie California ma adres `www.ci.manhattan-beach.ca.us`.

Przedstawione powyżej nazwy domenowe są określane jako w pełni kwalifikowanie nazwy domenowe (FQDN — *Fully Qualified Domain Name*). Na końcu ciągu takiej nazwy często dostawiana jest kropka (np. `mit.edu.`), która informuje, że nazwa jest kompletna — podczas jej odwzorowywania nie należy dołączać żadnych dodatkowych informacji. Przeciwieństwem dla ciągów FQDN są **niekwalifikowane nazwy domenowe**, które są używane w połączeniu z domyślną domeną lub listą przeszukiwanych domen definiowanych w czasie konfiguracji systemu. Nazwy niekwalifikowane są w sposób automatyczny uzupełnione dodatkowymi ciągami dołączanymi na końcu nazwy. Konfiguracja ustawień sieciowych systemu wiąże się zazwyczaj z dostarczeniem z serwera DHCP (patrz rozdział 6.) informacji o domenie domyślnej oraz liście przeszukiwanych domen (opcje RDNSS oraz DNSSL RA). Przykładowo komputery pracujące w Wydziale Telekomunikacji i Elektroniki Politechniki Poznańskiej mogłyby otrzymywać informacje o tym, że pracują domyślnie w domenie `et.put.poznan.pl`. Wówczas wpisanie w jednym z systemów nazwy `serwer` wymusiłoby na lokalnym resolverze odwzorowanie nazwy FQDN `serwer.et.put.poznan.pl.` na odpowiadający jej adres IP.

Tabela 11.1. *Rodzajowe domeny najwyższego poziomu (gTLD)*

TLD	Pierwsze użycie (szacunkowo)	Przeznaczenie	Przykład
AERO	21 grudnia 2001	Przemysł lotniczy	*www.bzg.aero*
ARPA	1 stycznia 1985	Infrastruktura Internetu	*18.in-addr.arpa*
ASIA	2 maja 2007	Azja i azjatycki obszar Pacyfiku	*www.seo.asia*
BIZ	26 czerwca 2001	Wykorzystanie w biznesie	*neustar.biz*
CAT	19 grudnia 2005	Katalońska społeczność językowa (kulturowa)	*www.domini.cat*
COM	1 stycznia 1985	Ogólne przeznaczenie	*icanhascheezburger.com*
COOP	15 grudnia 2001	Stowarzyszenia współpracujących firm	*www.ems.coop*
EDU	1 stycznia 1985	Policealne instytucje edukacyjne w Stanach Zjednoczonych	*hpu.edu*
GOV	1 stycznia 1985	Organizacje rządowe w Stanach Zjednoczonych	*whitehouse.gov*
INFO	25 czerwca 2001	Ogólne przeznaczenie	*germany.info*
INT	3 listopada 1988	Organizacje międzynarodowe	*nato.int*
JOBS	8 września 2005	Zarządzanie zasobami ludzkimi	*intel.jobs*
MIL	1 stycznia 1985	Organizacje militarne w Stanach Zjednoczonych	*dtic.mil*
MOBI	30 października 2005	Klienci i producenci urządzeń i usług mobilnych	*jestem.mobi*
MUSEUM	30 października 2001	Muzea	*chopin.museum*
NAME	16 sierpnia 2001	Osoby prywatne	*www.poznan.name*
NET	1 stycznia 1985	Ogólne przeznaczenie	*mikrofale.net*
ORG	9 grudnia 2002	Ogólne przeznaczenie	*www.joemonster.org*
PRO	6 maja 2002	Certyfikowani specjaliści i firmy	*nic.pro*
TEL	1 marca 2007	Dane kontaktowe dla firm i osób prywatnych	*telnic.tel*
TRAVEL	27 lipca 2005	Przemysł turystyczny	*cancun.travel*
XXX	15 kwietnia 2011	Przemysł erotyczny	*whois.nic.xxx*

Nazwa domenowa składa się ze zbioru **etykiet** rozdzielonych kropkami. Za ich pomocą określane jest położenie nazwy w drzewie. Każda kropka stanowi separator poziomów drzewa, a analiza zapisu od prawej do lewej strony odpowiada przejściu przez kolejne poziomy drzewa od góry do dołu. Przykładowo nazwa FQDN

```
www.net.in.tum.de.
```

obejmuje etykietę nazwy komputera (`www`) określoną na czwartym poziomie drzewa domen (`net.in.tum.de.`). Analizując zapis od prawej strony można wywnioskować, że pod węzłem głównym (root) występuje domena TLD o wartości `de` (czyli niemiecka

domena ccTLD), a następnie kolejno skrót nazwy uniwersytetu Technische Universität München (tum), skrót nazwy wydziału informatycznego (in) oraz nazwa grupy urządzeń sieciowych na wydziale informatyki (net). Wielkość liter w poszczególnych etykietach nie ma znaczenia. Zatem zapis ACME.COM jest tożsamy z acme.com oraz AcMe.cOm [RFC4343]. Każda z etykiet może się składać z 63 znaków, a cały ciąg FQDN jest ograniczony do 255 (1-bajtowych) znaków. I tak ciąg:

```
thelongestdomainnameintheworldandthensomeandthensomemoreandmore.com
```

został rzekomo zgłoszony do Księgi Rekordów Guinessa jako najdłuższa nazwa domenowa (o etykiecie złożonej z 63 znaków), ale nie został zarejestrowany, ponieważ komisja kwalifikacyjna uznała, że nie zasługuje na wpis do księgi.

Hierarchiczna struktura przestrzeni nazw DNS umożliwia przekazanie kontroli nad poszczególnymi jej fragmentami różnym organizacjom zarządzającym. Utworzenie np. nowego wpisu DNS o treści it.wsg.byd.pl wymaga uzgodnienia takiej modyfikacji jedynie z właścicielem poddomeny wsg.byd.pl. Obszary byd.pl i pl nie wymagają wprowadzania żadnych modyfikacji, więc ich właściciele nie muszą być angażowani w wykonanie zadania. Cecha ta jest jednym z kluczowych elementów zapewniających **skalowalność** rozwiązania. Nie ma jednego centralnego urzędu, który musiałby zarządzać całą przestrzenią DNS i wprowadzać w niej wszystkie zmiany. W praktyce utworzenie hierarchicznej struktury nazw było jednym z pierwszych przypadków reakcji społeczności internetowej na dążenie do uelastycznienia wykorzystywanych mechanizmów. Pierwotny system nazw Internetu miał charakter jednowymiarowego rejestru (bez hierarchii). Pojedyncza organizacja była odpowiedzialna za przyznawanie nazw, utrzymywanie i dystrybuowanie listy niekolidujących ze sobą nazw. Z czasem, gdy trzeba było wprowadzać coraz więcej modyfikacji, rozwiązanie stało się nieefektywne [MD88].

11.3. Serwery nazw i strefy

Zarządzanie wybranymi obszarami przestrzeni nazw DNS należy do pojedynczych administratorów lub organizacji. Osoba odpowiedzialna za utrzymywanie określonej części aktywnej przestrzeni nazw DNS (jednej domeny lub większej liczby domen) musi zapewnić dostępność co najmniej dwóch **serwerów nazw** (**serwerów DNS**) przechowujących informacje na temat tej przestrzeni, z których mogą korzystać użytkownicy Internetu (wysyłając żądania DNS). Zbiór serwerów tworzy usługę DNS — system, którego podstawowe zadanie polega na odwzorowywaniu nazw na adresy. Choć często serwery DNS oferują również wiele dodatkowych informacji.

W terminologii DNS określona jednostka administracyjna jest nazywana **strefą**. Strefa odpowiada fragmentowi przestrzeni nazw, który może być zarządzany niezależnie od innych stref. Każda nazwa domenowa jest zdefiniowana w pewnej strefie. Przykładowo domeny TLD są opisane w **strefie głównej** (*root zone*). Gdy do strefy dodawany jest nowy rekord, administrator strefy musi wpisać nazwę domenową i towarzyszące jej informacje (zazwyczaj adres IP) do bazy danych serwera nazw. W niewielkich sieciach zadanie to może wykonywać jedna osoba (np. za każdym razem, gdy włączany jest nowy serwer). Jednak w dużych korporacjach zarządzanie systemem jest delegowane do odpowiednich wydziałów lub jednostek organizacyjnych, ponieważ pojedyncza osoba nie poradziłaby sobie z ilością pracy.

Serwer DNS może przechowywać informacje o większej liczbie stref. Każdy punkt zmiany w nazwie domenowej (każda kropka) może również wyznaczać nową strefę i nowy serwer udostępniający informacje na temat tej nazwy. Mechanizm przekazywania odpowiedzialności za poszczególne obszary nazywa się **delegacją**. Delegacje są często wykorzystywane w strefach opisujących domeny drugiego poziomu (takich jak `berkeley.edu`). W danej domenie mogą zostać zdefiniowane pojedyncze stacje (np. `www.berkeley.edu`) lub inne domeny (np. `cs.berkeley.edu`). Każda strefa ma swojego zarządcę lub organizację odpowiedzialną za zarządzanie nazwami, adresami i strefami podrzędnymi w stosunku do danej strefy. Często wspomniana osoba operuje nie tylko danymi strefy, ale również serwerami nazw, które przechowują bazę danych (lub bazy danych) strefy.

Ze względów niezawodnościowych informacje na temat strefy są przechowywane w co najmniej dwóch miejscach (ponieważ dane strefy muszą być składowane w nie mniej niż dwóch serwerach). Jeśli jeden serwer przestanie działać, drugi serwer będzie mógł przejąć jego zadania. Wszystkie serwery muszą przechowywać identyczne informacje na temat stref. Zazwyczaj wyróżnia się serwer **podstawowy** utrzymujący bazę danych strefy w pliku dyskowym oraz co najmniej jeden serwer **zapasowy**, który otrzymuje kopię bazy danych z serwera podstawowego. Proces dystrybuowania danych nazywa się **transferem strefy**. Do wykonania transferu strefy służy specjalny protokół, choć można to zadanie zrealizować również za pomocą innych mechanizmów (np. narzędzia `rsync` [RSYNC]).

11.4. Buforowanie

Serwery nazw przechowują powiązania między nazwami i adresami. Dane te są pozyskiwane z trzech źródeł — z bazy danych strefy, w wyniku transferu strefy (np. w serwerze podrzędnym) oraz z innego serwera DNS w wyniku operacji odwzorowania adresu. Pierwszy przypadek dotyczy serwera przechowującego **autorytatywne** informacje na temat strefy. Serwer taki jest nazywany **serwerem autorytatywnym** i jest wymieniony z nazwy w danych strefy.

Większość serwerów nazw (z wyjątkiem serwerów głównych i TLD) **buforuje** pozyskane informacje o strefach i utrzymuje te dane do upłynięcia **czasu ważności** (TTL — *Time To Live*) określonego wpisu. Zbuforowane informacje służą serwerom do udzielania odpowiedzi na kolejne zapytania. Wykorzystanie danych zapisanych w pamięci podręcznej znacznie zmniejsza natężenie ruchu DNS w Internecie [J02]. Jednak odpowiadając na żądanie, serwer musi dołączyć informację o tym, czy dane pochodzą z bufora, czy zostały pozyskane z autorytatywnego źródła informacji o strefie. Często do danych pochodzących z bufora dodawane są nazwy serwerów DNS, z którymi można się skontaktować w celu uzyskania autorytatywnych informacji o strefie.

Każdemu rekordowi DNS (powiązaniu między nazwą i adresem IP) przypisany jest parametr TTL oznaczający dopuszczalny czas przechowywania w buforze. Wartości TTL są ustawiane i modyfikowane przez administratora strefy. Ponieważ decydują one o czasie utrzymywania poszczególnych rekordów w pamięci podręcznej serwerów DNS, wprowadzenie zmian w konfiguracji strefy nie zawsze jest odzwierciedlane w sieci. Trzeba pamiętać o tym, że do czasu wygaśnięcia ważności rekordów niektóre z serwerów mogą korzystać z wcześniejszych informacji, a to z kolei może prowadzić do niewłaściwego

odwzorowania nazw. Dlatego część administratorów przed wprowadzeniem zaplanowanych modyfikacji zmniejsza wartość TTL rekordów objętych zmianami. Okres udostępniania nieprawdziwych informacji w sieci jest wówczas istotnie skrócony.

Warto przy tej okazji wspomnieć, że buforowanie jest niezależne od tego, czy odwzorowanie zostało zakończone pomyślnie, czy niepomyślnie (**buforowanie negatywne**). Jeśli zadanie zamiany określonej nazwy domenowej zakończy się niepowodzeniem (nie zostanie zwrócony żaden rekord), fakt ten również jest rejestrowany w pamięci podręcznej. Taki sposób postępowania pozwala na ograniczenie ruchu internetowego w przypadku, w którym błędnie działająca aplikacja okresowo ponawia żądania odwzorowania niezdefiniowanej nazwy. W zaleceniu [RFC2308] buforowanie negatywne zostało uznane za obowiązkowe (wcześniej miało charakter opcjonalny).

W niektórych konfiguracjach sieciowych (np. bazujących na starszych systemach operacyjnych UNIX) pamięć podręczna jest utrzymywana w serwerze nazw, a nie w oprogramowaniu resolvera działającego w jednostkach klienckich. Umieszczenie bufora na serwerze umożliwia współdzielenie zarejestrowanych informacji przez wszystkie komputery pracujące w sieci LAN, ale wnosi również dodatkowe opóźnienie w odwołaniu do danych (związane z koniecznością odwołania się do serwera w sieci lokalnej). W systemach Windows i nowszych systemach uniksowych (w tym w Liunksach) program kliencki dysponuje własnym buforem, który jest dostępny dla wszystkich aplikacji działających w tym systemie. W środowisku Windows mechanizm buforowania jest włączany automatycznie. W systemach Linux ma formę usługi, którą można włączyć lub wyłączyć.

Działanie lokalnego mechanizmu buforowania w systemie Windows jest kontrolowane przez następujący wpis w rejestrze:

```
HKLM\SYSTEM\CurrentControlSet\Services\DNSCache\Parameters
```

Zapisana w nim wartość `MaxNegativeCacheTtl` (typu `DWORD`) określa maksymalny czas (wyrażony w sekundach) przechowywania negatywnej odpowiedzi w pamięci podręcznej resolvera DNS. Z kolei wartość `MaxCacheTtl` (również typu `DWORD`) definiuje maksymalny czas (wyrażony w sekundach) utrzymywania w buforze prawidłowego rekordu DNS. Jeśli zdefiniowane w ten sposób okresy ważności są różne od wartości TTL odebranego rekordu DNS, za obowiązującą uznawana jest mniejsza wartość. Żaden z wymienionych kluczy rejestru nie jest definiowany domyślnie przez system. Żeby skorzystać z opisanych funkcji, trzeba samodzielnie utworzyć odpowiednie wpisy.

W systemach Linux buforowanie po stronie klienta zapewnia usługa *Name Service Caching Daemon* (NSCD) (usługa ta jest dostępna również w innych systemach operacyjnych poza Linuksem). Sposób jej działania jest opisany w pliku konfiguracyjnym */etc/nscd.conf*, w którym można wskazać rodzaje operacji odwzorowania podlegające buforowaniu (w systemie DNS i innych) oraz określić parametry buforowania, takie jak TTL. Ponadto w pliku */etc/nsswitch.conf* zdefiniowany jest przebieg samej operacji odwzorowania nazw. Można w nim określić, czy do ustalenia adresu będą wykorzystywane pliki lokalne, protokół DNS (patrz punkt 11.5), czy mechanizm NSCD.

11.5. Protokół DNS

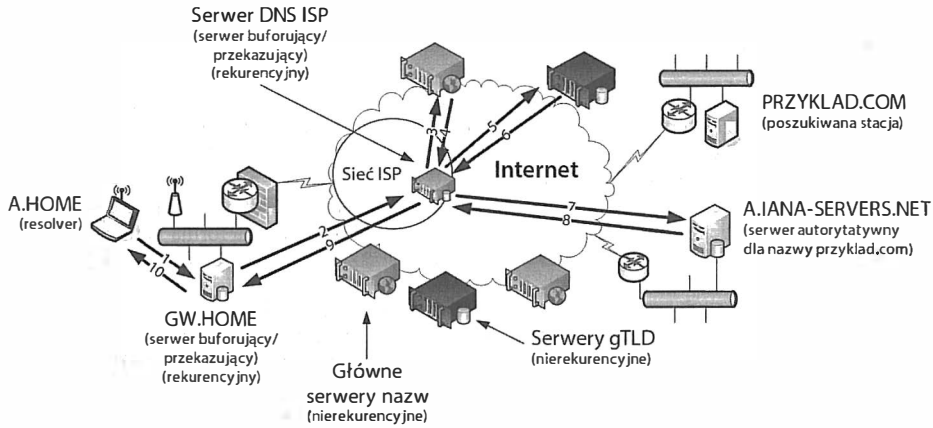
Protokół DNS składa się z dwóch komponentów — protokołu obsługującego żądania i odpowiedzi w operacjach odwzorowania nazw w systemie DNS oraz protokołu wymiany baz rekordów między serwerami nazw (transferu stref). Protokół DNS uwzględnia również funkcje powiadamiania serwerów zapasowych o wystąpieniu zmian w bazie danych strefy i konieczności wykonania transferu strefy (funkcja DNS NOTIFY) oraz funkcję dynamicznej aktualizacji strefy. Zdecydowanie najczęściej wykorzystywaną formą komunikacji jest przesyłanie żądań i odpowiedzi w celu zamiany nazwy domenowej na adres IPv4.

Choć proces odwzorowania nazw DNS oznacza najczęściej udostępnienie adresu IPv4 odpowiadającego podanej nazwie domenowej, analogiczne działanie jest realizowane również w odniesieniu do adresów IPv6. W operacje żądania-odpowiedzi DNS angażowane są rozproszone serwery systemu DNS, a szczególnie serwery lokalne (działające w ramach sieci lokalnej lub sieci operatora ISP), specjalna grupa **serwerów głównych** oraz zbiór **serwerów domen najwyższego poziomu**, które usprawniają obsługę większych domen gTLD (szczególnie COM i NET). Od połowy 2011 roku funkcjonuje trzynaście głównych serwerów nazw oznaczonych literami od *A* do *M* (więcej informacji na ich temat znajduje się w opracowaniu [ROOTS]). Dziewięć z nich dysponuje adresami IPv6. Wyznaczono również trzynaście serwerów gTLD, także o oznaczeniach od *A* do *M*. Dwa z tych serwerów operują adresami IPv6. Odwołania do serwerów głównych i serwerów gTLD pozwalają na ustalenie adresu każdego serwera TLD w Internecie. Prace wszystkich wymienionych jednostek są ze sobą skorelowane w taki sposób, aby udostępniane informacje miały taką samą treść. W niektórych przypadkach usługa nie jest realizowana przez jeden serwer fizyczny, ale przez kilka jednostek (w przypadku serwera głównego *J* jest ich ponad 50) o tym samym adresie IP (z wykorzystaniem adresacji IP typu anycast; patrz rozdział 2.).

Odwzorowanie, które nie może zostać zrealizowane z wykorzystaniem informacji zarejestrowanych wcześniej w buforze, angażuje kilka jednostek DNS. Przebieg komunikacji został przedstawiony na rysunku 11.2.

Na rysunku widoczny jest laptop o nazwie A.HOME pracujący w sieci serwera DNS o nazwie GW.HOME. Domena DOM jest domeną prywatną, więc nie jest znana w Internecie — znają ją tylko użytkownicy sieci lokalnej. Gdy użytkownik komputera A.HOME chce nawiązać połączenie ze stacją PRZYKLAD.COM (np. po wpisaniu w polu przeglądarki adresu *http://przyklad.com*), komputer A.HOME musi ustalić adres IP serwera PRZYKLAD.COM. Przy założeniu, że adres ten nie został wcześniej zarejestrowany w pamięci podręcznej, oprogramowanie resolvera w systemie użytkownika rozpoczyna działanie od przesłania zapytania do lokalnego serwera nazw GW.HOME. Zapytanie to jest żądaniem przekształcenia nazwy PRZYKLAD.COM na odpowiedni adres IP (stosowny komunikat został na rysunku oznaczony 11.2 cyfrą 1).

Jeśli w systemie A.HOME zdefiniowano listę przeszukiwanych domen, opisanemu żądaniu mogą towarzyszyć dodatkowe odwołania do systemu DNS. Jeśli np. domenę .HOME ustawiono jako domenę domyślną, pierwsze zapytanie może się odnosić do nazwy PRZYKLAD.COM.HOME. Nie zostanie ono poprawnie wykonane, ponieważ serwer nazw GW.HOME (serwer autorytatywny dla domeny .HOME) zwróci informację o błędzie. W kolejnym żądaniu domyślne rozszerzenie nazwy zostanie usunięte, co spowoduje wygenerowanie zapytania o nazwę PRZYKLAD.COM.



Rysunek 11.2. *Typowe rekurencyjne zapytanie o nazwę PRZYKLAD.COM wysłane ze stacji A.HOME wymaga przesłania aż dziesięciu komunikatów. Lokalny serwer rekurencyjny (GW.HOME) wykorzystuje serwer DNS operatora ISP. Ten z kolei, poszukując domeny PRZYKLAD.COM, odwołuje się do głównego serwera nazw oraz serwera gTLD (obsługującego domeny TLD COM i NET). Ostatni z wymienionych serwerów (w tym przypadku A.IANA-SERVERS.NET) udostępnia adres IP stacji o nazwie PRZYKLAD.COM. Serwery rekurencyjne buforują uzyskane informacje na potrzeby przyszłego odwzorowania*

Jeśli jednostka GW.HOME nie przechowuje w buforze adresu IP stacji PRZYKLAD.COM ani adresu serwera nazw obsługującego domenę PRZYKLAD.COM lub COM, przekazuje żądanie do kolejnego serwera DNS (działanie **rekurencyjne**). W takim przypadku zapytanie (komunikat 2) trafia do serwera DNS dostawcy usług internetowych. Jeżeli poszukiwany adres nie występuje w buforze serwera, następnym krokiem jest odwołanie się do serwera głównego (komunikat 3). Serwery główne nie działają rekurencyjnie, więc nie mogą przekazywać żądania do kolejnych serwerów. Jednak w tym przypadku serwer główny odeśle odpowiedź zawierającą informacje potrzebne do odwołania się do serwera nazw obsługującego domenę COM. Wynikiem może być np. nazwa A.GTLD-SERVERS.NET oraz odpowiadający jej adres IP (lub większa liczba adresów) (komunikat 4). Dzięki uzyskanym informacjom serwer DNS operatora ISP może skontaktować się z serwerem gTLD (komunikat 5) i uzyskać nazwę oraz adres IP serwera nazw obsługującego domenę PRZYKLAD.COM (komunikat 6). W analizowanym przykładzie jednym z takich serwerów jest jednostka o nazwie A.IANA-SERVERS.NET.

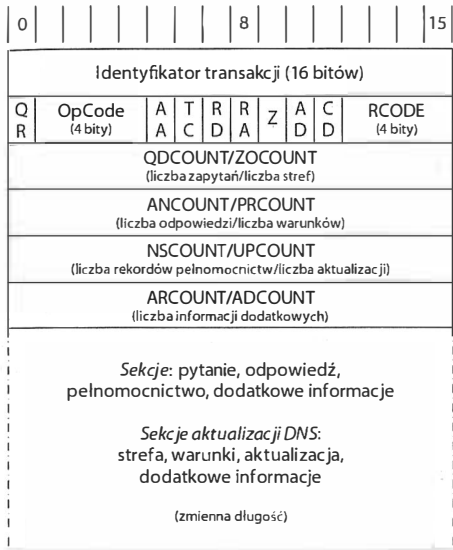
Znając adres serwera obsługującego domenę, serwer dostawcy usług internetowych kontaktuje się z odpowiednim serwerem DNS (komunikat 7) i uzyskuje poszukiwany adres IP (komunikat 8). Na tym etapie procedury serwer ISP może odesłać stosowne informacje do komputera GW.HOME (komunikat 9). Stacja GW.HOME może wówczas wysłać żądanie pod uzyskany adres IPv4 i (lub) IPv6 (komunikat 9) oraz odebrać odpowiedź z komputera o danym adresie (komunikat 10).

Z perspektywy stacji A.HOME mogłoby się wydawać, że żądanie zostało zrealizowane przez lokalny serwer nazw. W rzeczywistości jednak wykonane zostało **zapytanie rekurencyjne**, w którym serwery GW.HOME i jednostka ISP musiały wygenerować kilka dodatkowych żądań. Trzeba jednak pamiętać, że serwery główne oraz serwery TLD nie mają prawa wykonywać zapytań rekurencyjnych. Stanowią one bardzo ważne zasoby sieci internetowej,

więc obciążanie ich zapytaniami rekurencyjnymi dostarczanymi przez różne stacje kliencki mogłoby doprowadzić do znacznego obniżenia wydajności komunikacji internetowej.

11.5.1. Format komunikatu DNS

Komunikaty DNS mają jeden format opisany w dokumencie [RFC6195] i przedstawiony na rysunku 11.3. Jest on stosowany we wszystkich operacjach DNS (zapytaniach, odpowiedziach, transferach stref, powiadomieniach i dynamicznych aktualizacjach).



Rysunek 11.3. *Komunikat DNS obejmuje stały 12-bajtowy nagłówek. Cała wiadomość jest zazwyczaj przenoszona w jednym datagramie UDP/IPv4, a jej rozmiar nie przekracza 512 bajtów. Komunikaty DNS UPDATE (dynamiczne aktualizacje DNS) przenoszą pola o nazwach ZOCOUNT, PRCOUNT, UPCOUNT oraz ADCOUNT. Specjalny format rozszerzeń (o nazwie EDNS0) umożliwia przesyłanie komunikatów o rozmiarach większych niż 512 bajtów, wymaganych w rozwiązaniu DNSSEC (patrz rozdział 18.)*

Podstawowy komunikat DNS rozpoczyna się od stałego 12-bajtowego nagłówka, po którym następują cztery **sekcje** o zmiennej długości — sekcja zapytania, sekcja odpowiedzi, rekordy pełnomocnictw oraz rekordy dodatkowe. Wszystkie sekcje oprócz pierwszej zawierają co najmniej jeden **rekord zasobu** (RR — *Resource Record*), omówiony w punkcie 11.5.6 (sekcja zapytania obejmuje dane o strukturze bardzo zbliżonej do rekordu RR). Rekordy RR można buforować, natomiast zapytań buforować nie wolno.

Zapisane w nagłówku pole *Identyfikator transakcji* jest ustawiane przez klienta i zwracane w niezmienionej formie przez serwer. Umożliwia ono powiązanie odpowiedzi z żądaniami przez stację kliencką. Kolejne 16-bitowe słowo definiuje kilka znaczników. Najbardziej znaczący bit pola (QR) przyjmuje wartości: 0 w przypadku przekazywania żądania i 1 podczas transmisji odpowiedzi. Następne pole (*OpCode*) to 4-bitowa liczba, która w przypadku standardowego zapytania oraz standardowej odpowiedzi ma wartość 0.

Wartości od 1 do 3 są uznawane za przedawnione (nie są używane). Natomiast 4 i 5 oznaczają odpowiednio powiadomienie i aktualizację. Kolejny bit (AA) wskazuje „odpowiedź autorytatywną" (odróżniającą wynik odwołania do serwera autorytatywnego od pobranego z bufora). Bit TC oznacza podział komunikatu. Ustawienie znacznika TC w komunikacie UDP oznacza, że całkowity rozmiar datagramu przekracza 512 bajtów oraz że zwróconych zostało tylko 512 bajtów odpowiedzi.

Bit RD informuje, że wymagane jest wykonanie odwzorowania rekurencyjnego. Może on zostać ustawiony w komunikacie żądania oraz odpowiedzi. Wymusza na serwerze działanie rekurencyjne. Jeśli bit nie jest ustawiony, a serwer nie dysponuje autorytatywną odpowiedzią, odpytywany serwer zwraca listę innych serwerów nazw, które można wykorzystać w operacji odwzorowania nazwy. W reakcji na taką odpowiedź stacja może kontynuować wykonywanie zapytania, kierując żądania do kolejnych serwerów z listy. Taka forma zapytania jest nazywana **zapytaniem iteracyjnym**. Ustawienie bitu RA oznacza „możliwość działania rekurencyjnego". Znacznik ten jest ustawiany w odpowiedziach serwerów obsługujących zapytania rekurencyjne. Serwery główne **nie pracują w trybie rekurencyjnym**, zmuszając jednostki klienckie do wykonywania zapytań rekurencyjnych w dalszej fazie procesu odwzorowania nazw. Bit Z musi mieć obecnie wartość zero, ale planowane jest przyszłe jego wykorzystanie.

Bit AD ma wartość 1, jeśli przenoszona informacja została **uwierzytelniona**. Z kolei bit CD ma wartość 1, gdy mechanizm weryfikacji zabezpieczeń jest wyłączony (patrz rozdział 18.). Pole **kodów odpowiedzi** (RCODE — *Response Code*) składa się z 4 bitów odzwierciedlających wartości zdefiniowane w dokumencie [DNSPARAM]. Najczęściej występującymi wartościami są 0 (brak błędu) oraz 3 (błąd w nazwie lub nieistniejąca domena; zapisywane jako NXDOMAIN). Wykaz pierwszych 11 kodów błędu został zamieszczony w tabeli 11.2 (wartości od 11 do 15 nie zostały określone). Dodatkowe typy są definiowane za pomocą specjalnych rozszerzeń (opisanych w punkcie 11.5.2). Informacja o błędzie w nazwie jest zwracana tylko przez autorytatywny serwer nazw, gdy nazwa domenowa podana w zapytaniu nie istnieje.

Cztery kolejne pola to 16-bitowe wartości określające liczbę wpisów przenoszonych w sekcjach zapytania, odpowiedzi, pełnomocnictw oraz informacji dodatkowych. W przypadku żądania liczba zapytań wynosi zazwyczaj 1, a pozostałe liczniki mają wartość 0. W odpowiedzi liczba wpisów wynosi co najmniej 1. Każde zapytanie ma określoną nazwę, typ i klasę (klasy identyfikują rekordy inne niż internetowe; informacje o klasach będą pomijane w dalszych analizach; do rozróżniania zapytań będą wykorzystywane typy). Wszystkie pozostałe sekcje mogą zawierać kilka rekordów RR, ale mogą też być puste. Każdemu rekordowi RR odpowiadają pewna nazwa, typ i klasa informacji, a także wartość TTL, która wyznacza czas przechowywania danych w pamięci podręcznej. Omówienie najważniejszych typów rekordów zasobów znajduje się bezpośrednio za opisem zasad zapisywania nazw w systemie DNS oraz korzystania z protokołów transportowych do przenoszenia komunikatów.

11.5.1.1. Nazwy i etykiety

Końcowa część komunikatu DNS składa się ze zbioru zapytań, odpowiedzi, informacji o pełnomocnictwach (nazw serwerów DNS przechowujących autorytatywne informacje) oraz dodatkowych danych. Pola te nie mają ustalonego rozmiaru i pozwalają na ograniczenie

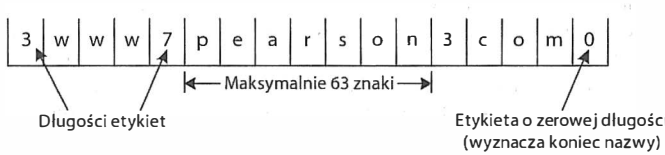
Tabela 11.2. *Jedenaście pierwszych typów błędów opisywanych za pomocą pola RCODE*

Wartość	Nazwa	Specyfikacja	Opis i przeznaczenie
0	NoError	[RFC1035]	Brak błędu
1	FormErr	[RFC1035]	Błąd formatu; nie można zinterpretować zapytania
2	ServFail	[RFC1035]	Błąd serwera; błąd podczas przetwarzania zapytania po stronie serwera
3	NXDomain	[RFC1035]	Nieistniejąca domena; wskazana domena nie jest znana serwerowi
4	NotImp	[RFC1035]	Niezaimplementowane; dany rodzaj żądania nie jest obsługiwany po stronie serwera
5	Refused	[RFC1035]	Odrzucone; serwer nie dostarczy odpowiedzi
6	YXDomain	[RFC2136]	Nazwa istnieje, mimo że nie powinna (używane w aktualizacjach)
7	YXRRSet	[RFC2136]	Zbiór RRSet istnieje, mimo że nie powinien (używane w aktualizacjach)
8	NXRRSet	[RFC2136]	Zbiór RRSet nie istnieje, mimo że powinien (używane w aktualizacjach)
9	NotAuth	[RFC2136]	Serwer nieautoryzowany w danej strefie (używane w aktualizacjach)
10	NotZone	[RFC2136]	Nazwa nie należy do strefy (używane w aktualizacjach)

liczby wysyłanych żądań. Każde zapytanie i każdy rekord RR rozpoczynają się od **nazwy** (zwanej nazwą domenową lub nazwą właściciela), do której się odnoszą. Każda nazwa składa się z szeregu **etykiet**. Etykiety są z kolei podzielone na dwie kategorie, **etykiety danych** oraz **etykiety kompresji**. Etykiety danych są zapisywane z użyciem znaków. Etykiety kompresji pełnią rolę wskaźników na inne etykiety. Pozwalają na zmniejszenie rozmiaru komunikatu DNS, gdy przenosi on kilka kopii tego samego ciągu tekstowego.

11.5.1.2. Etykiety danych

Każda etykieta danych rozpoczyna się od jednobajtowego licznika, który określa, ile bajtów następuje zaraz za nim samym. Znacznikiem końca nazwy jest bajt o wartości 0. Zero oznacza również etykietę o zerowej długości (czyli etykietę węzła głównego w drzewie nazw). Przykładowo nazwa `www.pearson.com` jest kodowana w sposób przedstawiony na rysunku 11.4.

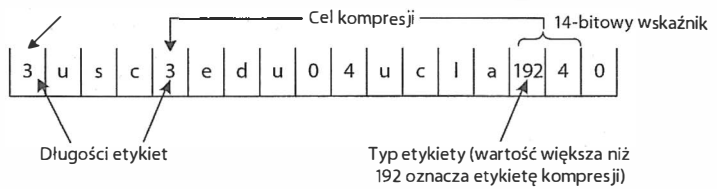


Rysunek 11.4. *Nazwy DNS są kodowane jako ciągi etykiet. W przykładzie przedstawiono sposób zapisu nazwy www.pearson.com, która z technicznego punktu widzenia składa się z czterech etykiet. Zakończenie nazwy wyznacza etykieta o zerowej długości, odpowiadająca węzłowi głównemu*

Bajt *Długości* w etykietach danych musi mieć wartość z przedziału od 0 do 63 (długość etykiety jest ograniczona do 63 bajtów). Tekstowi nie może towarzyszyć żadne dopełnienie, więc całkowita długość nazwy może być wartością nieparzystą. Choć opisywane etykiety są nazywane czasami etykietami „tekstowymi", mogą przechowywać również znaki spoza zestawu ASCII. Takie rozwiązanie jest jednak rzadko stosowane i odradzane. W praktyce nawet umiędzynarodowione nazwy domenowe (które mogą zawierać znaki Unicode) [RFC5890][RFC5891] bazują na dość oryginalnej składni kodowania (punycode) [RFC3492] zapewniającej przedstawienie znaków Unicode w formie znaków zestawu ASCII. Gdy chcemy mieć stuprocentową pewność, że zdefiniowana nazwa zostanie uznana za poprawną, „należy rozpoczynać każdą etykietę literą, kończyć literą lub cyfrą i używać w części środkowej jedynie liter, cyfr i znaków podkreślenia".

11.5.1.3. Etykiety kompresji

Dane przenoszone w sekcjach odpowiedzi, pełnomocnictw oraz informacji dodatkowych są zazwyczaj powiązane z jedną nazwą domenową. Oznacza to, że te same etykiety danych są wielokrotnie powtarzane w treści odpowiedzi DNS. Aby uniknąć nadmiarowości i zmniejszyć ilość transmitowanych bajtów, wykorzystuje się mechanizm kompresji. Jak wiadomo, każda etykieta jest poprzedzana informacją o jej długości (która może się zawierać w przedziale od 0 do 63 bajtów). W przypadku zastosowania kompresji dwa najbardziej znaczące bity licznika mają wartość 1, a pozostałe bity pierwszego bajta wraz z bitami kolejnego bajta stanowią 14-bitowy wskaźnik do określonego obszaru komunikatu DNS. Wskaźnik ten określa pozycję właściwej etykiety danych (**celu kompresji**) liczonej w bajtach od początku komunikatu. Etykiety kompresji mogą więc wskazywać bajt oddalony o 16 383 bajty od początkowego bajta wiadomości. Na rysunku 11.5 przedstawiono sposób wykorzystania etykiet kompresji do zapisania nazw domenowych `usc.edu` i `ucla.edu`.



Rysunek 11.5. *Etykieta kompresji może wskazywać inne etykiety, co pozwala na skrócenie komunikatu. Wymaga to ustawienia dwóch najbardziej znaczących bitów bajta poprzedzającego samą etykietę. Dzięki temu wiadomo, że 14 kolejnych bitów wyznacza położenie etykiety względem początku komunikatu. W prezentowanym przykładzie w nazwach usc.edu i ucla.edu wyodrębniono wspólną etykietę edu*

Na rysunku 11.5 przedstawiono sposób wykorzystania jednej etykiety `edu` w dwóch nazwach domenowych. Załóżmy, że przesunięcie nazw względem początku komunikatu wynosi 0, a nazwa `usc.edu` została zapisana za pomocą etykiet danych (tak jak w poprzednio analizowanym przykładzie). W takim przypadku nazwa `ucla.edu` może zostać przedstawiona za pomocą etykiety danych `ucla` oraz etykiety `edu` współdzielonej z nazwą `usc.edu`. W tym celu wystarczy ustawić dwa najbardziej znaczące bity bajta długości na 1 i zakodować na kolejnych 14 bitach przesunięcie etykiety `edu` względem początku komunikatu. Ponieważ pierwsze wystąpienie ciągu `edu` znajduje się na pozycji 4, zadanie sprowadza się do zapisania wartości 192 (6 bitów zerowych) w pierwszym bajcie oraz

liczby 4 w drugim bajcie. W przykładzie zaprezentowanym na rysunku 11.5 zaoszczędzono jedynie 4 bajty, ale kompresja dłuższych etykiet może dawać znacznie lepsze rezultaty.

11.5.2. Format rozszerzenia DNS (EDNS0)

Podstawowy format komunikatu DNS ma wiele ograniczeń. Poszczególne pola mają określony rozmiar, a całkowita długość komunikatu wysyłanego za pomocą protokołu UDP nie przekracza 512 bajtów (bez uwzględnienia nagłówków UDP i IP). Ograniczona jest również liczba kodów błędów (pole *RCODE* ma tylko 4 bity). W zaleceniu [RFC2671] zaproponowano więc rozszerzenie formatu komunikatów o nazwie **EDNS0** (założono, że w przyszłości mogą powstać kolejne opracowania o indeksie większym niż 0). Choć w czasie pisania książki mechanizm rozszerzeń nie był powszechnie stosowany, jego użycie jest konieczne w przypadku korzystania z zabezpieczeń DNSSEC (opisanych w rozdziale 18.). Należy się więc spodziewać stopniowego upowszechniania tego rozwiązania.

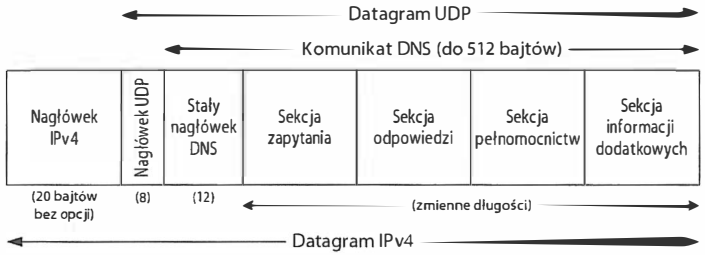
Rozszerzenie EDNS0 definiuje szczególny typ rekordu RR (nazywany pseudorekordem opcji — *OPT pseudoRR* lub *metaRR*), który jest uwzględniany w sekcji informacji dodatkowych żądania lub odpowiedzi i ma na celu poinformowanie odbiorcy o zastosowaniu rozszerzenia EDNS0. W danym komunikacie DNS może występować tylko jeden taki rekord. Format rekordu OPT pseudoRR został szczegółowo opisany w punkcie 11.5.6 (wraz z innymi rekordami). Na tym etapie rozważań istotne jest, aby pamiętać, że dołączenie rekordu OPT pseudoRR do komunikatu DNS przekazywanego za pomocą protokołu UDP eliminuje ograniczenie rozmiaru wiadomości do 512 bajtów oraz pozwala na użycie rozbudowanego zbioru kodów błędów.

W specyfikacji EDNS0 opisano również nowy typ etykiety (uzupełniający omówione wcześniej etykiety danych i etykiety kompresji). Etykiety rozszerzenia mają na dwóch pierwszych bitach bajta *Typu/Długości* ustawioną kombinację 01, która odpowiada wartościom liczbowym z przedziału od 64 do 127 (włącznie). Typ 65 był w pewnym czasie wykorzystywany do eksperymentalnego binarnego kodowania etykiet, ale obecnie nie jest używany. Z kolei wartość 127 jest zarezerwowana do przyszłego użycia, a kolejne wartości nie są na razie zdefiniowane.

11.5.3. Protokół UDP czy TCP?

Serwery DNS posługują się zarezerwowanych dla nich portem 53 zarówno w protokole UDP, jak i w protokole TCP. Najczęściej wykorzystywany format datagramu UDP/IPv4 został pokazany na rysunku 11.6.

Jeśli powracająca do resolvera odpowiedź zawiera bit TC (bit podziału) o wartości 1, oznacza to, że rzeczywista odpowiedź miała rozmiar większy niż 512 bajtów, ale jedynie 512 pierwszych bajtów zostało wysłanych z serwera. Resolver może wówczas ponowić zapytanie z użyciem protokołu TCP, którego obsługa, zgodnie z zaleceniem [RFC5966], jest obecnie obowiązkowa. Takie rozwiązanie umożliwia przesłanie więcej niż 512 bajtów, ponieważ mechanizm TCP obsługuje dzielenie większych komunikatów na kilka segmentów.



Rysunek 11.6. *Komunikaty DNS są zazwyczaj zapisywane w datagramach UDP/IPv4, a ich rozmiary nie przekraczają 512 bajtów (o ile nie został użyty protokół TCP lub rozszerzenie EDNS0). W każdej sekcji (poza sekcją zapytania) przenoszone są zbiory rekordów zasobów*

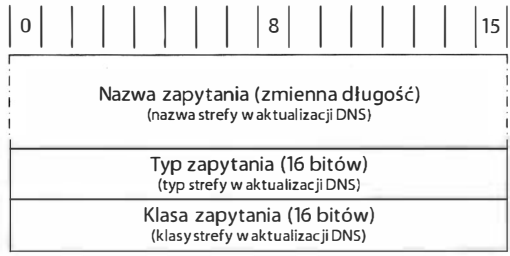
Uruchomienie zapasowego serwera nazw dla danej strefy wiąże się z wykonaniem transferu strefy z serwera podstawowego. Transfer strefy bywa inicjowany również w określonych interwałach oraz w wyniku przesłania komunikatu DNS NOTIFY (więcej informacji na ten temat znajduje się w podpunkcie 11.5.8.3). Pełny transfer strefy zawsze wymaga zastosowania protokołu TCP z uwagi na ilość przekazywanych danych. Przyrostowe transfery strefy (w których przekazywane są jedynie aktualizowane wpisy) mogą bazować na protokole UDP, ale w przypadku zbyt dużego rozmiaru komunikatu muszą mieć możliwość użycia protokołu TCP, podobnie jak w tradycyjnym zapytaniu.

Jeśli w komunikacji między resolverem i serwerem wykorzystywany jest protokół UDP, obie aplikacje muszą we własnym zakresie dbać o retransmisję utraconych datagramów. Zalecenia dotyczące implementacji odpowiedniego mechanizmu znajdują się w opracowaniu [RFC1536]. Zgodnie z zawartymi w nim sugestiami początkowo czas oczekiwania na odpowiedź powinien wynosić 4 sekundy. Wartość ta powinna być jednak zwiększana wykładniczo wraz z kolejnymi próbami komunikacji (podobnie jak w algorytmie TCP; patrz rozdział 14.). Zmiana parametru czasu retransmisji w systemach Linux i UNIX sprowadza się do edycji pliku */etc/resolv.conf* (i odpowiedniego ustawienia opcji `timeout` [czas oczekiwania] oraz `attempts` [liczba prób]).

11.5.4. Format sekcji zapytania i sekcji strefy

Zdefiniowana w komunikacie DNS sekcja zapytania przechowuje listę zgłoszonych zapytań. Format każdego z nich został przedstawiony na rysunku 11.7. Zazwyczaj w żądaniu DNS sformułowane jest tylko jedno zapytanie. Niemniej standard protokołu umożliwia zdefiniowanie większej ich liczby. Analogiczna struktura danych jest wykorzystywana w sekcji strefy w dynamicznych aktualizacjach (punkt 11.5.7). Różne są jedynie treści nazw domenowych.

Nazwa zapytania odpowiada poszukiwanej nazwie domenowej. Do jej zapisu wykorzystuje się opisane wcześniej kodowanie etykiet. Każde zapytanie ma zdefiniowany **typ zapytania** oraz **klasę zapytania**. Wartości 1, 254 i 255 oznaczają odpowiednio klasę internetową, brak klasy oraz zapytanie bezklasowe. W sieciach TCP/IP inne ustawienia nie są istotne. Pole **typ zapytania** przechowuje jedną z wartości zamieszczonych w tabeli 11.3. Odzwierciedla więc rodzaj zapytania. W praktyce najczęściej generowane jest zapytanie typu A (lub AAAA w przypadku odwzorowania DNS w sieciach IPv6), oznaczające

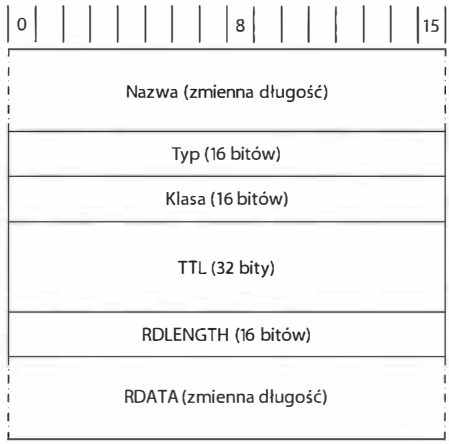


Rysunek 11.7. *Sekcja zapytania zawarta w komunikacie DNS nie uwzględnia wartości TTL, ponieważ nie podlega buforowaniu*

poszukiwanie adresu IP dla danej nazwy domenowej. Niekiedy wysyłane są również zapytania typu ANY, które powodują zwrócenie wszystkich rekordów RR (niezależnie od ich typu) z danej klasy zgodnych z nazwą zapytania.

11.5.5. Format odpowiedzi, pełnomocnictw oraz informacji dodatkowych

Trzy ostatnie sekcje komunikatu DNS — odpowiedź, pełnomocnictwa oraz informacje dodatkowe — przechowują zbiory rekordów RR. W większości przypadków nazwy domen właściciela w rekordach RR wymienionych sekcji zawierają **wieloznaczne** nazwy domenowe. Są to nazwy, w których pierwsza etykieta (etykieta danych) składa się jedynie ze znaku gwiazdki [RFC4592]. Każdy rekord zasobu ma format zgodny z pokazanym na rysunku 11.8.



Rysunek 11.8. *Format rekordu zasobu DNS. W internetowych odwołaniach do serwerów DNS pole Klasa zawsze ma wartość 1. Parametr TTL określa maksymalny czas (wyrażony w sekundach) przechowywania rekordu RR w pamięci podręcznej*

Pole *Nazwa* (określane czasami jako „nazwa właściciela", „właściciel" lub „rekord nazwy właściciela") jest nazwą domenową, z którą powiązane są dalsze informacje. Ma taki sam format jak opisywane wcześniej nazwy i etykiety. Pole *Typ* przechowuje kod jednego z typów rekordów RR (patrz punkt 11.5.6). Są to te same wartości, co opisane

wcześniej typy zapytania. W internetowych operacjach DNS pole *Klasa* ma wartość 1. W polu *TTL* zapisywana jest liczba sekund, w czasie których rekord RR może być przechowywany w buforze. Pole *RDLENGTH* (długość danych zasobu) odpowiada liczbie bajtów przechowywanych w polu danych zasobu (*RDATA*). Format samych danych zależy od typu przenoszonej informacji. Przykładowo rekordy A (typ 1) zawierają w polu *RDATA* 32-bitowy adres IPv4. Pozostałe typy rekordów RR zostały omówione w dalszej części rozdziału.

W dokumencie [RFC2181] zdefiniowano również **zbiór rekordów zasobów** (RRSet — *Resource Record Set*). Jest to zbiór zasobów, które mają wspólną nazwę, klasę i typ, ale przenoszą różne dane. Wspomniane zbiory znajdują zastosowanie np. w przypadkach, w których stacji o danej nazwie odpowiada więcej niż jeden rekord adresu (więcej niż jeden adres IP). Parametry TTL rekordów RR wchodzących w skład zbioru RRSet muszą być równe.

11.5.6. Typy rekordów zasobów

Choć system DNS jest wykorzystywany najczęściej do ustalania adresów IP odpowiadających określonej nazwie, znajduje także zastosowanie w odwzorowaniu odwrotnym oraz w innych podobnych operacjach. Współdziała z protokołami IPv4 i IPv6, a także z rozwiązaniami spoza Internetu (zgodnie ze specyfikacją DNS służą do tego inne klasy [RFC6195]). Szeroki zakres zastosowań mechanizmu DNS wynika przede wszystkim ze zdolności do przekazywania informacji w różnych typach rekordów.

Lista typów rekordów zasobów jest naprawdę imponująca (pełne zestawienie znajduje się w opracowaniu [DNSPARAMS]), a pojedyncza nazwa może być związana z wieloma takimi rekordami. W tabeli 11.3 wymieniono typy rekordów RR wykorzystywane w komunikacji ze standardowymi serwerami DNS (tj. serwerami DNS pozbawionymi rozszerzeń zabezpieczających DNSSEC).

Rekordy zasobów mają szeroki zakres zastosowań. Można je jednak podzielić na trzy ogólne kategorie — typy danych, typy zapytań oraz metatypy. Typy danych są używane do przenoszenia informacji zgromadzonych w systemie DNS (takich jak adresy IP czy nazwy autorytatywnych serwerów DNS). Typy zapytań są wyznaczane przez te same wartości, które opisują typy danych, ale z dodatkiem kilku nowych pozycji (np. AXFR, IXFR oraz *). Znajdują zastosowanie w opisanej wcześniej sekcji zapytania. Metatypy definiują tymczasowe struktury danych, obowiązujące jedynie w czasie transmisji pojedynczego komunikatu DNS. W dalszej części rozdziału opisany został tylko jeden taki typ — OPT RR (wszystkie pozostałe zostały omówione w rozdziale 18.). Do najczęściej wykorzystywanych rekordów RR danych z pewnością należy zaliczyć rekordy A, NS, SOA, MX, CNAME, PTR, TXT, AAAA, SRV i NAPTR. Rekordy NS zapewniają powiązanie przestrzeni nazw DNS z serwerami, które odpowiadają za odwzorowanie nazw — przechowują nazwy autorytatywnych serwerów DNS danej strefy. Rekordy A i AAAA udostępniają adresy IPv4 i IPv6 powiązane z daną nazwą. Rekord CNAME umożliwia pozyskanie informacji o alternatywnej nazwie domenowej. Natomiast rekordy SRV i NAPTR ułatwiają wyznaczenie położenia serwerów oferujących określone usługi i umożliwiają wykorzystanie alternatywnego systemu nazewnictwa (spoza przestrzeni DNS) w odwołaniach do tych usług. Każdy z wymienionych typów rekordów został szczegółowo opisany w kolejnych podpunktach rozdziału.

Tabela 11.3. *Rekordy zasobów i typy zapytań stosowane w standardowych komunikatach protokołu DNS. W rozwiązaniach DNSSEC używane są dodatkowe rodzaje rekordów (niewymienione w tabeli)*

Wartość	Typ RR	Specyfikacja	Opis i przeznaczenie
1	A	[RFC1035]	Rekord adresu w sieciach IPv4 (32-bitowy adres IP)
2	NS	[RFC1035]	Serwer nazw; udostępnia nazwę autorytatywnego serwera nazw danej strefy
5	CNAME	[RFC1035]	Nazwa kanoniczna; odwzorowanie jednej nazwy na inną (funkcja aliasów nazw)
6	SOA	[RFC1035]	Rekord początkowy strefy; udostępnia informacje o samej strefie (adresy serwerów nazw, adres e-mail osoby do kontaktu, numer seryjny konfiguracji, parametry czasowe transferu strefy)
12	PTR	[RFC1035]	Wskaźnik; zapewnia odwzorowanie adresu IP na nazwę; wykorzystywany w domenach `in-addr.arpa` i `ip6.arpa` do obsługi zapytań odwrotnych z sieci IPv4 i IPv6
15	MX	[RFC1035]	Przekaźnik poczty; przechowuje nazwę jednostki obsługującej pocztę elektroniczną w danej domenie
16	TXT	[RFC1035] [RFC1464]	Tekst; udostępnia różne informacje (np. w mechanizmie antyspamowym SPF służy do identyfikacji zaufanych serwerów pocztowych)
28	AAAA	[RFC3596]	Rekord adresu w protokole IPv6 (128-bitowy adres IP)
33	SRV	[RFC2782]	Wybór serwera; określenie punktu końcowego usługi
35	NAPTR	[RFC3403]	Wskaźnik na właściciela nazwy; obsługa alternatywnych przestrzeni nazw
41	OPT	[RFC2671]	PseudoRR; obsługa dłuższych datagramów, etykiet i kodów wynikowych w rozszerzeniu EDNS0
251	IXFR	[RFC1995]	Przyrostowy transfer strefy
252	AXFR	[RFC1035] [RFC5936]	Pełny transfer strefy w protokole TCP
255	(ANY)	[RFC1035]	Pobranie wszystkich rekordów (dowolnego rekordu)

11.5.6.1. Rekordy adresu (A, AAAA) i serwera nazw (NS)

Bez wątpienia najczęściej używanymi rekordami systemu DNS są rekordy adresu (A, AAAA) oraz serwera nazw (NS). Rekord A przechowuje 32-bitowy adres IPv4, a rekord AAAA adres IPv6. W rekordach NS są z kolei zapisywane nazwy autorytatywnych serwerów DNS określonej strefy. Ponieważ sama nazwa serwera DNS nie jest przydatna do wykonania zapytania, w sekcji informacji dodatkowych wraz z nimi są zazwyczaj przekazywane adresy IP tych serwerów (w tzw. rekordach doklejonych). Przekazanie rekordów doklejonych eliminuje konieczność wygenerowania kolejnego żądania w sytuacjach, w których autorytatywne serwery DNS używają nazwy domenowej obsługiwanej w tych serwerach (gdy np. trzeba odwzorować nazwę `ns1.przyklad.pl` w serwerze DNS domeny `przyklad.pl` dostępnym pod adresem `ns1.przyklad.pl`). Strukturę rekordów

A, AAAA i NS doskonale prezentuje program `dig` instalowany w większości systemów Linux i UNIX. Oto przykład pobrania rekordów dowolnego typu powiązanych z domeną `rfc-editor.org`:

```
Linux% dig +nostats -t ANY rfc-editor.org
; <<>> DiG 9.6.0-P1 <<>> +nostats -t ANY rfc-editor.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 53052
;; flags: qr rd ra; QUERY: 1, ANSWER: 12, AUTHORITY: 0, ADDITIONAL: 2

;; QUESTION SECTION:
;rfc-editor.org. IN ANY

;; ANSWER SECTION:
...
rfc-editor.org.    1654 IN   AAAA   2001:1890:1112:1::2f
rfc-editor.org.    1654 IN   A      64.170.98.47
rfc-editor.org.    1654 IN   NS     ns0.ietf.org.
rfc-editor.org.    1654 IN   NS     ns1.hkg1.afilias-nst.info.
...
;; ADDITIONAL SECTION:
ns0.ietf.org.       756 IN   A      64.170.98.2
ns0.ietf.org.       756 IN   AAAA   2001:1890:1112:1::14
```

Pierwsze dwa wiersze zestawienia wynikowego wskazują wersję programu `dig`, dostarczone do niego opcje oraz opcje domyślne (opcja `+cmd` oznacza, że wynik powinien zostać wyświetlony). Kolejne sekcje prezentują dane zawarte w komunikacie odpowiedzi DNS — kod operacji `QUERY` (zapytanie), status `NOERROR` (informujący, że żądanie zostało przetworzone bez błędów) oraz identyfikator transakcji o wartości `53052`. Wartość `QUERY` występuje w polach *OpCode* zarówno zapytania, jak i odpowiedzi. Dopiero kolejny wiersz (`flags`) informuje, że dany komunikat jest odpowiedzią na zapytanie (znacznik `qr`), a nie samym zapytaniem. Ponadto dwa kolejne znaczniki `rd` i `ra` wskazują, że żądane działanie rekurencyjne (znacznik `rd`) jest obsługiwane przez odpowiadający serwer (znacznik `ra`). Komunikat składa się z sekcji zapytania obejmującej jedno zapytanie oraz sekcji odpowiedzi z dwunastoma rekordami zasobów (z których tylko cztery zostały przedstawione). Brak rekordów w sekcji pełnomocnictw oznacza, że odpowiedź została pobrana z serwera buforującego (rekordy RR nie są autorytatywne). Istnieje więc ryzyko, że pobierając dane z innego serwera, można otrzymać inne wartości. Informacje zapisane w sekcji dodatkowej obejmują adresy IPv4 i IPv6 jednego z serwerów autorytatywnych, z którym można ewentualnie nawiązać komunikację. W sekcji zapytania znajduje się kopia pierwotnego żądania — typ `ANY`, domena `rfc-editor.org`.

W zbiorze rekordów dostarczonych w sekcji odpowiedzi znajdują się: jeden rekord typu A, jeden rekord typu AAAA oraz dwa rekordy NS. Z ich treści wynika, że nazwa domenowa `rfc-editor.org` jest powiązana z adresem IPv4 `64.170.98.47` oraz adresem IPv6 `2001:1890:1112:1::2f`. Rekordy NS świadczą dodatkowo o tym, że jest to również poddomena. Kolejny test potwierdza tę tezę. W poddomenie pracuje przynajmniej jeden komputer:

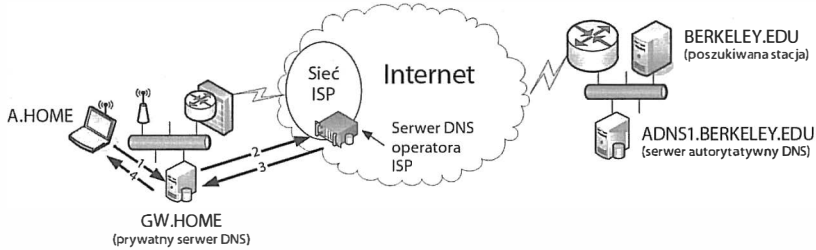
```
Linux% host ftp.rfc-editor.org
ftp.rfc-editor.org has address 64.170.98.47
```

Powyższe przykłady dostarczają kilku ciekawych informacji na temat rekordów A, AAAA i NS. Oto one. Z pojedynczą nazwą domenową może być powiązanych kilka rekordów różnych typów. Taka konfiguracja jest często stosowana w odniesieniu do serwe-

rów IPv6, które są „dobrze znanymi" serwerami danej organizacji. Z analizy zestawienia wynika również, że każdemu rekordowi odpowiada parametr TTL. Wartości parametru mogą się różnić, o ile nie należą do jednego zbioru RRSet. W zaprezentowanym przykładzie czas ważności rekordów z sekcji odpowiedzi wynosił 1654 s (ok. pół godziny). Natomiast rekordy w sekcji informacji dodatkowych są ważne jedynie przez 756 s (ok. 12 minut). Trzeba pamiętać, że wartość TTL zbuforowanego rekordu nie może być większa niż wynika z parametru TTL rekordu pozyskanego z autorytatywnego źródła. Wartości TTL buforowanych rekordów są stopniowo zmniejszane, aż do czasu ponownego pobrania danych z serwera autorytatywnego. Oznacza to, że w odpowiedziach na kolejne zapytania kierowane do serwera buforującego czas ważności rekordów jest coraz mniejszy.

11.5.6.2. Przykład

Znając format komunikatu DNS, opcje protokołu transportowego oraz typy rekordów RR stosowane z zapytaniach i odpowiedziach, możemy przystąpić do testowania mechanizmu. W pierwszym z prezentowanych przykładów zademonstrujemy przebieg komunikacji między resolverem stacji klienckiej, lokalnym serwerem nazw oraz zdalnym serwerem nazw, zarządzanym przez dostawcę usług internetowych. Przeprowadzone doświadczenie potwierdza zasadność buforowania danych DNS. Topologia sieci testowej została pokazana na rysunku 11.9.



Rysunek 11.9. *Przykład żądania i odpowiedzi DNS. Lokalny serwer DNS (GW.HOME) przetwarza zapytanie klienta (A.HOME) w sposób rekurencyjny — odwołuje się do serwera DNS operatora ISP, gdy nie dysponuje danymi w pamięci podręcznej*

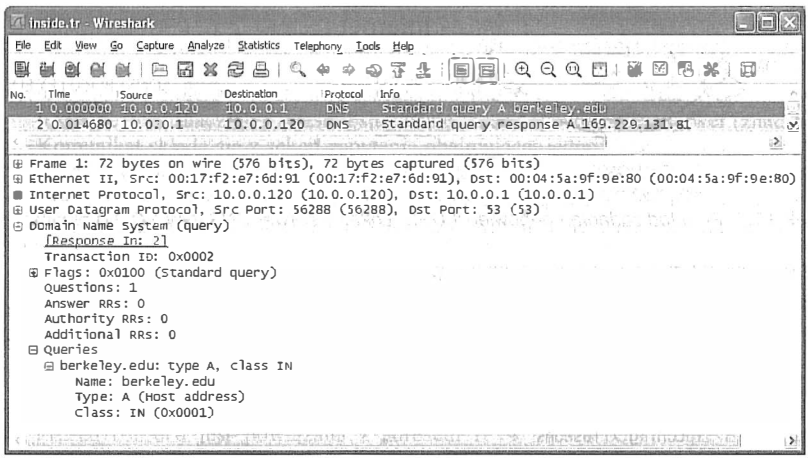
W systemie klienckim (Windows; `A.HOME`) pracę należy rozpocząć od wykonania polecenia, które usunie dane DNS zbuforowane wcześniej przez oprogramowanie resolvera. Następnie wystarczy wygenerować zapytanie o adres (rekord typu A) powiązany z nazwą domenową `berkeley.edu`:

```
C:\> ipconfig /flushdns
Konfiguracja IP systemu Windows

Pomyślnie opróżniono pamięć podręczną programu rozpoznawania nazw DNS.

C:\> nslookup
Serwer domyślny: gw
Adres: 10.0.0.1

> set type=a
> berkeley.edu.
```

```
Serwer: gw
Adres: 10.0.0.1

Nieautorytatywna odpowiedź:
Name: berkeley.edu
Address: 169.229.131.81
```

Pierwsze polecenie obowiązuje tylko w systemie Windows. Odpowiada za usunięcie danych zarejestrowanych w pamięci podręcznej przez oprogramowanie resolvera. Program `nslookup` (dostępny zarówno w systemach Windows, jak i Linux/UNIX) umożliwia wysyłanie zapytań do serwera DNS ze wskazaniem rodzaju spodziewanych informacji. Po uruchomieniu narzędzia na ekranie wyświetlane są dane serwera nazw używanego w odwołaniach do systemu DNS (w prezentowanym przykładzie serwerem jest jednostka o nazwie `gw` i adresie `10.0.0.1`). Instrukcja `set` umożliwia wskazanie rodzaju rekordów (w tym przypadku `A`), które powinny zostać zwrócone w wyniku sprawdzenia podanej nazwy (`berkeley.edu.`). Po wykonaniu zapytania program `nslookup` po raz kolejny informuje o tym, który serwer został wykorzystany do uzyskania danych. W analizowanym przykładzie program dodatkowo wyświetlił ostrzeżenie, że odpowiedź nie jest autorytatywna (czyli została pobrana z serwera buforującego). Prezentowany jest również adres poszukiwanej jednostki — `169.229.131.81`.

Aby sprawdzić, jakie zadania są realizowane przez protokół DNS na poziomie pakietów, wystarczy użyć programu Wireshark i przeanalizować pierwszy z wysyłanych datagramów (pokazany na rysunku 11.10).

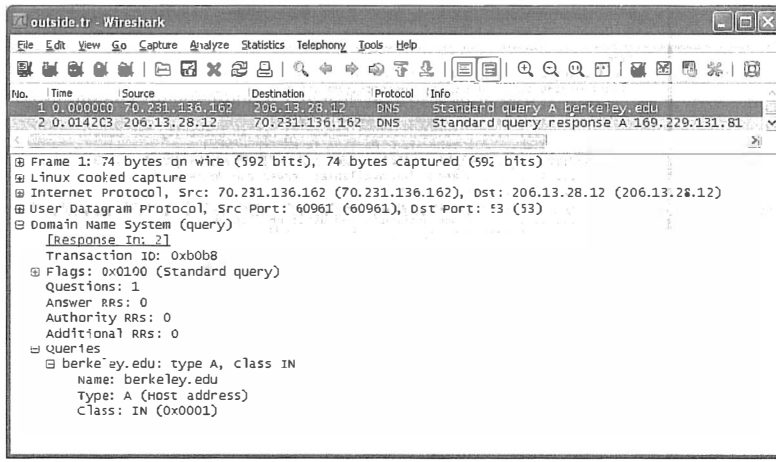


Rysunek 11.10. *Datagram UDP/IPv4 przenoszący standardowe zapytanie DNS — zapytanie o adres IPv4 jednostki berkeley.edu.*

Podczas realizacji zadania przechwycone zostały dwa komunikaty — standardowe zapytanie i standardowa odpowiedź. W pierwszym odwołaniu (w zapytaniu) źródłowy adres IPv4 ma wartość `10.0.0.120` (adres przydzielony przez serwer DHCP jednostce klienckiej; patrz rozdział 6.), a docelowy `10.0.0.1` (adres serwera DNS). Zapytanie jest przenoszone w datagramie UDP/IPv4 z portu źródłowego 56 288 (losowy numer) do portu 53

(zarejestrowanego portu usługi DNS). Analizując cały proces enkapsulacji, można stwierdzić, że ramka ethernetowa składa się z 72 bajtów, a w jej skład wchodzą: nagłówek ethernetowy (14 bajtów), nagłówek IPv4 (20 bajtów), nagłówek UDP (8 bajtów), stały nagłówek DNS (12 bajtów), definicja typu zapytania (2 bajty), definicja klasy zapytania (2 bajty) oraz etykiety danych — `berkeley` (9 bajtów), `edu` (4 bajty) i zerowy bajt zakończenia.

Z treści nagłówka wynika, że identyfikator transakcji ma wartość `0x0002` i zajmuje dwa pierwsze bajty nagłówka (zaraz na początku pola danych UDP). Ustawiony jest tylko jeden znacznik (żądanie operacji rekurencyjnej; ustawienie domyślne), z czego wynika, że weryfikowany komunikat jest zapytaniem. W treści komunikatu znajduje się standardowa sekcja zapytania z jednym konkretnym zapytaniem. Pozostałe sekcje są puste. Zapytanie dotyczy nazwy `berkeley.edu`, a jego celem jest pozyskanie informacji typu A (rekordów adresu) w klasie IN (internetowej). Pracujący pod adresem `10.0.0.1` serwer nazw nie może sam odpowiedzieć na dostarczone do niego zapytanie, więc przekazuje je do kolejnego serwera nazw (zdefiniowanego w jego lokalnej konfiguracji). W analizowanym przykładzie ostateczny serwer DNS posługiwał się adresem `206.13.28.12` (co wynika z rysunku 11.11).



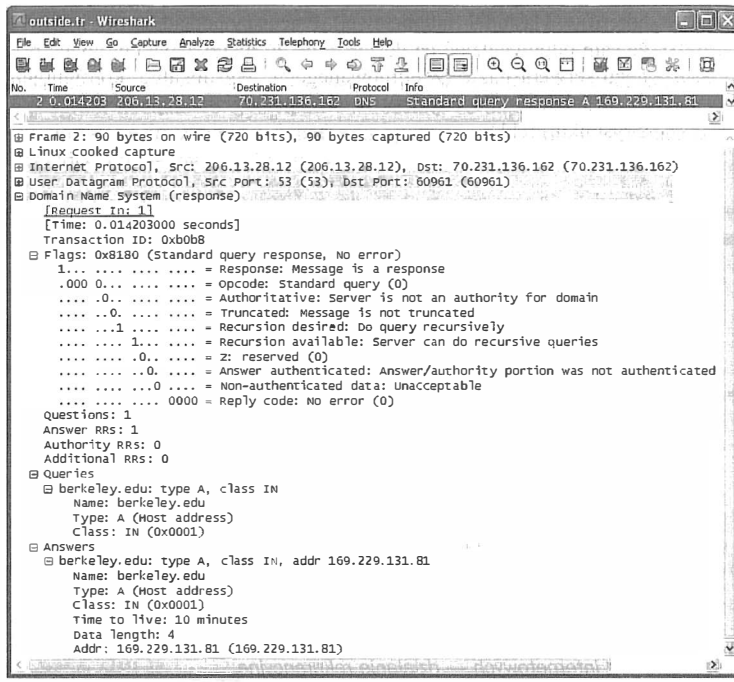
Rysunek 11.11. *Żądanie DNS wygenerowane w systemie GW.HOME i przekazane do serwera DNS dostawcy usług internetowych — działanie rekurencyjne*

Zrzut widoczny na rysunku 11.11 przypomina zapytanie przesłane przez stację kliencką. Inny jest jednak źródłowy adres IPv4 — `70.231.136.162` (adres IPv4 zewnętrznego interfejsu jednostki `GW.HOME`). Docelowy adres IPv4 (`206.13.28.12`) należy do serwera DNS udostępnianego przez dostawcę usług internetowych. Port źródłowy jest natomiast losowo wybranym portem lokalnego serwera DNS (60 961). Identyfikator transakcji otrzymał nową wartość `0xb0b8`. Ponadto program Wireshark informuje, że odpowiedź na zapytanie znajduje się w pakiecie o numerze 2.

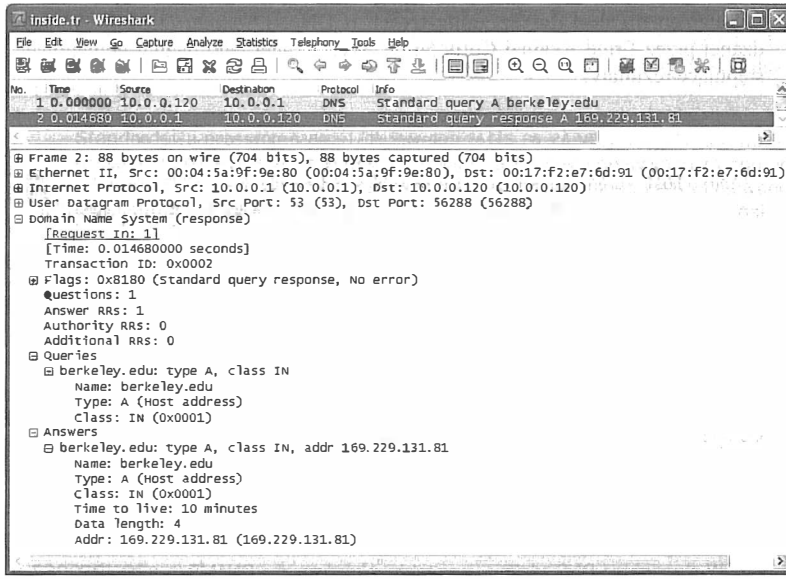
Pakiet 2. został przedstawiony na rysunku 11.12. Przenosi on pierwszą zarejestrowaną odpowiedź serwera DNS. Na początek warto zauważyć, że źródłowy port UDP ma wartość 53, natomiast port docelowy odpowiada losowo wybranej wartości 60 961. Identyfikator

transakcji odpowiada zdefiniowanemu w zapytaniu (0xb0b8), ale pole znaczników (*Flags*) ma tym razem wartość 0x8180 (z ustawionymi bitami odpowiedzi, żądania rekurencji i dostępności rekurencji). Sekcja zapytania przechowuje kopię zapytania, które stało się podstawą wygenerowania analizowanej odpowiedzi i dokładnie odpowiada zapytaniu pierwotnemu (dostarczonemu przez jednostkę kliencką; zachowane są np. pierwotne wielkości liter). W sekcji odpowiedzi znajduje się jeden rekord RR typu A (adres) o dziesięciominutowym czasie ważności (TTL), rozmiarze czterech bajów (rozmiar adresu IPv4) i wartości `169.229.131.81`. Jest to adres IPv4 poszukiwanej jednostki (`berkeley.edu`). Brak bitu *Authority* oraz pusta sekcja pełnomocnictw oznaczają, że odpowiedź pochodzi z pamięci podręcznej serwera — nie jest autorytatywna. Lokalny serwer nazw buforuje uzyskane dane (ale jedynie na okres 10 minut, zgodnie z wartością TTL otrzymanego rekordu) i przesyła odpowiedź do klienta (patrz rysunek 11.13).

Rysunek 11.12. *Standardowa odpowiedź DNS wysłana z serwera dostawcy usług internetowych do jednostki GW.HOME*



Odpowiedź przedstawiona na rysunku 11.13 (pakiet 2.) jest bardzo zbliżona do komunikatu otrzymanego z jednostki `206.13.28.12`. Różnią się jedynie adresy IP (zamiast `10.0.0.1` jest adres klienta `10.0.0.120`) oraz identyfikator transakcji (który musi odpowiadać identyfikatorowi pierwotnego żądania). Z perspektywy stacji klienckiej realizacja żądania zajęła serwerowi 14,7 ms. Wiadomo jednak, że większość tego czasu (14,2 ms) zajęła transakcja między lokalnym serwerem nazw (`GW.HOME`) i serwerem nazw dostawcy usług internetowych (`206.13.28.12`).



Rysunek 11.13. *Odpowiedź utworzona w komputerze GW.HOME i wysłana do stacji klienckiej. Komunikat ten kończy rekurencyjną transakcję DNS*

11.5.6.3. Rekordy nazw kanonicznych (CNAME)

Rekordy CNAME są nazywane rekordami **nazw kanonicznych** i znajdują zastosowanie w tworzeniu aliasów dla pojedynczych nazw domenowych. Przykładowo nazwa `www.berkeley.edu` może zostać powiązana za pomocą rekordów CNAME z nazwami różnych komputerów (np. `www.w3.berkeley.edu`). Dzięki temu upublicznienie serwera WWW po przeniesieniu usługi do innego systemu wiąże się z relatywnie nieskomplikowaną operacją zmiany wpisu w bazie danych DNS. Poza tym rekordy CNAME są używane do definiowania aliasów odpowiadających poszczególnym usługom dostępnym w danym systemie. Oznacza to, że nazwy, takie jak `www.onet.pl`, `ftp.helion.pl` czy `poczta.interia.pl`, są w istocie jedynie wpisami CNAME odnoszącymi się do innych rekordów RR.

Nazwa kanoniczna powiązana z określoną nazwą domenową jest przechowywana w rekordzie CNAME w polu *RDATA*. Do zapisu tego rodzaju nazw stosuje się te same mechanizmu, które obowiązują w definiowaniu innych nazw (wykorzystywane są etykiety danych i etykiety kompresji). Jeśli do danej nazwy zostanie przypisany rekord CNAME, nie można definiować innych jej atrybutów [RFC1912] (poza przypadkiem użycia mechanizmu DNSSEC, opisanego w rozdziale 18.). Nazw domenowych rekordów CNAME nie można również wykorzystywać w niektórych miejscach, w których standardowe nazwy domenowe są używane bez problemu (np. jako nazwy docelowej w rekordzie NS). Nazwa kanoniczna może jednak odnosić się do kolejnej nazwy kanonicznej (tworząc **łańcuch CNAME**), choć takie rozwiązanie nie jest zalecane, ponieważ prowadzi niekiedy do zwiększenia liczby zapytań. Niemniej jednak część serwisów z niego korzysta. Przykładem jest chociażby portal `www.whitehouse.gov`, który w czasie pisania książki

był elementem **sieci dostarczania treści**[2] (CDN — *Content Delivery Network*) zarządzanej przez firmę Akamai Corporation. Weryfikacja jego nazwy domenowej daje następujące rezultaty:

```
Linux% host -t any www.whitehouse.gov
www.whitehouse.gov is an alias for www.whitehouse.gov.edgesuite.net.

Linux% host -t any www.whitehouse.gov.edgesuite.net
www.whitehouse.gov.edgesuite.net is an alias for  www.eop-edge-lb.akadns.net.

Linux% host -t any www.eop-edge-lb.akadns.net
www.eop-edge-lb.akadns.net is an alias for a1128.dsch.akamai.net.

Linux% host -t any a1128.dsch.akamai.net
a1128.dsch.akamai.net has address 217.89.106.66
a1128.dsch.akamai.net has address 217.89.106.75
```

Łańcuchy CNAME są więc wykorzystywane w systemie DNS. Jednak z powodu negatywnego wpływ na wydajność komunikacji resolvery często ograniczają liczbę tego typu odwołań (do np. pięciu). Dlatego długie łańcuchy powiązań mogą spowodować błędne działanie programów klienckich. Trudno również znaleźć dla nich uzasadnienie, jeśli konfiguracja serwerów nie jest szczególnie udziwniona.

W standardzie DNS zdefiniowano także rekord zasobu DNAME (typ 39) [RFC2672][IDDN], którego działanie jest zbliżone do rekordu CNAME, ale odnosi się do całej strefy. Umożliwia np. zdefiniowanie mapowania między nazwami `NAZWA.przyklad.pl` i `NAZWA.nowyprzyklad.pl`. Rekordy DNAME nie są jednak stosowane w odniesieniu do rekordów najwyższego poziomu (w tym przypadku do rekordu `pl`).

11.5.6.4. Odwrotne zapytania DNS — rekordy wskaźnika (PTR)

Choć podstawowym zadaniem systemu DNS jest odwzorowywanie nazw domenowych na adresy IP, okazuje się on również przydatny w wielu sytuacjach do realizacji odwrotnego zadania. Przykładowo serwer odbierający żądanie ustanowienia połączenia TCP/IP może ustalić adres stacji inicjującej połączenie, analizując odebrany datagram IP, ale informacja o nazwie (lub nazwach) przypisanej danemu komputerowi nie już przenoszona w pakietach połączenia. Aby ją ustalić, serwer musi się odwołać do serwera DNS.

Odpowiedzi na odwrotne zapytania DNS są przenoszone w rekordach PTR (służą zazwyczaj do zamiany adresu IP na nazwę jednostki). Wykonanie operacji wiąże się jednak z użyciem specjalnych domen `in-addr.arpa` (oraz `ip6.arpa` w sieciach IPv6). Przeanalizujmy adres IPv4 o wartości `128.32.112.208`. Zgodnie z zasadami adresowania klasowego (więcej informacji na ten temat znajduje się w rozdziale 2.) jest to adres z klasy B o identyfikatorze sieci `128.32`. Aby ustalić nazwę jednostki dysponującej podanym adresem, konieczne jest odwrócenie kolejności bajtów adresu oraz dodanie specjalnej domeny. W rozważanym przypadku oznacza to wygenerowanie zapytania o rekord PTR odpowiadający nazwie

```
208.112.32.128.in-addr.arpa.
```

[2] Sieć dostarczania treści składa się wielu zsynchronizowanych buforów treści, rozmieszczonych w wybranych miejscach sieci. Rozwiązania CDN pozwalają na zmniejszenie opóźnień w dostępie do informacji. Za możliwość korzystania z tego systemu płacą dostawcy treści.

W praktyce jest to jednak zapytanie o stację 208 w domenie 112.32.128.in-addr.arpa. Więcej przykładów odwrotnych zapytań DNS zostało przedstawionych w dalszej części rozdziału.

Standardowa przestrzeń nazw DNS (rekordy NS, A i AAAA) nie jest automatycznie wiązana z przestrzenią nazw odwzorowania odwrotnego opisywaną za pomocą rekordów PTR. Jest więc możliwe (a nawet często spotykane), że zdefiniowanemu odwzorowaniu prostemu nie towarzyszy odwzorowanie odwrotne (lub odnosi się do innych wartości). Niektóre usługi sprawdzają, czy powiązanie między adresem i nazwą jest identyczne w obu kierunkach i w zależności od wyniku testu wykonują swoje zadanie lub nie.

Jak wiadomo, adresy IPv4 są zazwyczaj zapisywane w formacie dziesiętnym z kropkami, a adresy IPv6 w formacie szesnastkowym (np. 169.229.131.81 i 2001:502:a83e::2:30). Adresy te można traktować jak nazwy, ale z założeniem, że hierarchia ważności poszczególnych bajtów odpowiada zapisowi od lewej do prawej strony. Przykładowo adres 169.229.131.81 składa się z bajtów, które ułożone w kolejności od najważniejszego w hierarchii do najmniej istotnego tworzą szereg 169, 229, 131, 81. Odwrócenie kolejności zapisu adresu IPv4 i potraktowanie go jak nazwy domenowej pozwala na wykorzystanie systemu DNS do odwzorowania adresu IP na nazwę (lub nazwy). Zatem nazwa 81.131.229.169 jest odwróconym adresem IPv4 169.229.131.81. Adresy IPv6 podlegają tym samym regułom, ale wszystkie zredukowane zera muszą zostać odtworzone, a każda cyfra szesnastkowa jest uznawana za osobny znak. I tak w wyniku odwrócenia adresu 201:503:a83e::2:30 powstaje ciąg 0.3.0.0.2.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.e.3.8.a.3.0.5.0.1.0.0.2. Na szczęście, użytkownicy rzadko muszą osobiście wpisywać takie adresy.

Zgodnie z wcześniejszym stwierdzeniem, obsługa odwrotnych zapytań DNS wiąże się z użyciem w rekordach PTR specjalnej domeny .in-addr.arpa (w odniesieniu do adresów IPv4) oraz .ip6.arpa (w odniesieniu do adresów IPv6). Oto przykładowe odwołania do serwera DNS:

```
C:\> nslookup
Serwer domyślny: gw
Adres: 10.0.0.1

> server c.in-addr-servers.arpa
Serwer domyślny: c.in-addr-servers.arpa
Adres: 196.216.169.10

> set type=ptr
> 81.131.229.169.in-addr.arpa.
Serwer: c.in-addr-servers.arpa
Adres: 196.216.169.10
169.in-addr.arpa nameserver = w.arin.net
169.in-addr.arpa nameserver = t.arin.net
169.in-addr.arpa nameserver = dill.arin.net
169.in-addr.arpa nameserver = x.arin.net
169.in-addr.arpa nameserver = z.arin.net
169.in-addr.arpa nameserver = y.arin.net
169.in-addr.arpa nameserver = u.arin.net
169.in-addr.arpa nameserver = v.arin.net
```

W powyższym przykładzie widać sposób konfiguracji domen .in-addr.arpa. Zgodnie z zaleceniem [RFC5855] serwery DNS odpowiedzialne za obsługę mechanizmu odwrot-

nego odwzorowania mają przypisane nazwy domenowe `in-addr-servers.arpa` oraz `ip6-servers.arpa` (w sieciach IPv4 i IPv6). W 2011 roku działało pięć takich serwerów w każdej z wersji protokołu IP: *x*`.in-addr-servers.arpa` oraz *x*`.ip6-servers.arpa` (*x* należy zastąpić literą z przedziału od `a` do `f` włącznie).

Mimo że każdy z dziesięciu wspomnianych serwerów zawiera autorytatywne informacje związane z odwrotnym odwzorowaniem, żaden z nich nie dysponuje danymi, które były poszukiwane. Odpowiedź dostarczona przez pierwszy z odpytanych serwerów zawierała jedynie wskazówkę, że należy się skontaktować z jednym z ośmiu serwerów utrzymywanych przez organizację ARIN mającą pełnomocnictwa do operowania adresami IPv4 rozpoczynającymi się do liczby 169. Oznacza to, że można wybrać jeden z wymienionych serwerów do uzyskania rekordu PTR odpowiadającego nazwie `81.131.229.169.`
`↪in-addr.arpa.`:

```
> server w.arin.net
Serwer domyślny: w.arin.net
Adres: 72.52.71.2

Serwer domyślny: w.arin.net
Adres: 2001:470:1a::2

> 81.131.229.169.in-addr.arpa.
Serwer: w.arin.net
Adres: 72.52.71.2

229.169.in-addr.arpa nameserver = adns1.berkeley.edu.
229.169.in-addr.arpa nameserver = phloem.uoregon.edu.
229.169.in-addr.arpa nameserver = aodns1.berkeley.edu.
229.169.in-addr.arpa nameserver = adns2.berkeley.edu.
```

Z uzyskanych informacji wynika, że prefiks sieci `169.229/16` należy do instytucji edukacyjnej o nazwie Berkeley. W sieci uczelnianej pracują trzy serwery nazw obsługujące przestrzeń `in-addr.arpa`. Kopia rejestru jest również utrzymywana na serwerze uniwersytetu w Oregonie. Ostateczna odpowiedź zostanie więc zwrócona przez jeden z tych serwerów (wynik wykonania tego samego zadania za pomocą polecenia `nslookup` w systemie Linux jest sformatowany nieco inaczej):

```
Linux% nslookup
> set type=ptr
> server adns1.berkeley.edu
Default Server: adns1.berkeley.edu
Address: 128.32.136.3#53
Default Server: adns1.berkeley.edu
Address: 2607:f140:ffff:fffe::3#53
> 81.131.229.169.in-addr.arpa.
Server: adns1.berkeley.edu
Address: 128.32.136.3#53

81.131.229.169.in-addr.arpa name = webfarm.Berkeley.EDU
```

W ten sposób uzyskujmy poszukiwane informacje — jednostka o adresie IPv4 `169.229.131.81` posługuje się nazwą `webfarm.Berkeley.EDU`. Serwer DNS pracuje na porcie 53, o czym świadczy ciąg #53 dołączony do adresu IP. Z listingu wynika również, że komunikacja DNS w protokole UDP/IPv4 (w przeciwieństwie do protokołu

UDP/IPv6) pozwala także na pozyskiwanie danych o odwzorowaniach IPv6 zdefiniowanych za pomocą rekordów AAAA — w treści odpowiedzi znajduje się adres IPv6 serwera `2607:f140:ffff:fffe::3`.

Gdyby nie istniała oddzielna gałąź drzewa DNS przeznaczona do uzyskiwania nazwy na podstawie adresu, praktycznie nie byłoby żadnej możliwości wykonania odwrotnego odwzorowania poza kolejnymi próbami odwołań do wszystkich serwerów DNS najwyższego poziomu. Przy obecnej wielkości Internetu jest to — oczywiście — zadanie niemożliwe do zrealizowania. Użycie domen `in-addr.arpa` wydaje się więc całkiem efektywne, mimo konieczności odwracania kolejności bajtów i dopisywania specjalnych domen.

Na szczęście, użytkownicy bardzo rzadko korzystają z opisanego mechanizm. Nawet twórcy aplikacji nie muszą przekształcać adresów, żeby zaimplementować mechanizm odwrotnego odwzorowania, ponieważ stosowny kod jest już uwzględniony w funkcjach bibliotek programistycznych (np. w funkcji języka C `getnameinfo()`).

Obsługa zapytań PTR jest jednak pewnym obciążeniem dla globalnych serwerów DNS. Przeanalizujmy działanie sieci domowej, której przypisano jeden z prywatnych prefiksów, takich jak `10.0.0.0/8` (IPv4) lub `fc00:/7` (IPv6). Gdy jeden z systemów odbierze żądanie ustanowienia połączenia z drugą jednostką (pracującą w tej samej sieci prywatnej), może zażądać odwzorowania źródłowego adresu pakietu na nazwę przy użyciu do tego celu zapytania PTR. Jeśli odpowiedzi nie będzie mógł udzielić lokalny serwer DNS, zapytanie najprawdopodobniej zostanie przekazane do serwerów internetowych. Z tego powodu (a także z kilku innych przyczyn) w dokumencie [RFC6303] zdefiniowano zastrzeżenie odnoszące się do lokalnych serwerów nazw (szczególnie tych, które działają w sieciach o adresach prywatnych i mają połączenie z Internetem), zgodnie z którym muszą one zapewniać odwzorowania PTR dla wszystkich adresów z prywatnej przestrzeni adresowej opisanej w zaleceniach [RFC1918] (w sieciach IPv4) oraz [RFC4193] (w sieciach IPv6) (tj. odwzorowania w domenach `in-addr.arpa` oraz `d.f.ip6.arpa`).

11.5.6.5. Bezklasowe delegacje in-addr.arpa

Gdy dana organizacja przyłączy swoją sieć do Internetu i uzyska pełnomocnictwo do zdefiniowania odpowiedniej przestrzeni nazw DNS, dostaje również pozwolenie na opisanie fragmentu przestrzeni `in-addr.arpa` związanej z przydzielonymi jej adresami IPv4. W przypadku omawianego wcześniej uniwersytetu w Berkeley pełnomocnictwo dotyczy prefiksu sieciowego `169.229/16`, który w starszej terminologii zostałby określony jako sieć klasy B o numerze 169.229. Zatem uniwersytet w Berekely powinien zdefiniować rekordy PTR w tym fragmencie drzewa DNS, w którym wykorzystywane są nazwy kończące się ciągiem `229.169.in-addr.arpa`. Takie rozwiązanie sprawdza się doskonale, jeśli przydzielony organizacji prefiks adresu pokrywa się z klasami A, B lub C adresów (czyli gdy liczba bitów prefiksu jest całkowicie podzielna przez 8). W praktyce jednak wiele firm posługuje się prefiksami o liczbie bitów większej niż 24 lub większej niż 16, ale mniejszej niż 24. Wówczas zakresu adresów nie można zapisać jako zwykłego odwrócenia bajtów adresu IP. Konieczne jest uwzględnienie rzeczywistej długości prefiksu.

Typowy sposób postępowania w opisanym przypadku został przedstawiony w dokumencie [RFC2317]. Polega on na dołączeniu informacji o długości prefiksu do ciągu odwróconych oktetów i wykorzystaniu go jako pierwszej etykiety w nazwie domenowej. Jeśli np. sieć

dysponuje prefiksem 12.17.136.128/25 (wyznaczającym 128 adresów), w konfiguracji DNS powinny zostać uwzględnione dwa typy rekordów. Na początek każdej nazwie *x*.136.17.12.in-addr.arpa (w której *x* ma wartość z przedziału od 128 do 255) przypisywany jest rekord CNAME (zazwyczaj zajmują się tym dostawcy usług internetowych), zgodnie z poniższym wzorcem:

```
128.136.17.12.in-addr.arpa. canonical name =
                            128.128/25.136.17.12.in-addr.arpa.
129.136.17.12.in-addr.arpa. canonical name =
                            129.128/25.136.17.12.in-addr.arpa.
...
255.136.17.12.in-addr.arpa. canonical name =
                            255.128/25.136.17.12.in-addr.arpa.
```

W powyższym zestawieniu widać sposób zapisywania prefiksu sieci z uwzględnieniem znaku ukośnika — w analizowanym przykładzie towarzyszy on drugiej etykiecie nazwy domenowej. Wpisy te są zazwyczaj przygotowywane przez dostawcę usług internetowych, który definiuje w ten sposób delegacje dla niepokrywających się zakresów adresów. Klient może wówczas utworzyć własne odwzorowania w ramach strefy 128/25.136.17.12.
↪ in-addr.arpa. Prześledzenie delegacji sprowadza się do wykonania następujących instrukcji:

```
C:\> nslookup
Serwer domyślny: gw
Adres: 10.0.0.1

> server f.in-addr-servers.arpa
Serwer domyślny: f.in-addr-servers.arpa
Adres: 193.0.9.1

> set type=ptr
> 129.128/25.136.17.12.in-addr.arpa.
Serwer: f.in-addr-servers.arpa
Adres: 193.0.9.1

12.in-addr.arpa nameserver = dbru.br.ns.els-gms.att.net
12.in-addr.arpa nameserver = cbru.br.ns.els-gms.att.net
12.in-addr.arpa nameserver = cmtu.mt.ns.els-gms.att.net
12.in-addr.arpa nameserver = dmtu.mt.ns.els-gms.att.net

> server dbru.br.ns.els-gms.att.net.
Serwer domyślny: dbru.br.ns.els-gms.att.net
Adres: 199.191.128.106

> 129.128/25.136.17.12.in-addr.arpa.
128/25.136.17.12.in-addr.arpa nameserver = ns2.intel-research.net
128/25.136.17.12.in-addr.arpa nameserver= ns1.intel-research.net

> server ns1.intel-research.net.
Serwer: ns1.intel-research.net
Adres: 12.155.161.131

> 129.128/25.136.17.12.in-addr.arpa.
129.128/25.136.17.12.in-addr.arpa
                    name = dmz.slouter.seattle.intel-research.net
128/25.136.17.12.in-addr.arpa
                    nameserver = bldmzsvr.berkeley.intel-research.net
```

```
128/25.136.17.12.in-addr.arpa
                    nameserver = sldmzsvr.intel-research.net
bldmzsvr.berkeley.intel-research.net internet address = 12.155.161.131
sldmzsvr.intel-research.net internet address = 12.17.136.131
```

Celem przeprowadzonego testu było ustalenie nazwy komputera o adresie IPv4 12.17.136.129. Z wcześniejszych rozważań wiadomo, że jednostce odpowiada rekord CNAME odnoszący się do nazwy kanonicznej 129.128/25.136.17.12.in-addr.arpa. Z tego względu pierwsze działanie polegało na skierowaniu resolvera do jednego z głównych serwerów (F) i ustawienie typu zapytania na PTR. W kolejnym kroku do wskazanego serwera zostało wysłane żądanie odwzorowania nazwy 129.128/25.136.17.12.inaddr.arpa. Serwer główny nie ma poszukiwanych informacji, ale zwraca nazwy serwerów obsługujących domenę 12.in-addr.arpa. Następnie to samo żądanie zostało wysłane do jednego z wymienionych serwerów (D8RU). Jego wynikiem jest lista złożona z nazw dwóch innych serwerów (ns1 i ns2). Po wybraniu jednego z nich możliwe jest skierowanie zapytania o rekord PTR, które dostarczy informacji o powiązaniu adresu i nazwy. Poszukiwana nazwa to dmz.slouter.seattle.intel-research.net.

11.5.6.6. Rekordy pełnomocnictw (SOA)

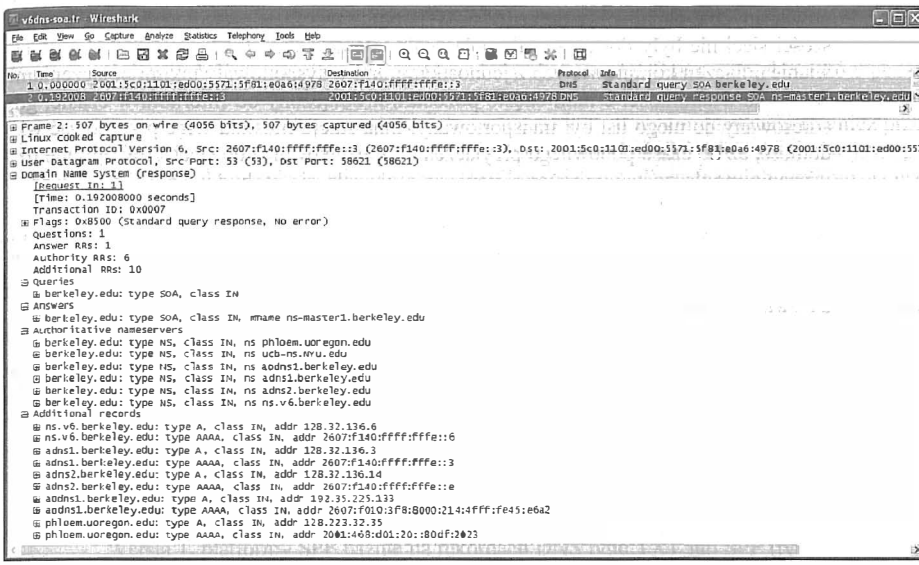
Każda strefa systemu DNS musi dysponować rekordem pełnomocnictwa (*authority*) zdefiniowanym jak rekord RR typu SOA (*Start of Authority*). Rekordy SOA zapewniają tworzenie zaufanych powiązań między obszarami przestrzeni nazw DNS oraz między serwerami, które udostępniają informacje na temat stref. Rekord RR SOA pozwala na określenie nazwy jednostki, która przechowuje oficjalną trwałą bazę danych informacji o strefie. Odpowiada także za udostępnianie adresu e-mail osoby odpowiedzialnej za konfigurację (znak @ w adresie jest zastępowany znakiem kropki), wyznaczenie parametrów aktualizacji strefy, a także domyślnego czasu ważności danych. Domyślna wartość TTL odnosi się do wszystkich rekordów RR, którym ten parametr nie został przypisany wprost.

Parametry aktualizacji strefy obejmują takie ustawienia jak numer seryjny konfiguracji, czas odświeżania, czas powtarzania oraz czas wygasania danych. Numer seryjny jest zwiększany (co najmniej o jeden) przez administratora sieci za każdym razem, gdy wprowadza jakiekolwiek zmiany w treści strefy. Wartość ta jest wykorzystywana przez serwery zapasowe do ustalenia, czy konieczny jest transfer strefy (transfer jest konieczny wtedy, gdy serwer zapasowy przechowuje dane strefy o niższym numerze seryjnym). Czas odświeżania informuje serwer zapasowy o tym, w jakich interwałach powinien sprawdzać rekord SOA serwera podstawowego, a szczególnie numer seryjny konfiguracji (aby w razie konieczności rozpocząć transfer strefy). Czasy powtarzania i wygasania są istotne w przypadku problemów w komunikacji serwera zapasowego z podstawowym. Wartość czasu powtarzania wyznacza liczbę sekund oczekiwania przed ponowną próbą nawiązania połączenia. Czas wygasania danych (wyrażony w sekundach) wyznacza czas, w którym próby są ponawiane. Jeśli w tym czasie połączenie nie zostanie zrealizowane, serwer zapasowy przestaje odpowiadać na zapytania związane z daną strefą. Informacje gromadzone w strefie mogą dotyczyć zarówno protokołu IPv4, jak i protokołu IPv6. Również do ich pozyskiwania można używać dowolnej wersji protokołu IP. W kolejnym z prezentowanych przykładów pokazano działanie mechanizmu w protokole IPv6 (wykorzystano do tego celu polecenie nslookup pracujące w systemie Windows wyposażonym jedynie w interfejsy IPv6).

```
C:\> nslookup
Serwer domyślny: gw
Adres: fe80::204:5aff:fe9f:9e80

> set type=soa
> berkeley.edu.
Serwer: gw
Adres: fe80::204:5aff:fe9f:9e80
Nieautorytatywna odpowiedź:
berkeley.edu
        primary name server = ns-master1.berkeley.edu
        responsible mail addr = hostmaster.berkeley.edu
        serial = 2009050116
        refresh = 10800 (3 hours)
        retry = 1800 (30 mins)
        expire = 3600000 (41 days 16 hours)
        default TTL = 300 (5 mins)

>server adns1.berkeley.edu.
Serwer domyślny: adns1.berkeley.edu
Adresy: 2607:f140:ffff:fffe::3
        128.32.136.3

> berkeley.edu.
Serwer: adns1.berkeley.edu
Adresy: 2607:f140:ffff:fffe::3
        128.32.136.3

berkeley.edu
        primary name server = ns-master1.berkeley.edu
        responsible mail addr = hostmaster.berkeley.edu
        serial = 2009050116
        refresh = 10800 (3 hours)
        retry = 1800 (30 mins)
        expire = 3600000 (41 days 16 hours)
        default TTL = 300 (5 mins)

berkeley.edu    nameserver = ns.v6.berkeley.edu
berkeley.edu    nameserver = aodns1.berkeley.edu
berkeley.edu    nameserver = adns2.berkeley.edu
berkeley.edu    nameserver = phloem.uoregon.edu
berkeley.edu    nameserver = adns1.berkeley.edu
berkeley.edu    nameserver = ucb-ns.NYU.edu
ns.v6.berkeley.edu      internet address = 128.32.136.6
ns.v6.berkeley.edu      AAAA IPv6 address = 2607:f140:ffff:fffe::6
adns1.berkeley.edu      internet address = 128.32.136.3
adns1.berkeley.edu      AAAA IPv6 address = 2607:f140:ffff:fffe::3
adns2.berkeley.edu      internet address = 128.32.136.14
adns2.berkeley.edu      AAAA IPv6 address = 2607:f140:ffff:fffe::e
aodns1.berkeley.edu     internet address = 192.35.225.133
aodns1.berkeley.edu     AAAA IPv6 address = 2607:f010:3f8:8000:214:4fff:fe45:e6a2
phloem.uoregon.edu      internet address = 128.223.32.35
phloem.uoregon.edu      AAAA IPv6 address = 2001:468:d01:20::80df:2023
```

W wyniku testu zwrócony został nie tylko rekord SOA, ale również lista sześciu autorytatywnych serwerów nazw oraz adresy IPv4/IPv6 (doklejone rekordy) pięciu z nich (adres serwera NYU nie został podany w formie rekordu doklejonego [NYU.edu], ponieważ

należy do innej strefy, zarządzanej przez inny serwer DNS). Ponieważ jest to jedna z najciekawszych odpowiedzi uzyskanych w prezentowanych przykładach, przeanalizujmy nieco dokładniej zawartość pakietu stanowiącego odpowiedź na żądanie dostarczone do autorytatywnego serwera nazw `adns1.berkeley.edu` (patrz rysunek 11.14).



Rysunek 11.14. *Odpowiedź na żądanie dostarczenia rekordu SOA z użyciem adresu IPv6. Odpowiedź uwzględnia zarówno adresy IPv4, jak i adresy IPv6*

Przechwycone zostały dwa pakiety, ale ponieważ odpowiedź (pakiet 2.) jest zdecydowanie ciekawsza, to ona zostanie szczegółowo omówiona. Zapytanie o rekord RR SOA zostało przesłane do jednostki o adresie `2607:f140:ffff:fffe::3` (`adns1.Berkeley.EDU`) z globalnego adresu IPv6 jednostki lokalnej `2001:5c0:1101:ed00:5571:5f81:e0a6:4978`. Za dostarczenie wyników odpowiada datagram IPv6 o całkowitej długości 491 bajtów (parametr *Długość pola danych* ma wartość 451 bajtów). Cały datagram składa się z nagłówka IPv6 (40 bajtów), nagłówka UDP (8 bajtów) oraz komunikatu DNS (443 bajty). W komunikacie DNS występują: jedno zapytanie, jedna odpowiedź, sześć rekordów RR pełnomocnictw oraz dziesięć dodatkowych rekordów RR.

Sekcja zapytania obejmuje etykiety `berkeley` i `edu`, zajmujące 18 bajtów. W sekcji odpowiedzi znajdują się wszystkie ważne informacje na temat domeny `berkeley.edu`. Treść sekcji zapytania pozwala na wykorzystanie etykiet kompresji w treści odpowiedzi. Całkowity rozmiar sekcji odpowiedzi wynosi 58 bajtów. Sekcja pełnomocnictw składa się z sześciu rekordów NS identyfikujących serwery nazw. Definicja ta zajmuje następne 135 bajtów. W sekcji informacji dodatkowych zapisanych zostało pięć rekordów A oraz pięć rekordów AAAA o całkowitej długości 220 bajtów. Rozmiar pojedynczego pola *RDATA* rekordu AAAA wynosi 16 bajtów — mimo że adres IPv6 można zapisać tekstowo w skróconym formacie z użyciem znaków `::`, format ten nie jest stosowany do przenoszenia adresów w pakietach. Obowiązują pełne 128-bitowe formy notacji.

11.5.6.7. Rekordy przekaźnika poczty (MX)

Rekord MX dostarcza informacji na temat **przekaźnika poczty**, czyli jednostki z uruchomionym **prostym protokołem transferu poczty** (SMTP — *Simple Mail Transfer Protocol*) [RFC5321], która odbiera pocztę elektroniczną kierowaną do użytkowników powiązanych z daną nazwą domenową. W początkowej fazie rozwoju Internetu w większości sieci nie były dostępne stałe połączenia internetowe, więc konieczne było ustanawianie połączeń komutowanych z jednostkami, które dysponowały takimi połączeniami. Odbiorca wiadomości e-mail często pozostawał odłączony od Internetu w czasie, gdy adresowany do niego list był transportowany. Inna stacja musiała więc przechować wiadomość, aż do czasu ponownego przyłączenia się użytkownika do sieci. Aby opisane rozwiązanie mogło zostać wdrożone, konieczne okazało się rozbudowanie listy rekordów DNS o element MX. Dzięki niemu stacja nadawcza może pozostawić list w urządzeniu pośredniczącym (w „przekaźniku poczty") w czasie, gdy końcowy odbiorca był niedostępny. Obecnie rekordy MX nadal są wykorzystywane, ponieważ dostarczanie wiadomości do jednostek wymienionych w rekordach MX jest preferowaną formą transferu poczty.

Rekordy MX uwzględniają wartość **priorytetu**, dzięki któremu w jednej domenie można zdefiniować wiele rekordów MX. Priorytet umożliwia stacji nadawczej posortowanie odbiorców w odpowiedniej kolejności (im mniejsza wartość, tym wyższy priorytet serwera poczty) i wybranie jednego z serwerów poczty. Aby pobrać listę rekordów MX związanych z daną nazwą domenową, wystarczy wykonać polecenie `host`. Oto przykład jego użycia w odniesieniu do domeny `cs.ecla.edu`:

```
Linux% host -t MX cs.ucla.edu ns3.dns.ucla.edu
Using domain server:
Name: ns3.dns.ucla.edu
Address: 192.35.210.7#53
Aliases:

cs.ucla.edu mail is handled by 13 Mailman.cs.ucla.edu.
cs.ucla.edu mail is handled by 3 Moa.cs.ucla.edu.
cs.ucla.edu mail is handled by 13 Pelican.cs.ucla.edu.
```

Z listingu wynika, że poczta wysyłana na adres `osoba@cs.ucla.edu` jest przetwarzana przez jeden z trzech serwerów pocztowych skonfigurowanych w systemie DNS. Wszystkie wymienione serwery należą do domeny `cs.ucla.edu`, mimo że nie ma obowiązku umieszczania serwerów pocztowych w tej samej domenie, za którą odpowiadają. Trzy wspomniane serwery są podzielone na dwie grupy — jedną o priorytecie 3 i jedną o priorytecie 13. Jednostka o niższej wartości jest jednostką preferowaną, więc nadawca najpierw spróbuje dostarczyć pocztę do serwera `Moa.cs.ucla.edu`. Jeśli operacja się nie powiedzie, wybrany zostanie jeden z serwerów `Pelican` lub `Mailman` (losowo).

Oczywiście, istnieje ryzyko, że żadna ze stacji wskazywanych przez rekordy MX nie jest dostępna. Jest to jednak stan błędu. Dopuszczalna jest również sytuacja, w której w pliku strefy nie będzie żadnych rekordów MX, ale zostaną zdefiniowane rekordy CNAME, A lub AAAA odnoszące się do nazwy domeny. W przypadku zapisania wspomnianego rekordu CNAME zamiast pierwotnej nazwy domeny wykorzystana zostanie nazwa wskazywana przez rekord CNAME. Jeśli w strefie występują rekordy A lub AAAA, programy pocztowe mogą się komunikować z serwerami o adresach wskazywanych przez

te rekordy. Każdy z nich ma wówczas zerowy priorytet (i jest nazywany **niejawnym rekordem MX**). Elementami wskazywanymi przez rekordy MX muszą być nazwy domenowe wymienione w rekordach A lub AAAA. Nie można natomiast używać do tego celu rekordów CNAME [RFC5321].

11.5.6.8. Walka ze spamem — rekordy SPF i TXT

Rekordy MX umożliwiają wyszukiwanie w systemie DNS nazw przekaźników poczty i serwerów poczty obsługujących daną domenę. Jest to konieczne podczas wysyłania wiadomości e-mail. Niemniej ostatnio system DNS jest wykorzystywany również przez odbiorcze serwery pocztowe do ustalania, który z przekaźników lub serwerów jest upoważniony do wysyłania wiadomości z określonej domeny. Rozwiązanie to wspomaga walkę ze spamem (niechcianą pocztą), czyli z pocztą wysyłaną przez jednostki udające serwery uprawnione do dystrybuowania wiadomości e-mail.

Odbierana przez serwer poczta jest odrzucana, magazynowana lub przekazywana do kolejnego serwera e-mail. Powodów odrzucenia może być bardzo dużo — błąd stosu protokołów, brak odpowiedniej przestrzeni dyskowej itp. Jednym z nich może być również to, że stacja kliencka nie jest jednostką uprawnioną do wysyłania poczty. Funkcja ta jest elementem **platformy definiowania polityki nadawcy** (SPF — *Sender Policy Framework*) opisanej w dokumencie [RFC4408] (mającym status zalecenia eksperymentalnego). Mechanizmy SPF są również wdrożone w innym rozwiązaniu, nazywanym po prostu **identyfikatorem nadawcy** (*Sender ID*) [RFC4406]. Także ten system ma status eksperymentalnego, a ponadto jest znacznie mniej rozpowszechniony.

Wersja 1. platformy SPF bazuje na rekordach DNS typu TXT lub SPF (typ 99). Rekordy te są zapisywane w systemie DNS przez właściciela domeny. Ich celem jest wskazywanie serwerów uprawnionych do wysyłania poczty z danej domeny. Choć użycie rekordu SPF jest właściwszym sposobem implementacji opisywanego mechanizmu, trzeba pamiętać, że niektóre stacje klienckie nie obsługują poprawnie tego typu rekordów. Aby uniknąć komplikacji, stosuje się rekordy TXT, które przechowują prosty ciąg tekstowy skojarzony z nazwą domeny. Pierwotnie rejestrowane dane miały format łatwy do odczytania przez człowieka, co ułatwiało rozwiązywanie problemów oraz identyfikowanie właściciela bądź lokalizacji domeny. Obecnie zawartość rekordów jest analizowana przede wszystkim przez programy, takie jak aplikacje SPF.

Składnia rekordów SPF bywa dość skomplikowana, ale pozwala na zdefiniowanie kryteriów, które muszą być spełnione zarówno przez odbieraną wiadomość, jak i przez połączenie, w którym jest dostarczana. Przykładowo uniwersytet w Berkeley korzysta z następującego wpisu SPF (niektóre wiersze zostały przełamane w celu zwiększenia czytelności):

```
Linux% host -t txt berkeley.edu
berkeley.edu descriptive text
        "v=spf1 ip4:169.229.218.128/25 ip6:2607:F140:0:1000::/64
        include:outboundmail.convio.net ~all"
```

Informacja zawarta w rekordzie jest przeznaczona dla wersji 1. mechanizmu SPF (o czym świadczy ciąg `v=spf1`) i odnosi się do rekordu TXT. Odbierając pocztę, oprogramowanie e-mail zakłada, że pochodzi ona z domeny `berkeley.edu`. Wykonuje więc zapytanie DNS i żąda dostarczenia rekordów typu TXT z domeny `berkeley.edu`. Treść rekordu definiuje

kryteria dopasowania (nazywane **mechanizmami**) oraz dodatkowe informacje (nazywane **modyfikatorami**). Definicję każdego mechanizmu poprzedza **kwalifikator**, który określa sposób postępowania po dopasowaniu rekordu. Przetwarzanie rekordów SPF rozpoczyna się od wywołania funkcji `check_host()`. Jej zadanie polega na sprawdzeniu poszczególnych mechanizmów i zakończeniu działania po odszukaniu pierwszego pasującego. Wynik działania funkcji `check_host()` odpowiada jednej z wartości: `None`, `Neutral`, `Pass`, `Fail`, `SoftFail`, `TempError` oraz `PermError`. Wartości `None` i `Neutral` świadczą o tym, że informacje potrzebne do działania mechanizmu nie były dostępne albo że — mimo ich dostępności — nie ustalono wyniku. Wartość `Pass` informuje o dopasowaniu (wartość ta została opisana w kolejnym akapicie). Wynik `Fail` oznacza, że stacja nie jest uprawniona do wysyłania poczty z danej domeny. Wartość `SoftFail` jest niejednoznaczna i zgodnie ze specyfikacją [RFC4408] należy ją traktować jako stan pośredni między `Fail` i `Neutral`. Wynik `TempError` informuje o chwilowym błędzie (np. komunikacyjnym), natomiast `PermError` wskazuje na trwały błąd w konfiguracji SPF, wynikający zazwyczaj z niewłaściwie zdefiniowanego rekordu TXT lub SPF w domenie.

Przedstawiony w poprzednim przykładzie tekst rekordu należy odczytywać od lewej do prawej strony. Pierwszy fragment (`v=spf1`) to modyfikator definiujący wersję 1. algorytmu SPF. Mechanizm `ip4` informuje, że nadawca SMTP ma adres IPv4 o prefiksie `169.229.218.128/25`. Mechanizm `ip6` wyróżnia jako stacje nadawcze wszystkie komputery o prefiksie `2607:F140:0:1000::/64`. Ostatni mechanizm (`include`) włącza (za pomocą referencji) rekordy TXT z domeny `outboundmail.convio.net`:

```
Linux% host -t txt outboundmail.convio.net
outboundmail.convio.net descriptive text
       "v=spf1 +ip4:66.45.103.0/25 +ip4:69.48.252.128/25
        +ip4:209.163.168.192/26 ~all"
outboundmail.convio.net descriptive text
        "spf2.0/pra
        +ip4:66.45.103.0/25 +ip4:69.48.252.128/25
        +ip4:209.163.168.192/26 ~all"
```

Rekordy TXT są wykorzystywane zarówno w rozwiązaniu SPF, jak i Sender ID (które jest oznaczane w sekcji wersji jako `spf2.0/pra`). Pierwszy rekord znajduje zastosowanie w systemie SPF. Kwalifikator `+` informuje, że w przypadku dopasowania wynikiem jest `Pass`. Jeśli definicja mechanizmu nie jest poprzedzona kwalifikatorem, przyjmuje się kwalifikator domyślny `+`. Inne oznaczenia kwalifikatorów to: `-` (`Fail`), `~` (`SoftFail`) oraz `?` (`Neutral`). Jeżeli żaden z mechanizmów nie spowoduje zwrócenia wyniku `Pass`, o rezultacie decyduje mechanizm końcowy (`all`) równoważny spełnieniu warunku. Znak tyldy (`~`) przed słowem kluczowym `all` oznacza, że w przypadku, w którym `all` jest jedynym dopasowanym mechanizmem, wynikiem operacji powinna być wartość `SoftFail`. Sposób interpretacji wyniku `SoftFail` zależy od konkretnego programu odbioru poczty. Warto zwrócić uwagę na to, że nawet przy zastosowaniu techniki SPF weryfikacją objęta jest domena nadawcza oraz system nadawczy, a nie użytkownik. W rozdziale 18. zostało opisane rozwiązanie DKIM, w którym funkcje zbliżone do SPF są realizowane przez mechanizmy kryptografii i uwierzytelniania.

11.5.6.9. Pseudorekordy opcji (OPT)

W zaleceniu [RFC2671] wraz z definicją rozszerzenia EDNS0 opisany został rekord **OPT pseudoRR**. Dodanie określenia „pseudo" wynika z tego, że rekord odnosi się jedynie do treści pojedynczego komunikatu DNS i nie ma charakteru klasycznego rekordu RR. Z tego powodu rekordy OPT nie są buforowane, przekazywane oraz trwale zapisywane. W jednym komunikacie DNS może wystąpić tylko jeden taki rekord. Jeśli jest uwzględniony, musi być przenoszony w sekcji informacji dodatkowych.

Rekord OPT składa się z 10-bajtowego stałego fragmentu oraz części o zmiennej długości. Część stała jest podzielona na 16-bitowy identyfikator typu RR (41), 16-bitowy rozmiar pola danych UDP, 32-bitowy obszar rozszerzonego kodu *RCODE* i znaczników, a także 16-bitową wartość odzwierciedlającą rozmiar zmiennej części rekordu. Wymienione pola znajdują się na tych samych pozycjach, na których w klasycznych rekordach RR przenoszone są pola *Nazwa*, *Typ*, *Klasa*, *TTL* oraz *RDLEN* (patrz rysunek 11.8). Rekordy OPT nie mają zdefiniowanego pola *Nazwy* (przechowują 0 bajtów). W obszarze rozszerzonego kodu *RCODE* i *Znaczników* (32-bity odpowiadające wartości *TTL* z rysunku 11.8) wydzielono dodatkowo 8-bitowy fragment przeznaczony na osiem dodanych bitów pola *RCODE* (przedstawionego na rysunku 11.3) oraz 8-bitowe pole *Wersji* (w rozszerzeniu EDNS0 ustawione trwale na 0). Pozostałe 16 bitów nie zostało zdefiniowanych — muszą być ustawione na 0. Dodanie 8 bitów kodu wyniku pozwoliło na rozbudowanie listy typów błędów DNS o wartości przedstawione w tabeli 11.4 (wartość 16 została zdefiniowana w dwóch różnych dokumentach RFC).

Tabela 11.4. *Rozszerzona lista wartości RCODE. Większość odnosi się do rozszerzeń zwiększających bezpieczeństwo protokołu*

Wartość	Nazwa	Specyfikacja	Opis i przeznaczenie
16	BADVERS	[RFC2671]	Błędna wersja EDSN
16	BADSIG	[RFC2845]	Błędna sygnatura TSIG (patrz rozdział 18.)
17	BADKEY	[RFC2845]	Błędny klucz TSIG (patrz rozdział 18.)
18	BADTIME	[RFC2845]	Błędna sygnatura TSIG (problem z czasem; patrz rozdział 18.)
19	BADMODE	[RFC2930]	Błędny tryb TKEY (patrz rozdział 18.)
20	BADNAME	[RFC2930]	Zduplikowana nazwa klucza (patrz rozdział 18.)
21	BADALG	[RFC2930]	Nieobsługiwany algorytm (patrz rozdział 18.)

Zgodnie z wcześniejszymi informacjami rekord OPT obejmuje pole *RDATA* o zmiennej długości. Jest ono przeznaczone do przechowywania rozszerzalnej listy par atrybut-wartość. Bieżący zestaw atrybutów, ich znaczenie oraz źródło RFC są rejestrowane przez organizację IANA [DNSPARAMS]. Jedna z opcji (o nazwie NSID; opcja EDNS o kodzie 3) [RFC5001] operuje specjalnym identyfikatorem serwera generującego odpowiedź. Format pola nie jest określony w standardzie i może być ustalony przez administratora serwera DNS. Opcja ta okazuje się użyteczna, gdy do wskazania grupy serwerów wykorzystany zostanie adresu emisji dowolnej (anycast). Wartość NSID pozwala na identyfikację serwera na podstawie wartości innej niż adres IP nadawcy. Więcej przykładów zastosowania rekordu OPT i rozszerzenia EDNS0 zostało przedstawionych w rozdziale 18.

11.5.6.10. Rekordy usług (SRV)

Rekord **usługi** (SRV — *service*) został zdefiniowany w dokumencie [RFC2782]. Jest on uogólnieniem rekordu MX umożliwiającym opisywanie stacji, protokołów i numerów portów potrzebnych do skontaktowania się z określoną usługą. Zapis rekordu odpowiada zazwyczaj przedstawionemu poniżej:

```
_Usługa._Protokół.Nazwa TTL IN SRV Priorytet Waga Port Cel
```

Element *Usługa* jest oficjalną nazwą usługi. Z kolei *Protokół* definiuje protokół warstwy transportowej wykorzystywany w komunikacji z usługą, zazwyczaj jest to TCP lub UDP. Wartość *TTL* odpowiada klasycznej wartości czasu ważności rekordu, a słowa kluczowe IN i SRV określają klasę (internetową) i typ (SRV) rekordu. Parametr *Priorytet* jest 16-bitową wartością, o takim samym przeznaczeniu jak priorytety rekordów MX (im niższa wartość, tym wyższy priorytet). Parametr *Waga* służy do wyboru jednego z rekordów RR o takich samych priorytetach. Pozwala on na zdefiniowanie ważonego mechanizmu rozkładania obciążenia, w którym większa waga oznacza większe prawdopodobieństwo wyboru określonego rekordu. Element *Port* odpowiada numerowi portu protokołu TCP lub UDP (lub innego protokołu transportowego). Wartość *Cel* jest nazwą domenową jednostki udostępniającej usługę. Z kolei ciąg *Nazwa* jest nazwą domeny, w której pracuje usługa. Jednym z powodów stosowania rekordów SRV jest możliwość wyróżniania poszczególnych serwerów udostępniających tę samą usługę.

Gdyby np. klient chciał uzyskać informację o tym, jaki jest adres i port komputera z usługą LDAP pracującego w protokole TCP w domenie *przyklad.pl*, mógłby przesłać zapytanie o rekord SRV z nazwą domenową `_ldap._tcp.przyklad.pl`. Oto praktyczny przykład:

```
Linux% host -t srv _ldap._tcp.openldap.org
_ldap._tcp.openldap.org has SRV record 0 0 389 www.openldap.org.
```

Celem operacji jest wyszukanie serwera udostępniającego usługę **lekkiego protokołu usług katalogowych** (LDAP — *Lightwieght Directory Access Protocol*) [RFC4510] w protokole TCP w ramach domeny `openldap.org`. Uzyskany wynik świadczy o tym, że usługa jest utrzymywana na serwerze `www.openldap.org` na porcie TCP 389 (na domyślnym porcie usługi LDAP). Priorytet i waga mają wartości zerowe, co oznacza, że nie ma serwerów alternatywnych.

W zaleceniu [RFC2782] nie uwzględniono nowo zarejestrowanych (przez IANA) wartości *Usługa* i *Protokół* rekordu SRV. Dlatego nazwy muszą odpowiadać nazwom zdefiniowanym w rejestrze „Service Name and Transport Protocol Port Number" [ISPR], a jako wartości protokołu można używać jedynie specyfikatorów `_tcp` lub `_udp`. Zdarzają się jednak odstępstwa od tych reguł. W dokumencie [RFC5509] opisano sposób wykorzystania rekordów SRV do obsługi protokołu SIP. Zdefiniowano w nim następujące wartości parametrów *Usługa* i *Protokół*: `_im._sip` oraz `_pres._sip`. W zaleceniu [RFC6186] są z kolei wymienione wartości parametru *Usługa*, które ułatwiają programom pocztowym wyszukiwanie serwerów IMAPS, SMTP, IMAP oraz POP3 (dwie pierwsze usługi są preferowane przez większość klientów e-mail), są to: `_submission`, `_imap`, `_imaps`, `_pop3` oraz `_pop3s`. Mimo że zalecenie nie narzuca obowiązku powiązania wymienionych nazw z protokołem TCP (za pomocą parametru *Protokół*), w praktyce jest to jedyne dostępne rozwiązanie. Przy użyciu rekordów SRV użytkownik, konfigurując nowy **program pocztowy** (MUA — *Mail User Agent*), mógłby określić jedynie domenę (`przyklad.pl`).

Pozostałe informacje program pocztowy uzyskałby po wysłaniu zapytania DNS z podaniem chociażby następujących nazw: `_submission._tcp.przyklad.pl` oraz `_imaps._tcp.` `➥przyklad.pl`.

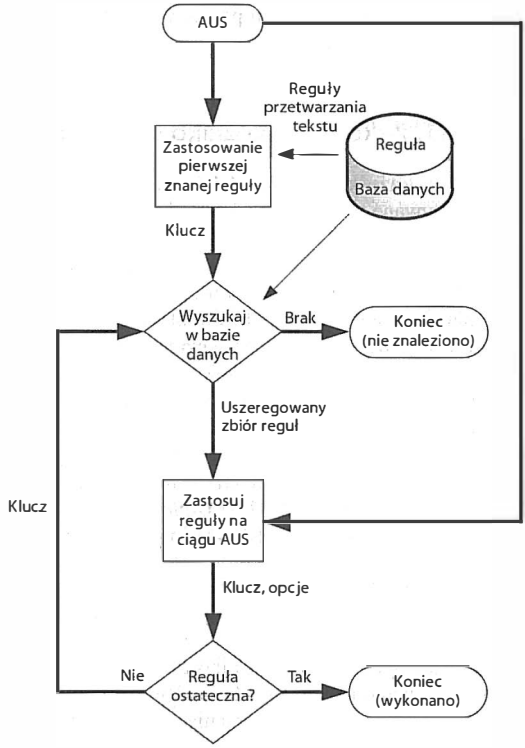
11.5.6.11. Rekordy wskaźnika do właściciela nazwy

Rekordy **wskaźnika do właściciela nazwy** (NAPTR — *Name Authority Pointer*) są wykorzystywane w serwerach DNS obsługujących **system wykrywania dynamicznych delegacji** (DDDS — *Dynamic Delegation Discovery System*) [RFC3401]. DDDS jest ogólnym, abstrakcyjnym algorytmem przekształcania ciągów tekstowych (dostarczanych przez aplikacje) za pomocą dynamicznie pozyskiwanych reguł przetwarzania. Wyniki jego działania są zazwyczaj używane do lokalizowania zasobów sieciowych. Każda aplikacja DDDS dostosowuje ogólne mechanizmy DDDS do własnych potrzeb. W działaniach wykorzystuje bazę danych reguł oraz zbiór algorytmów, które pozwalają na odpowiednie formatowanie ciągów tekstowych. Jedną z takich baz danych jest system DNS [RFC3403], który przechowuje reguły przekształceń w rekordach typu NAPTR. Przykładem aplikacji DDDS odwołującej się do serwerów DNS może być system konwertowania międzynarodowych numerów telefonicznych do formatu opisanego w standardzie **ujednoliconego identyfikatora zasobów** (URI — *Uniform Resource Identifier*) [RFC3986] i zapisywania ich zgodnie z rozwiązaniem ENUM (patrz podpunkt 11.5.6.12).

W systemie DDDS **algorytm** [RFC3402] określa zasady przetwarzania **niepowtarzalnego ciągu aplikacji** (AUS — *Application-Unique String*), wykorzystując do tego celu reguły przechowywane w bazie danych. Wynikiem może być jedynie **ciąg ostateczny** (*terminal string*; pełny wynik) albo ciąg nieostateczny (używany do pobrania następnej reguły przetwarzania ciągu AUS). Zbiór odpowiednich reguł pozwala na utworzenie efektywnego systemu przetwarzania tekstu, który umożliwia zakodowanie niemal każdej treści o regularnej składni. Najważniejsze cechy algorytmu zostały przedstawione na rysunku 11.15.

Zilustrowany na rysunku 11.15 proces rozpoczyna się od użycia pierwszej znanej reguły do zmodyfikowania ciągu AUS (jednoznacznie skojarzonego z każdą aplikacją). Wynikiem jest klucz wskazujący kolejną regułę w bazie danych. Reguły składają się ze wzorców przepisywania tekstu oraz towarzyszących im opcji. Są one używane do operacji na ciągu AUS, ale nigdy do przetwarzania wyników przepisywania. Konkretny sposób działania mechanizmu zależy od aplikacji, ale zazwyczaj reguły są wyrażeniami regularnymi, podobnymi do tych, które są dostępne w uniksowym programie `sed` [DR97]. W przypadku użycia systemu DNS jako bazy danych DDDS [RFC3403] kluczami są nazwy domenowe, a reguły są przechowywane w rekordach NAPTR. Każdy rekord NAPTR składa się z następujących pól: *Kolejność*, *Priorytet*, *Opcje*, *Usługi*, *Wyrażenie regularne* oraz *Ciąg zastępczy*.

Pole *Kolejność* to 16-bitowa liczba całkowita bez znaku wyznaczająca kolejność użycia poszczególnych rekordów NAPTR (niższe wartości poprzedzają wyższe) — system DNS nie gwarantuje sortowania zbiorów rekordów. Wartość *Priorytet* określa kolejność przetwarzania reguł o takiej samej wartości pola *Kolejność*. Pole *Kolejność* narzuca obowiązkowy porządek sortowania rekordów RR, natomiast numer priorytetu ma charakter uzupełniający. Pole *Opcji* przechowuje nieuporządkowaną listę pojedynczych znaków z przedziału od A do Z (bez rozróżniania wielkości liter) oraz od 0 do 9. Zasady



Rysunek 11.15. *Ogólny schemat działania algorytmu DDDS. Rekordy nieostateczne mogą inicjować pętle. Każda iteracja wiąże się z wykonaniem pewnej operacji przetwarzania ciągu aplikacji*

interpretacji treści pola pozostają w gestii aplikacji korzystającej z rekordów NAPTR (wspomniane opcje są np. używane w rozwiązaniu ENUM, opisanym w kolejnym podpunkcie). Pole *Usługi* jest uzupełniane przez aplikację do przedstawienia usługi, która to pole opisuje. Pole *Wyrażenie regularne* przechowuje wyrażenie definiujące operację podmiany tekstu, które po zastosowaniu na ciągu AUS generuje ciąg wynikowy (przypadek ostateczny) lub identyfikator kolejnego serwera, z którego pobierane są rekordy NAPTR (przypadek nieostateczny). *Ciąg zastępczy* (definiowany tylko w przypadku braku wyrażenia regularnego) wskazuje następny serwer do odpytania. Wartość pola jest zapisywana jako osobna nazwa FQDN (bez kompresji). Dostępność dwóch wzajemnie wykluczających się pól wynika ze sposobu prowadzenia prac nad rekordami NAPTR w przeszłości.

Aby lepiej zrozumieć sposób wykorzystania rekordów NAPTR w praktyce, warto przeanalizować działanie aplikacji ENUM i SIP DDDS, aplikacji URI/URN DDDS oraz rozwiązań alternatywnych S-NAPTR i U-NAPTR. Budowa systemu DDDS wiąże się z określeniem ciągu AUS, pierwszej znanej reguły, spodziewanych danych wyjściowych, prawidłowej bazy danych, opcji oraz parametrów usług.

11.5.6.12. Systemy ENUM i SIP

W rozwiązaniu ENUM DDDS [R06] [RFC6116] [RFC6117] [RFC5483], przeznaczonym do przekształcania numerów telefonicznych w ciągi URI, wartość AUS ma format numeru telefonicznego zgodnego ze standardem E.164 (maksymalnie 15 cyfr poprzedzonych znakiem +). Początkowy znak + wyróżnia numery E.164 spośród innych wartości liczbowych jako nadające się do przetwarzania za pomocą mechanizmu ENUM DDDS. Pierwsza znana reguła usuwa z ciągu AUS wszelkie spacje oraz znaki inne niż cyfry. Bazą danych jest system DNS, w którym kluczami są nazwy domenowe utworzone na podstawie ciągu AUS (który po opisanej operacji składa się jedynie z cyfr). Pomiędzy cyfry wstawiany jest znak kropki. Następnie ciąg jest odwracany i uzupełniany o sufiks `.e164.arpa`. Przykładowo numer E.164 `+48-52-5679900` zostałby przekształcony w ciąg `0.0.9.9.7.6.5.2.5.8.4.e164.arpa`. Powstała nazwa domenowa służy do pobrania rekordów NAPTR.

Wynikiem końcowym (uzyskanym prawdopodobnie po kilku iteracjach) pokazanego na rysunku 11.15 algorytmu DDDS jest bezwzględny (a nie względny) adres URI. Jedyna zdefiniowana opcja *U* oznacza regułę ostateczną, która generuje wartość URI. Brak opcji oznacza, że reguła ma charakter nieostateczny i jest niekiedy nazywana **nieostatecznym rekordem NAPTR** (NTN — *Non-Terminal NAPTR*). Parametry usługi są zapisane w polu *Usługa* rekordu NAPTR i mają format `E2U+Usługa`, powstały w wyniku użycia identyfikatora E2U (odzwierciedlającego przekształcenie ciągu E.164 na URI) oraz nazwy usługi skojarzonej z danym numerem. Razem pola te tworzą identyfikator nazywany *enumservice*, którego wartości są rejestrowane przez organizację IANA [ENUM] [RFC6117]. Opracowano wiele tego rodzaju identyfikatorów, w tym na potrzeby faksu, komunikatorów i wideokonferencji.

Aby sprawdzić, jak ten mechanizm działa, można przygotować zapytanie o numer uniwersytetu w Ostrawie (wiersze zostały przełamane w celu zwiększenia czytelności):

```
Linux% host -t naptr 1.1.1.1.1.5.8.3.7.0.2.4.e164.arpa
1.1.1.1.1.5.8.3.7.0.2.4.e164.arpa has NAPTR record
       50 50 "u" "E2U+sip" "!^\\+(.*)$!sip:\\1@osu.cz!" .
1.1.1.1.1.5.8.3.7.0.2.4.e164.arpa has NAPTR record
       100 50 "u" "E2U+sip""!^\\+(.*)$!sip:\\1@cesnet.cz!"
1.1.1.1.1.5.8.3.7.0.2.4.e164.arpa has NAPTR record
       200 50 "u" "E2U+h323" "!^\\+(.*)$!h323:\\1@gklext.cesnet.cz!"
```

Na listingu przedstawiono trzy rekordy NAPTR aplikacji ENUM DDDS odnoszące się do dwóch usług SIP i jednej usługi H.323 (wykorzystywanych w telefonii internetowej). Numery wyznaczające kolejność rekordów to 50, 100 (wpisy dotyczące telefonii SIP) oraz 200 (wpis odnoszący się do systemu H.323). Dzięki nim aplikacja ENUM może operować wieloma rekordami NAPTR skojarzonymi z jednym numerem telefonicznym, a administrator rekordów NAPTR może określić preferowaną kolejność odwołań do bram udostępniających tę samą usługę.

SIP jest protokołem organizacji IETF wykorzystywanym do sygnalizacji i znajduje szerokie zastosowanie w aplikacjach multimedialnych łączących jednostki klienckie z serwerami. Mechanizm H.323 jest protokołem opracowanym przez organizację ITU przeznaczonym do realizacji wideokonferencji i komunikacji głosowej. Obejmuje własny protokół sygnalizacyjny. Jest powszechnie implementowany w urządzeniach telekonferencyjnych. W poprzednim przykładzie, a także w przykładach kolejnych, program `host` generuje

wyniki, których można użyć jako danych do konfiguracji pliku strefy serwera DNS (np. usługi BIND). Z tego powodu występują w nich dodatkowe znaki odwrotnego ukośnika (widoczne tutaj jako symbole specjalne \\), których nie ma w odpowiedziach wysyłanych przez serwer DNS.

Aby zrozumieć zasady przetwarzania ciągów AUS za pomocą rekordów NAPTR, przyjrzyjmy się drugiemu z rekordów SIP przedstawionych na listingu. W odpowiedzi na zapytanie DNS komputer odbiera rekord NAPTR, w którym zawarte jest wyrażenie regularne (pomiędzy pierwszym i drugim znakiem !) odzwierciedlające kryteria dopasowania oraz zasadę podmiany tekstu. Ciąg +420738511111 jest więc porównywany z wyrażeniem regularnym `^\+(.*)$`. Ponieważ kryteria dopasowania są spełnione, odczytywana jest reguła przepisania ciągu: `sip:\1@cesnet.cz`. Specjalna zmienna `\1` jest zastępowana przez podciąg zgodny z wyrażeniem zapisanym w nawiasie. W analizowanym przypadku oznacza to wyodrębnienie całego ciągu AUS poza początkowym znakiem +. Ostatecznie ciąg +420738511111 jest przekształcany w ciąg `sip:420738511111@cesnet.cz`.

Kolejnym zadaniem aplikacji, po wyznaczeniu adresu URI, byłoby prawdopodobnie nawiązanie połączenia z serwerem SIP. Dane SIP nie mogą być jednak przenoszone przez dowolne protokoły transportowe. Dlatego konieczne jest wykorzystanie innej funkcji systemu DDDS przygotowanej specjalnie na potrzeby komunikacji SIP [RFC3263]. Odpowiada ona za zapisanie w rekordach nazw wartości docelowej, która wskazuje domenę z użytecznymi rekordami SRV. Oto ich treść:

```
Linux% host -t naptr cesnet.cz
cesnet.cz has NAPTR record 200 50 "s" "SIP+D2T" "" _sip._tcp.cesnet.cz.
cesnet.cz has NAPTR record 100 50 "s" "SIP+D2U" "" _sip._udp.cesnet.cz.
```

Tym razem w rekordzie NAPTR została zdefiniowana opcja `S` oznaczająca, że wynikiem jest rekord SRV. Ponieważ pole wyrażenia regularnego nie jest uwzględnione, wynikiem jest po prostu nazwa domenowa w formatach `SIP+D2`*x* lub `SIPS+ D2`*x*, w którym fragmenty `SIP` oraz `SIPS` wymuszają użycie protokołów SIP lub SIP z zabezpieczeniami (TLS; patrz rozdział 18.). Znak *x* określa z kolei rodzaj protokołu transportowego: `U` (UDP), `T` (TCP) lub `S` (SCTP) [RFC4960]. W analizowanym przykładzie aplikacja najpierw spróbowałaby wykorzystać rekord SRC odnoszący się do kombinacji protokołów SIP/UDP, a dopiero w razie problemów do protokołów SIP/TCP. Wpis odnoszący się do protokołu UDP ma niższą wartość priorytetu.

11.5.6.13. Odwzorowanie ciągów URI i URN

System ENUM jest prawdopodobnie pierwszą implementacją rekordów NAPTR mechanizmu DNS. Istnieją jednak również inne aplikacje DDDS przeznaczone do odwzorowywania ciągów URI [RFC3404] oraz do przechowywania adresów URI niezależnych od lokalizacji (nazywanych **ujednoliconymi nazwami zasobów** [URN — *Universal Resource Name*]) [RFC2141]. Wszystkie adresy URI (włącznie z URN) składają się z nazwy schematu oraz fragmentu właściwego dla określonego schematu. Bieżąca lista oficjalnych schematów jest utrzymywana przez organizację IANA [URI]. Aplikacje przetwarzające ciągi URI i URN są tak podobne w działaniu, że można je analizować razem. W systemach DDDS adresy URI i URN są ciągami AUS, dla których poszukiwany jest autorytatywny serwer. Pierwsza znana reguła aplikacji przetwarzających ciągi URI jest po prostu nazwą schematu. W rozwiązaniach operujących wartościami URN odpowiada natomiast identyfikatorowi przestrzeni nazw (fragmentowi występującemu po ciągu `urn:`,

a przed znakiem dwukropka). Przykładowo ciąg `http://www.pearson.com` jest adresem URI o schemacie (kluczu) `http`. Z kolei z wartości URN `urn:cos:przestrzen` jako klucz zostałby wyznaczony fragment `cos`. W czasie pisania książki wykorzystywane były cztery opcje: `S`, `A`, `U` oraz `P`. Pierwsze trzy mają charakter ostateczny i generują nazwę domenową, która pozwala pobrać rekord SRV, adres IP lub adres URI. Opcja `P` informuje, że działanie algorytmu DDDS musi zostać przerwane, ponieważ potrzebne jest przetwarzanie na poziomie aplikacji. Wszystkie opcje wzajemnie się wykluczają. Podobnie jak w rozwiązaniu ENUM, brak opcji oznacza wartość NTN.

Mechanizmy DDDS operujące wartościami URI i URN nadal są opracowywane. W czasie pisania książki (w 2011 roku) w domenie TLD `uri.arpa` zdefiniowane były następujące schematy:

```
Linux% host -t naptr http.uri.arpa
http.uri.arpa has NAPTR record 0 0 "" "" "!^http://([^:/?#]*).*$!\\1!i" .
Linux% host -t naptr ftp.uri.arpa
ftp.uri.arpa has NAPTR record 0 0 "" "" "!^ftp://([^:/?#]*).*$!\\1!i"
Linux% host -t naptr mailto.uri.arpa
mailto.uri.arpa has NAPTR record 0 0 "" "" "!^mailto:(.*)@(.*)$!\\2!i"
Linux% host -t naptr urn.uri.arpa
urn.uri.arpa has NAPTR record 0 0 "" "" "/urn.([^:]+)/\\1/i"
```

Trzy pierwsze z przedstawionych rekordów NAPTR zawierają reguły przepisywania tekstu, ale nie mają żadnych opcji. Wynika z nich, że aplikacja powinna wyodrębnić nazwę domenową z podanego ciągu URI i kontynuować wykonywanie algorytmu. Litera `i` po ostatnim znaku `!` informuje, że podczas dopasowywania nie należy uwzględniać wielkości liter. Przykładowo adres `mAiLto:osoba@przyklad.pl` po przepisaniu powinien mieć treść `przyklad.pl`. Czwarty rekord jest przeznaczony do wyodrębniania identyfikatora przestrzeni nazw z ciągu URN. Rekordów NAPTR związanych z ciągami URN nie zdefiniowano zbyt wiele — dwa w domenie *urn.arpa*:

```
Linux% host -t naptr pin.urn.arpa
pin.urn.arpa has NAPTR record 100 100 "" "" "" pin.verisignlabs.com.
Linux% host -t naptr uci.urn.arpa
uci.urn.arpa has NAPTR record 100 100 "" "" "" uci.or.kr.
```

Wydaje się, że obecnie przestrzeń nazw URN nie cieszy się szczególnym zainteresowaniem użytkowników i nie wiadomo, czy adresy URN będą powszechnie stosowane, ponieważ opracowano wiele alternatywnych metod określania lokalizacji obiektów z użyciem trwałych identyfikatorów (np. [P10]). Ponieważ jednak utworzonych zostało ponad 40 przestrzeni nazw URN, można sądzić, że społeczność internetowa będzie zainteresowana tworzeniem nowych, nawet jeśli tylko niektóre z nich zostaną odzwierciedlone w rekordach NAPTR.

11.5.6.14. Rekordy S-NAPTR i U-NAPTR

Ustalenie przez aplikację konkretnego adresu stacji, protokołu lub numeru portu usługi funkcjonującej w domenie często okazuje się nieco problematyczne. Przykładowo program pocztowy uruchomiony w systemie użytkownika w domenie `przyklad.pl` może poszukiwać serwera oferującego usługę IMAP. Zgodnie z obecnie obowiązującą konwencją nazwa domeny jest w takich przypadkach poprzedzana nazwą usługi (np. `imap.przyklad.com`). Rozwiązanie zadania za pomocą rekordów CNAME, A lub AAAA nie jest najlepsze,

ponieważ w rekordach tych nie ma informacji o rodzaju protokołu transportowego lub o numerze portu, którego trzeba użyć. Rekordy SRV zapewniają większą swobodę, ale ich elementy docelowe muszą odpowiadać nazwom domenowym powiązanym bezpośrednio z rekordami A lub AAAA. Zastosowanie rekordów NAPTR gwarantuje największą elastyczność dzięki dodatkowej warstwie pośredniczącej w odwzorowaniu oraz możliwości wskazywania rekordów innych typów (np. rekordów SRV).

Struktura rekordów NAPTR oraz towarzyszące im mechanizmy przepisywania tekstu wydawały się wielu projektantom i administratorom dość złożone, szczególnie z powodu użycia wyrażeń regularnych. Aby uprościć nieco definicję rekordów, a jednocześnie zapewnić większą elastyczność mechanizmu lokalizowania usług niż oferują rekordy SRV, opracowano **uproszczone rekordy NAPTR** (S-NAPTR — *Straightforward NAPTR*) [RFC3958] oraz aplikację DDDS odpowiadającą za odwzorowywanie etykiet z nazwami usług.

W rozwiązaniu S-NAPTR ciąg AUS jest etykietą domeny, w której poszukiwany jest serwer wskazanej usługi. Pierwsza znana reguła wskazuje funkcję identyfikacji. Wynikiem działania tej funkcji jest informacja (np. adres stacji, protokół, numer portu) niezbędna do odwołania się do konkretnej aplikacji działającej w domenie. Stosowane są jedynie dwie opcje ostateczne: `S` i `A`. Pierwsza z nich wskazuje rekord SRC, a druga nazwę domenową (wykorzystywaną zazwyczaj w zapytaniu o rekord A lub AAAA). Parametry pochodzą ze zbioru, którym zarządza organizacja IANA [SNP]. Pole wyrażenia regularnego nie jest używane. Dostępna jest jedynie wartość *Ciągu zastępczego*. Rekordy S-NAPTR są wykorzystywane w połączeniu z **usługą internetowych informacji rejestracyjnych** (IRIS — *Internet Registry Information Service*) [RFC3981], która wykorzystuje bazujący na standardzie XML protokół tekstowy odpowiedzialny za wymianę informacji na temat nazw domenowych oraz innych parametrów rejestracyjnych. Baza danych usługi jest utrzymywana w przestrzeni DNS w gałęzi `iris.arpa`. Oto przykład rekordu:

```
Linux% host -t naptr areg.iris.arpa
reg.iris.arpa NAPTR
100 10 "" "AREG1:iris.xpc:iris.lwz" "" areg.nro.net.
```

Widoczny na listingu rekord S-NAPTR (bez wyrażenia regularnego) informuje, że w celu wykonania zapytania IRIS o dane typu `AREG1` (patrz [RFC4698]) konieczne jest pobranie kolejnego rekordu NAPTR z domeny `areg.nro.net`.

Dodatkowe prace nad rozwiązaniem S-NAPTR doprowadziły od powstania rekordów NAPTR obsługujących ciągi URI (U-NAPTR) [RFC4848], w których zniesiono kilka ograniczeń właściwych dla rekordów S-NAPTR z zachowaniem wszystkich wcześniejszych funkcji. Dodatkowo wprowadzono opcję `U`, która umożliwia zdefiniowanie jako elementu docelowego adresu URI i skorzystanie z wyrażenia regularnego. Jest ona zatem podobna do standardowego rekordu NAPTR, ale z zastrzeżeniem, że wyrażenie regularne może mieć jedynie format `!.*!<URI>!`. Oznacza to, że cały ciąg AUS jest zastępowany wartością URI. Rekordy U-NAPTR są używane wraz **protokołem tłumaczenia lokalizacji na usługę** (LoST — *Location-to-Service Translation*) [RFC5222], który wskazuje usługę na podstawie punktu przyłączenia do sieci lub lokalizacji geograficznej. Takie informacje są przydatne w systemach bezpieczeństwa publicznego, w jakich na podstawie położenia obiektu można ustalić, która jednostka ratunkowa powinna zostać wezwana.

11.5.7. Dynamiczne aktualizacje DNS

Strefy DNS mogą podlegać automatycznym aktualizacjom (w procedurze **DNS UPDATE**) dzięki protokołowi opisanemu w dokumencie [RFC2136]. Umożliwia on określenie **warunków** (*prerequisite*), które są weryfikowane po stronie serwera. Jeśli nie zostaną spełnione, aktualizacja nie dochodzi do skutku, czemu towarzyszy wygenerowanie komunikatu z informacją o błędzie.

Procedura DNS UPDATE bazuje na komunikatach **dynamicznej aktualizacji** (*dynamic update*) przesyłanych do autorytatywnego serwera DNS danej strefy. Struktura komunikatów nie odbiega od tradycyjnych datagramów DNS, z tą jednak różnicą, że pola nagłówka oraz poszczególne sekcja mają inne nazwy (patrz rysunek 11.3). Do zadań sekcji należy wskazywanie aktualizowanych stref, definiowanie warunków aktualizacji (wyznaczanie rekordów RR, które muszą być zdefiniowane, aby aktualizacja została przeprowadzona) oraz przekazywanie samych **uaktualnień**. Nagłówek komunikatu aktualizacyjnego ma format zbliżony do standardowego zapytania, ale pole *OpCode* ma wartość Update (5). Zawarte w nagłówku pola *ZOCOUNT*, *PRCOUNT*, *UPCOUNT* oraz *ADCOUNT* przechowują liczniki: aktualizowanych stref (wartość 1), warunków przeznaczonych do interpretacji, uaktualnień oraz rekordów z dodatkowymi informacjami. Dokument [RFC2136] zawiera również wykaz wartości *RCODE* (przenoszonych w odpowiedziach DNS), w których odzwierciedlane są problemy ze spełnianiem poszczególnych warunków (wartości od 6 do 10 w tabeli 11.2).

Sekcja strefy (patrz rysunek 11.7) przenosi dane na temat nazwy, typu i klasy strefy. Typ wartości ma kod 6, który wskazuje na obecność rekordu SOA identyfikującego strefę. Klasa ma wartość 1 (strefa internetowa). Wszystkie aktualizowane rekordy muszą należeć do tej samej strefy.

Sekcja warunków aktualizacji przechowuje nie mniej niż jeden wpis w formacie rekordu RR opisanym w punkcie 11.5.5. Zdefiniowano pięć typów warunków: *RRSet exists* (zbiór RRSet istnieje; wariant zależny od wartości i niezależny od wartości), *RRSet does not exist* (zbiór RRSet nie istnieje), *Name in use* (nazwa w użyciu) oraz *Name not in use* (nazwa nie jest w użyciu). Zbiór RRSet jest kolekcją rekordów RR należących do jednej strefy, mających wspólną nazwę, klasę i typ. Interpretacja warunku wymaga ustawienia klasy, typu oraz wartości *RDATA* rekordów RR zgodnie z tabelą 11.5.

Tabela 11.5. *Pola Klasa i Typ używane w sekcji warunków do określania typu warunku*

Typ warunku (semantyka)	Ustawienie klasy	Ustawienie typu	Wartość RDATA
RRSet exists (niezależny od wartości)	ANY (dowolne)	Takie samo jak typu strefy	Pusta
RRSet exist (zależny od wartości)	Takie samo jak klasa strefy	Sprawdzany typ	Sprawdzany zbiór RRSet
RRSet does not exist	NONE (żadne)	Sprawdzany typ	Pusta
Name in use	ANY (dowolne)	ANY (dowolne)	Pusta
Name not in use	NONE (żadne)	ANY (dowolne)	Pusta

Typ *RRSet exists* oznacza, że w strefie wskazanej w sekcji strefy musi istnieć przynajmniej jeden zbiór RRSet o nazwie i typie odpowiadającym rekordowi RR z sekcji warunków. W przypadku operacji zależnej od wartości warunek jest uznawany za spełniony tylko wtedy, gdy dopasowane rekordy przechowują równocześnie pasujące wartości *RDATA*. Typ *RRSet does not exist* oznacza, że żaden ze zbiorów RRSet strefy wskazanej w sekcji strefy nie może pasować do nazwy i typu rekordu RR podanych w sekcji warunków. Dwa ostatnie warunki (*Name in use* oraz *Name not in use*) odnoszą się tylko do nazwy. Typ wartości nie jest sprawdzany. Kody klas DNS `NONE` oraz `ANY` to odpowiednio 254 i 255. W sekcji aktualizacji (poprzedzającej sekcję warunków) przenoszone są rekordy RR przeznaczone do dodania do strefy lub usunięcia z niej. W standardzie zdefiniowane zostały cztery rodzaje operacji aktualizacji, które są kodowane w rekordach za pomocą wartości pól *Klasa*, *Typ* oraz *RDATA*. Ich kombinacje zostały zestawione w tabeli 11.6.

Tabela 11.6. *Pola Klasa, Typ i RDATA rekordów, wykorzystywane w sekcji aktualizacji do określenia rodzaju operacji*

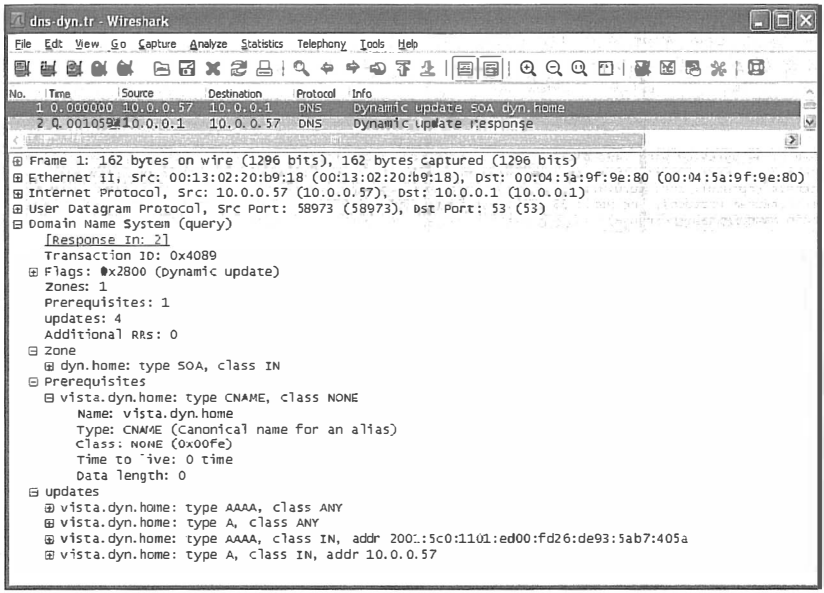
Operacja	Ustawienie klasy	Ustawienie typu	Wartość RDATA
Dodanie rekordu do zbioru RRSet	Takie samo jak klasa strefy	Typ dodawanego rekordu	Wartość *RDATA* dodawanego rekordu
Usunięcie zbioru RRSet	`ANY` (dowolne)	Typ usuwanego zbioru RRSet	Pusta (*TTL* i *RDLENGTH* również o wartości 0)
Usunięcie wszystkich zbiorów RRSet nazwy	`ANY` (dowolne)	`ANY` (dowolne)	Pusta (*TTL* i *RDLENGTH* również o wartości 0)
Usunięcie rekordu ze zbioru RRSet	`NONE` (żadne)	Typ usuwanego rekordu	Pasujące dane do usunięcia

Sekcja aktualizacji zawiera kolekcje rekordów RR, które są przetwarzane, jeśli nie wystąpiły żadne błędy związane z interpretacją warunków ani błędy serwera. W każdym rekordzie RR zapisana jest operacja dodania lub usunięcia danych. Modyfikowanie informacji polega na usunięciu i dodaniu nowych. Gdy chcemy przeanalizować przebieg operacji DNS UPDATE, wystarczy wykonać w systemie Windows następującą instrukcję:

```
C:\> ipconfig /registerdns
```

Systemy Windows domyślnie wysyłają żądanie aktualizacji ich nazwy oraz domeny. W zadaniu tym można uwzględnić również sufiks DNS. Należy w tym celu zaznaczyć opcję *Użyj sufiksu DNS tego połączenia do rejestracji w DNS*, która jest dostępna na zakładce *DNS* okna *Zaawansowane ustawienia TCP/IP* otwieranego po kliknięciu przycisku *Zaawansowane* w oknie *Protokół Internetowy w wersji 4 (TCP/IPv4)*. Analogiczna procedura włączenia opcji obowiązuje w ustawieniach protokołu IPv6. W przykładzie pokazanym na rysunku 11.16 można przeanalizować komunikat aktualizacji serwera DNS wygenerowany przez komputer o nazwie `vista` podczas uzupełniania lokalnej strefy `dyn.home`.

Na rysunku 11.6 widać również sposób kodowania aktualizacji. Serwer DNS o `adresie 10.0.0.1` (pracujący pod kontrolą oprogramowania BIND9 [AL06]) został skonfigurowany w taki sposób, aby umożliwiał dostarczanie dynamicznych aktualizacji. Strefa sekcji przechowuje rekord SOA, który wyznacza uaktualnianą strefę (`vista.dyn.home`).



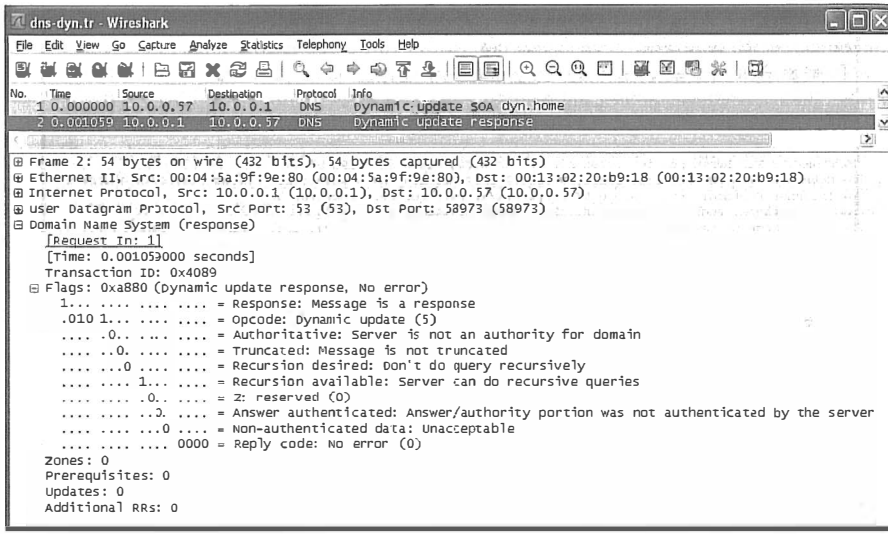
Rysunek 11.16. *Dynamiczna aktualizacja DNS wymaga zapisania rekordu SOA w sekcji strefy oraz rekordów RR w sekcji aktualizacji. W prezentowanym przykładzie rekordy te odpowiadają nowym adresom IPv4 i IPv6 stacji vista.dyn.home*

Sekcja warunków obejmuje rekord RR o zerowej długości pola RDATA oraz zerowej wartości parametru TTL. Rekord ten odpowiada więc trzeciemu wierszowi tabeli 11.5 (*RRSet does not exists*), ponieważ jego typ nie jest równy wartości ANY (na wartość CNAME), a klasa nie odpowiada wartości NONE (254).

Celem analizowanego komunikatu jest zarejestrowanie adresów 10.0.0.57 oraz 2001:5c0:1101:ed00:fd26:de93:5ab7:405a pod nazwą vista.dyn.home. Wykonanie zadania sprowadza się do usunięcia zbiorów RRSet typu AAAA i A (zgodnie z drugim wierszem tabeli 11.6) oraz dodania zbiorów RRSet typu AAAA i A (zgodnie z pierwszym wierszem tabeli 11.6) odpowiadających podanym adresom.

Odpowiedź serwera okazuje się bardzo zwarta i niezbyt złożona — odpowiedź na żądanie przedstawione na rysunku 11.16 pokazano na rysunku 11.7.

Pole znaczników (*Flags*) informuje o udanej operacji (brak błędów). Identyfikator transakcji o wartości (0x4089) gwarantuje powiązanie odpowiedzi z wcześniejszym żądaniem. Za uaktualnianie serwerów DNS w systemie Linux odpowiedzialny jest program nsupdate. Sam serwer DNS umożliwia aktualizowanie stref tylko wtedy, kiedy pozwalają na to procedury uwierzytelniające oraz procedury kontroli dostępu. Nie trzeba ich używać. Można je również sprowadzić do określenia listy adresów IP jednostek klienckich, ale nie jest to bezpieczne rozwiązanie. Warto skorzystać z nieco bardziej wyrafinowanych i bezpieczniejszych metod, które **uwierzytelniają transakcje** (takich jak opisane w rozdziale 18. rozwiązania TSIG i SIG(0)).



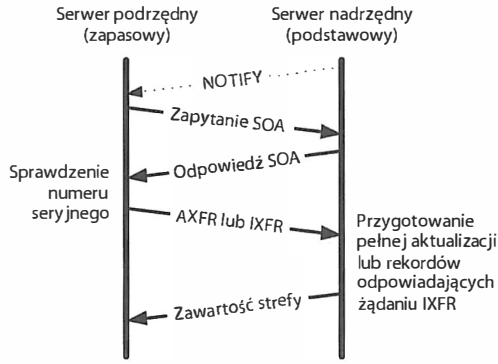
Rysunek 11.17. *Odpowiedź na żądanie dynamicznej aktualizacji składa się z identyfikatora transakcji i znaczników statusowych*

11.5.8. Transfer strefy i operacja DNS NOTIFY

Transfer strefy polega na kopiowaniu zbioru rekordów RR z jednego serwera do drugiego (zazwyczaj z serwera podstawowego do serwerów zapasowych). Jest wykonywany w celu zapewniania synchronizacji wielu serwerów strefy. Zwiększenie liczby serwerów zapewnia odporność na błędy wynikające z awarii jednego z systemów. Ponadto pozwala na zwiększenie wydajności odwzorowań, ponieważ ruch związany z zapytaniami jest rozkładany na różne serwery. Możliwe jest także zmniejszenie opóźnień w odwzorowaniach (opóźnień związanych z transmisją danych przez sieć), ale wymaga to uruchomienia serwerów w pobliżu jednostek klienckich.

Zgodnie z pierwotną specyfikacją transfer strefy następuje po **odpytaniu** (*polling*) serwera podstawowego przez serwer zapasowy. Serwery zapasowe okresowo kontaktują się z podstawowymi i sprawdzają, czy transfer jest konieczny. Porównują w tym celu numery seryjne konfiguracji strefy. Później wprowadzone rozwiązania uwzględniają również inicjowanie transferu strefy po wprowadzeniu zmian w jej treści. Służy do tego mechanizm asynchronicznej aktualizacji o nazwie **DNS NOTIFY**. Po zainicjowaniu operacji wykonywany jest pełny transfer strefy (z użyciem komunikatów AXFR) [RFC5936] lub **przyrostowy transfer strefy** (z użyciem komunikatów IXFR) [RFC1995]. Ogólna zasada działania mechanizmu została przedstawiona na rysunku 11.18.

W kolejnych podpunktach rozdziału znajduje się szczegółowe omówienie każdej operacji, włącznie z pełnym oraz częściowym transferem strefy i mechanizmem DNS Notify.



Rysunek 11.18. *Transfer strefy DNS polega na kopiowaniu zawartości stref między serwerami. Opcjonalne powiadomienie umożliwia serwerowi podrzędnemu wymuszenie pełnej lub przyrostowej aktualizacji*

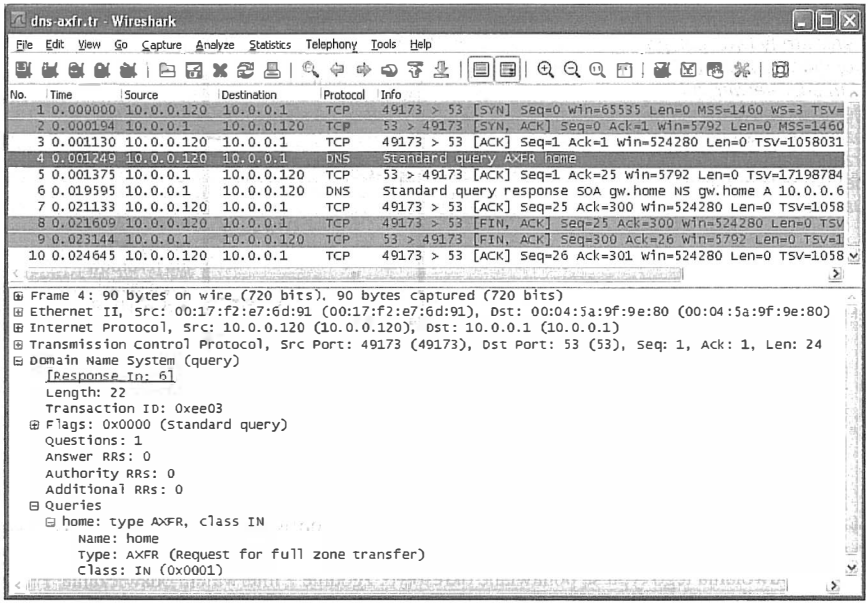
11.5.8.1. Pełny transfer strefy (komunikaty AXFR)

Operacje transferu strefy są regulowane przez parametry przekazywane w rekordzie SOA tej strefy, takie jak nazwa podstawowego serwera, numer seryjny, interwały odświeżania, ponawiania i wygasania danych. Po skonfigurowaniu serwer zapasowy próbuje skontaktować się z podstawowym, aby sprawdzić, czy transfer strefy jest konieczny. Odwołania tego typu są ponawiane okresowo, zgodnie z interwałem odświeżania. Pierwsza próba jest wykonywana zaraz po uruchomieniu serwera. Jeśli próba nawiązania komunikacji się nie powiedzie (z powodu braku odpowiedzi serwera), serwer zapasowy ponawia próby zgodnie z interwałem ponawiania (który z założenia jest krótszy niż interwał odświeżania). Jeśli komunikacja nie dojdzie do skutku przed upływem czasu wygasania, wszystkie dane strefy są usuwane z serwera zapasowego, co oznacza, że serwer przestaje udostępniać informacje na temat strefy.

Komunikat odpowiedzialny za pełny transfer strefy (standardowe zapytanie z rekordem typu AXFR w sekcji zapytania) jest przekazywany za pomocą protokołu TCP. Aby wygenerować stosowne żądanie, wystarczy wykonać polecenie `host`. Oto przykład jego użycia w sieci lokalnej:

```
Linux% host -l home.
Using domain server:
Name: 10.0.0.1
Address: 10.0.0.1#53
Aliases:

home name server gw.home.
ap.home has address 10.0.0.6
gw.home has address 10.0.0.1
...
```

Opcja `-l` stanowi dla programu `host` rozkaz wykonania pełnego transferu strefy z lokalnego serwera DNS. Program inicjuje wówczas dialog żądanie-odpowiedź w protokole TCP, co zostało pokazane na rysunku 11.19.

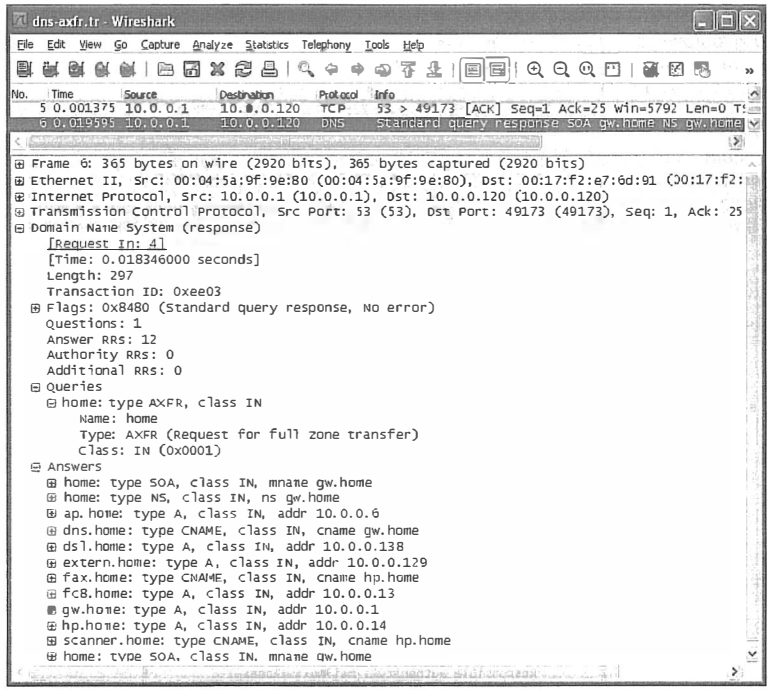


Rysunek 11.19. *Żądanie pełnego transferu strefy — przesłanie rekordu AXFR w protokole TCP*

Z rysunku 11.19 wynika, że operacja jest realizowana z użyciem protokołu TCP. Trzy pierwsze segmenty są standardowymi komunikatami uzgadniania połączenia TCP (więcej informacji na ten temat znajduje się w rozdziale 13.). Czwarty (przedstawiony szczegółowo) pakiet niesie zasadnicze żądanie. Jest to standardowe zapytanie DNS o typie AXFR i klasie IN (internetowej) kierowane do domeny o nazwie `home`. Odpowiedź na żądanie została przesłana w komunikacie o numerze 6, po segmencie `TCP ACK` (patrz rysunek 11.20).

Na rysunku 11.20 pokazano sposób przekazywania wszystkich danych strefy w komunikacie odpowiedzi. Po odebraniu odpowiedzi klient potwierdza dostarczenie danych TCP i rozpoczyna zamykanie połączenia. Rozłączenie następuje po wymianie komunikatów `FIN-ACK` (pakiety od 8. do 10.). Więcej informacji na temat ustanawiania i przerywania połączeń TCP znajduje się w rozdziale 13.

Choć w przeszłości transfer strefy można było zainicjować na niemal każdym serwerze DNS, obecnie operacja ta jest zarezerwowana jedynie dla autorytatywnych serwerów strefy (tj. serwerów wymienionych w rekordach NS strefy). Ograniczenia wynikają z konieczności zagwarantowania poufności danych i bezpieczeństwa systemu — informacje o komputerach pracujących w strefie mogłyby ułatwić przeprowadzenie ataku na określoną jednostkę lub usługę.



Rysunek 11.20. *Prawidłowa odpowiedź na żądanie pełnego transferu strefy zawiera wszystkie rekordy strefy. Transakcja jest realizowana z użyciem protokołu TCP z uwagi na ilość danych i konieczność zagwarantowania niezawodności komunikacji*

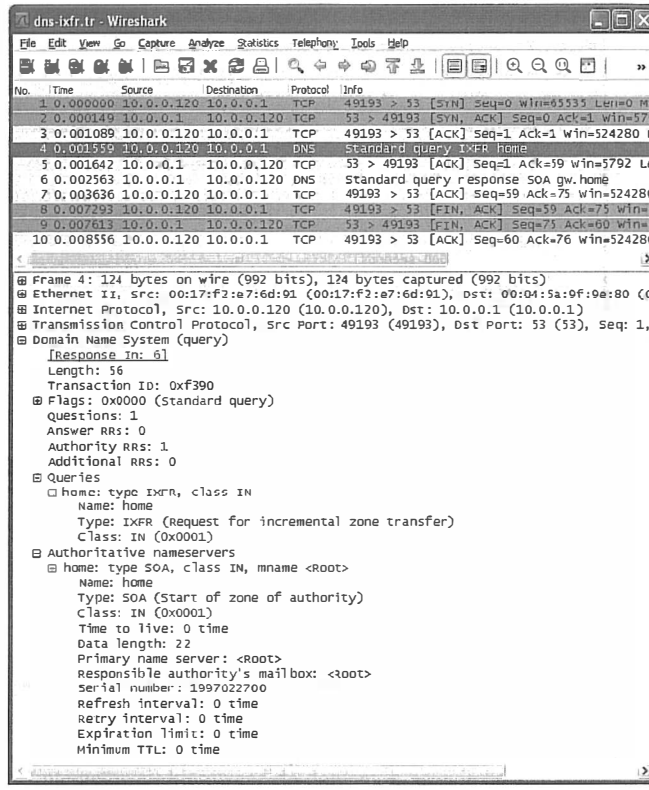
11.5.8.2. Przyrostowy transfer strefy (komunikaty IXFR)

Aby zwiększyć wydajność transferów stref, w dokumencie [RFC1995] opisano technikę **przyrostowego** transferu strefy. W operacji tego typu wykorzystywane są komunikaty IXFR, które odpowiadają za przekazywanie jedynie informacji o zmianach w strefie. Aby zainicjować procedurę, klient (serwer podrzędny) musi dostarczyć do serwera nadrzędnego bieżący numer seryjny strefy. W kolejnym przykładzie została zaprezentowana operacja wymuszenia transferu po dostarczeniu pewnego numeru seryjnego konfiguracji. W zadaniu wykorzystany został program `dig`.

```
Linux% dig +short @10.0.0.1 -t ixfr=1997022700 home.
gw.home. hostmaster.gw.home. 1997022700 10800 15 604800 10800
```

Z treści polecenia wynika, że końcowe zestawienie powinno mieć skróconą formę. Adres `10.0.0.1` jest adresem wykorzystywanego serwera DNS, a numer seryjny niezbędny do zainicjowania przyrostowego transferu strefy ma wartość `1997022700`. Wykonanie instrukcji wymusza wymianę informacji zbliżoną do przedstawionych na rysunkach 11.19 i 11.20. Jedyną różnicą jest to, że w tym przypadku numer seryjny żądania odpowiada bieżącemu numerowi seryjnemu (patrz rysunek 11.21).

Rysunek 11.21. *Żądanie przyrostowego transferu strefy (rekord typu IXFR) przenoszone w protokole TCP. Numer seryjny umożliwia ustalenie, które rekordy zostały zmienione od ostatniego transferu strefy*

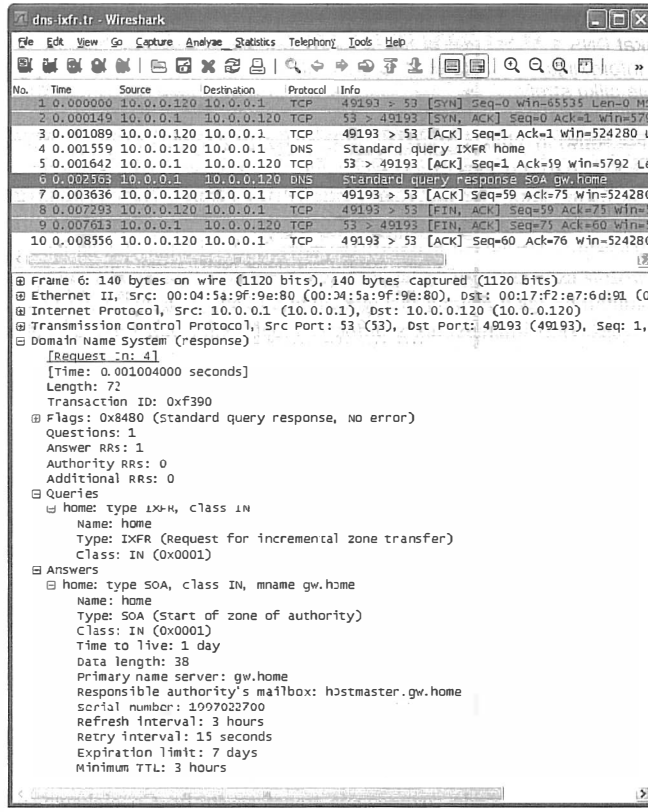


Ze zrzutu przedstawionego na rysunku 11.22 wynika, że żądanie IXFR przekazuje w sekcji pełnomocnictw niemal pusty rekord SOA. Jest w nim jednak zdefiniowany numer seryjny (1997022700). Odpowiedź (pakiet 6.) nie niesie żadnych użytecznych informacji, ponieważ numer seryjny żądania pokrywa się z numerem obowiązującym po stronie serwera.

Przedstawiona na rysunku 11.22 odpowiedź przenosi jedynie rekord SOA (w sekcji odpowiedzi). W przeciwieństwie do analogicznego rekordu zapisanego w sekcji zapytania, w tym wszystkie pola są wypełnione (np. adres e-mail oraz parametry transferu). Brak dodatkowych rekordów w odpowiedzi wynika z tego, że bieżący numer strefy odpowiada numerowi zawartemu w żądaniu. Serwer zakłada więc, że klient dysponuje aktualnymi informacjami i nie ma potrzeby dołączania dodatkowych danych lub przekazywania wszystkich informacji o strefie.

11.5.8.3. Mechanizm DNS NOTIFY

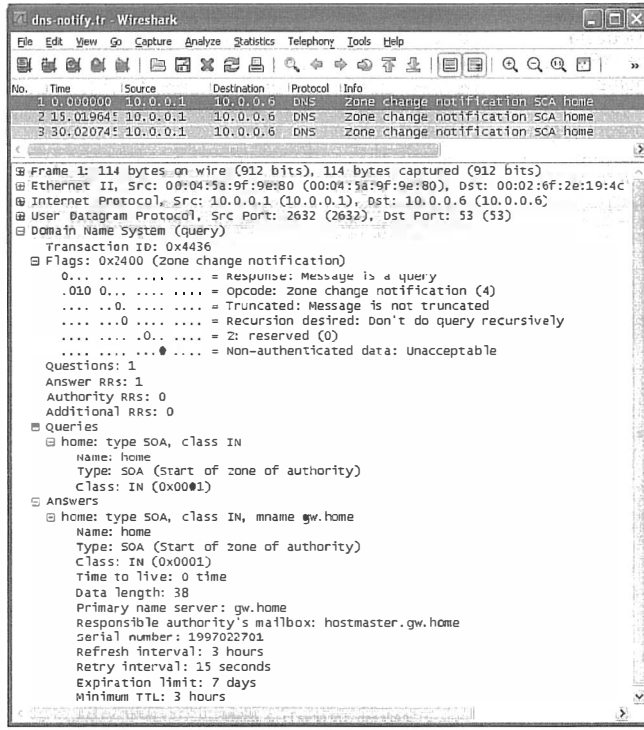
Zgodnie z zamieszczonymi wcześniej informacjami sprawdzenie, czy potrzebny jest transfer strefy, polega na odpytaniu serwera. Serwery podrzędne odwołują się do serwerów nadrzędnych w określonych interwałach (zgodnych z czasem odświeżania) i porównują



Rysunek 11.22. *Odpowiedź na żądanie IXFR. Jeśli numer seryjny jest zgodny, przenoszony jest jedynie rekord SOA bez dodatkowych informacji*

numery seryjne konfiguracji strefy. Jeśli są różne, rozpoczyna się transfer strefy. Rozwiązanie to wiąże się — niestety — z marnowaniem przepustowości łączy, ponieważ wymaga wykonania wielu niepotrzebnych zapytań. Aby udoskonalić system aktualizacji, w dokumencie [RFC1996] zdefiniowano mechanizm DNS NOTIFY. Umożliwia on serwerowi nadrzędnemu powiadomienie serwerów podrzędnych, że nastąpiła zmiana w pliku strefy i należy rozpocząć operację transferu strefy. Od strony technicznej oznacza to obowiązek wysłania komunikatu powiadomienia do zbioru serwerów za każdym razem, gdy zmieni się rekord SOA danej strefy (tzn. gdy zostanie zwiększony numer seryjny konfiguracji). Dzięki temu można inicjować transfer zawsze wtedy, kiedy trzeba. Na rysunku 11.23 został pokazany przebieg tej operacji w przypadku odwołania do lokalnego serwera nazw.

Przykład ten ilustruje sposób dostarczania komunikatu DNS NOTIFY do serwerów uwzględnionych w **zbiorze powiadamianych serwerów**. Komunikat informuje o tym, że w konfiguracji strefy zaszły zmiany. Powiadomienie jest dostarczane w formie zapytania DNS bazującego na protokołach UDP/IPv4 z odpowiednio ustawionymi polem *Opcji*. Sekcja zapytania zawiera identyfikatory typu i klasy rekordu SOA, a sekcja odpowiedzi przechowuje bieżącą wersję rekordu SOA danej strefy (z parametrem TTL



Rysunek 11.23. *Komunikat DNS NOTIFY informujący o zmianie pliku strefy. Transmisji zasadniczej towarzyszą dwie retransmisje, wykonywane co 15 sekund (niezgodnie ze standardem)*

równym 0) wraz z jego numerem seryjnym. Informacje te są dla powiadamianego serwera wystarczające do rozpoczęcia transferu strefy. Co ciekawe, pojedynczy serwer może otrzymać wiele powiadomień od różnych serwerów, w czasie gdy będą one uaktualniały swoje dane. Nie stanowi to jednak żadnej przeszkody podczas wykonania zadania.

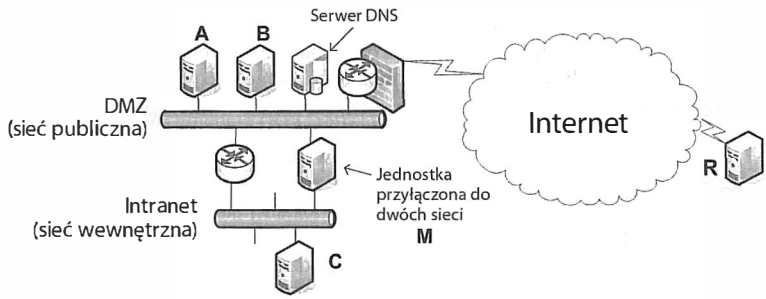
Domyślnie mechanizm DNS NOTIFY wykorzystuje protokół DNS, czyli protokół niegwarantujący dostarczenia informacji. W prezentowanym przykładzie odwołania były generowane do jednostki o adresie 10.0.0.11, która nie była serwerem DNS. Z tego powodu komunikat był powielany co 15 sekund, ale żadna odpowiedź nie nadeszła w ustalonym czasie.

Czas między retransmisjami oraz liczba retransmisji są zdefiniowanie w zaleceniu [RFC1996] i wynoszą odpowiednio 60 sekund i 5 retransmisji. Zaleca się również stosowanie adaptacyjnego lub wykładniczego algorytmu odmierzania czasu przerwy. Jak nietrudno zauważyć, oprogramowanie BIND9 działa niezgodnie z zaleceniami, ponieważ komunikat został ponowiony dwukrotnie w odstępie 15 sekund.

Odpowiedzi są klasycznymi komunikatami odpowiedzi DNS, pozbawionymi jednak użytecznych danych. Jedyną istotną informacją jest identyfikator transakcji. Odsyłanie odpowiedzi ma na celu spełnienie założeń protokołu i przerwanie procesu retransmisji po stronie serwera.

11.6. Listy sortowania, algorytm karuzelowy i dzielony DNS

W dotychczasowych rozważaniach przedstawione zostały zasady definiowania nazw domenowych, typy rekordów zasobów systemu DNS oraz sam protokół DNS służący do pobierania i aktualizowania danych o strefie. Kiedy chcemy mieć pełen obraz rozwiązania, warto przeanalizować również dane zwracane przez serwery oraz ich kolejność w odpowiedzi na zapytanie DNS. Serwery DNS mogą przekazywać wszystkie dane pasujące do zapytania klienckiego w takiej kolejności, jaką serwer uzna za stosowną. Jednak większość serwerów pozwala na zdefiniowanie specjalnych opcji konfiguracyjnych, które zapewniają generowanie odpowiedzi zgodnie z założeniami dotyczącymi funkcjonalności, poufności oraz wydajności transferu. Przeanalizujmy topologię przedstawioną na rysunku 11.24.



Rysunek 11.24. *W niewielkiej sieci serwery DNS mogą być skonfigurowane w taki sposób, aby zwracały różne adresy w zależności od adresu IP klienta*

Na rysunku 11.24 przedstawiona została typowa topologia sieci małego przedsiębiorstwa. Jest w niej sieć prywatna oraz strefa publiczna z serwerem DNS. W strefie DMZ pracują dwie stacje (A i B), a w sieci wewnętrznej jedna (C). W analizie zostanie uwzględniony również jeden z komputerów działający w Internecie. Pomiędzy segmentem DMZ i siecią wewnętrzną pracuje stacja (M) o dwóch adresach IP odpowiadających dwóm sieciom.

Jednostka, która chce ustanowić połączenie ze stacją M, musi najpierw odwołać się do serwera DNS, z którego otrzyma dwa adresy — jeden właściwy dla sieci wewnętrznej i jeden obowiązujący w strefie DMZ. Oczywiście, z perspektywy komputerów A, B i R poprawnym adresem jednostki M jest ten, który należy do sieci DMZ. Natomiast dla stacji C efektywniejsza jest bezpośrednia komunikacja przez sieć wewnętrzną. Takie rozróżnienie można wprowadzić, jeśli serwer DNS obsługuje sortowanie adresów w sposób zależny od źródłowego adresu IP żądania (choć mógłby również wykorzystać adres docelowy, szczególnie jeśli komputer M korzysta z dwóch podsieci w ramach tego samego interfejsu sieciowego). Jeśli jednostka generująca zapytanie używa adresu IP o takim samym prefiksie jak adresy w zwracanych rekordach, serwer DNS przenosi takie rekordy na początek wysyłanego komunikatu. Dzięki temu klient może użyć tego adresu IP serwera docelowego, który jest „najbliżej" niego — większość aplikacji próbuje nawiązać połączenie z wykorzystaniem pierwszego adresu ze zbioru zwróconych rekordów. Sposób działania oprogramowania w tym zakresie można zazwyczaj regulować za pomocą dy-

rektyw `sortlist` lub `rrset-order` (opcje te są wykorzystywane w konfiguracji resolverów i serwerów). Sortowanie może być również wykonywane przez serwer automatycznie, o ile takie rozwiązanie zostało zaimplementowane w danym oprogramowaniu.

Podobna zależność występuje również wtedy, gdy jedna usługa jest oferowana przez więcej niż jeden serwer. Przychodzące połączenia są wówczas kierowane tak, aby rozłożyć obciążenie serwerów (podzielić ruch między serwery). Załóżmy, że w sieci z poprzedniego przykładu w systemach A i B została uruchomiona taka właśnie usługa. Do jej identyfikacji mógłby wówczas służyć adres URL `http://www.przyklad.pl`. Klient usługi wysłałby w takim przypadku zapytanie DNS o nazwę domenową `www.przyklad.pl`, a serwer odesłałby odpowiedź składającą się ze zbioru rekordów adresu. Aby zapewnić rozkładanie obciążenia, serwer DNS może w takich sytuacjach stosować **algorytm karuzelowy** (*round robin*), który zapewnia cykliczną zmianę pozycji rekordów na liście. Dzięki niemu każdy nowy klient może zostać skierowany do innego serwera niż poprzedni klient. Rozwiązanie nie jest idealne, choć ułatwia rozkładanie obciążenia. Uzyskanie pożądanego efektu zakłóca mechanizm buforowania rekordów. Poza tym ruch jest rozkładany na pewną liczbę połączeń i nie zależy od bieżącego obciążenia serwerów. Zazwyczaj różne połączenia wymagają odmiennej mocy obliczeniowej serwera. Uzyskanie równomiernego obciążenia jest więc możliwe jedynie wtedy, gdy każde żądanie w takim samym stopniu zajmuje serwer.

Ostatnią rzeczą, na którą trzeba zwrócić uwagę podczas dostarczania danych, jest ich zabezpieczenie. W omawianym przykładzie zasadne mogłoby się okazać wprowadzenie podziału, zgodnie z którym komputery w sieci lokalnej otrzymywałyby informacje o wszystkich jednostkach pracujących w tej sieci. Natomiast komputer R miałby dostęp jedynie do wybranego zakresu danych. Taki sposób działania zapewnia rozwiązanie o nazwie **dzielony DNS** (*split DNS*). Jeśli jest ono zaimplementowane, zbiór zwracanych rekordów jest zależny od identyfikatora klienta lub docelowego adresu zapytania. Najczęściej oprogramowanie klienckie jest identyfikowane za pomocą adresu IP lub prefiksu adresu IP. Można więc skonfigurować serwery DNS w taki sposób, aby wszystkie komputery przedsiębiorstwa (współdzielące jeden prefiks) uzyskiwały pełne informacje z bazy danych DNS. Jednostki zdalne powinny mieć natomiast dostęp jedynie do danych systemów A i B, w których działa usługa WWW.

11.7. Otwarte serwery DNS i system DynDNS

Wielu właścicieli sieci domowych korzysta z pojedynczego adresu IPv4 przydzielanego przez lokalnego dostawcę usług internetowych. Udostępniany w ten sposób adres zmienia się wraz z każdym odłączeniem i ponownym przyłączeniem komputera lub routera. W rezultacie niemal niemożliwe okazuje się zdefiniowanie w systemie DNS wpisu, który pozwoli komputerom pracującym w Internecie na pozyskiwanie informacji o uruchomionej usłudze. Rozwiązaniem jest skorzystanie z oferty tzw. **dynamicznych serwerów DNS** (DDNS — *Dynamic DNS*), które obsługują specjalny protokół (**API aktualizacji DNS** [DYNDNS]) pozwalający na aktualizowanie wpisów w systemie DNS po wcześniejszym zarejestrowaniu się i założeniu konta. Technika ta **nie wykorzystuje** protokołu DNS UPDATE [RFC2136], lecz bazuje na niezależnym protokole warstwy aplikacji.

Aby skorzystać z usługi, trzeba uruchomić program klienta DDNS (np. inadyn, ddclient w systemie Linux lub DynDNS Updater w systemie Windows), który często jest również elementem routera domowego. W ustawieniach takiego programu zwykle można wpisać login i hasło zapewniające dostęp do zdalnej usługi DDNS. Program kliencki nawiązuje połączenie z serwerem, przesyła informacje o bieżącym globalnym adresie IP (adresie przydzielonym przez operatora ISP, który często jest wykorzystywany w usłudze NAT) i wstrzymuje swoje działanie na pewien czas. Operacja jest ponawiana okresowo, co pozwala serwerowi bezpiecznie usuwać dane, które nie zostały odświeżone w wyznaczonym czasie. Wspomniane usługi są dostępne w serwisach: *http://http://dyn.com/dns*, *http://freedns.afraid.org* oraz *http://www.no-ip.com/services/managed_dns/free_dynamic_dns.html*.

11.8. Przezroczystość i rozszerzalność

Usługa DNS to jedna z najpowszechniej wykorzystywanych usług internetowych i dlatego jest chętnie uzupełniana o nowe funkcje i rozszerzenia. Doskonałymi przykładami użyteczności mechanizmu są rekordy TXT, SRV, a nawet A (patrz [RFC5782]), które pozwalają na kodowanie danych w sposób umożliwiający wykorzystanie ich w wielu przyszłych usługach. W dokumencie [RFC5507] przedstawiono wiele metod rozszerzania systemu DNS wraz z konkluzją, że tworzenie i implementacja nowych typów RR wydaje się bardzo obiecującym rozwiązaniem. Dzięki wcześniejszej specyfikacji [RFC3597] standard przewiduje odpowiednią obsługę niezdefiniowanych rekordów RR (jako bloków danych). Zgodnie z nim nierozpoznawane rekordy nie są interpretowane, więc przetwarzanie jest dla nich **przezroczyste**. A zatem można wprowadzać nowe typy RR bez obaw o negatywny wpływ na przetwarzanie innych rekordów.

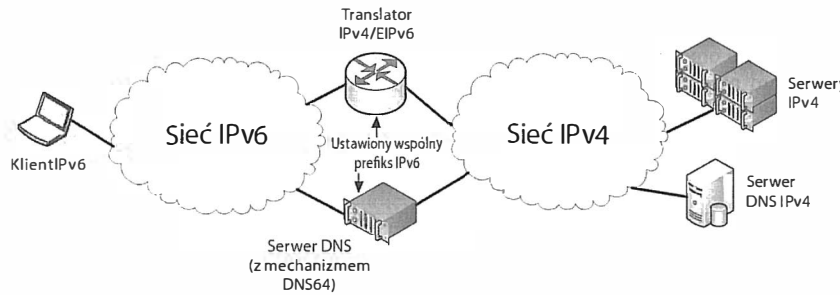
Jeden z problemów w zachowaniu przezroczystości wynika ze stosowanego kodowania nazw domenowych i kompresji. W nazwach domenowych zapisywanych w rekordach, które są obsługiwane przez system DNS, można modyfikować wielkość liter, aby zapewnić kompresję danych (z użyciem etykiet kompresji). Nazwy właściciela domeny (klucze zapytań) zawsze podlegają kompresji. Jednak nazwy domenowe zawarte w nieznanych rekordach nie mogą być poddawane kompresji. Z tego powodu niedozwolone jest stosowanie nowych rekordów RR z osadzonymi nazwami domenowymi (patrz sekcja 4. zalecenia [RFC3597]). Nie wyklucza to jednak porównywania bitowego rekordów (np. podczas dynamicznych aktualizacji). Wynika z tego, że wszystkie osadzone nazwy domenowe muszą być porównywane z uwzględnieniem wielkości liter [RFC4343], co jest sprzeczne ze sposobem działania większości funkcji DNS. Ta sama zasada dotyczy nazw domenowych definiowanych w rekordach TXT.

Wraz z opracowaniem nowych serwerów DNS i jednostek pośredniczących w transmisji ruchu uwidocznił się jeszcze jeden problem. Obecnie do domowych routerów i zapór sieciowych dość często włącza się moduły **proxy DNS**. Ich działanie polega na odbieraniu żądań z sieci wewnętrznej i przekazywaniu do serwerów nazw pracujących w sieci ISP. Urządzenia te odpowiadają również za dostarczanie odpowiedzi, które mogą (ale nie muszą) buforować. W przeszłości niektóre jednostki pośredniczące wykonywały nieco więcej zadań niż tylko przekazywanie żądań i odpowiedzi. Jednak wynikały z tego pewne problemy. Prawidłowe działanie urządzeń pośredniczących zostało opisane w dokumencie [RFC5625]. W zasadzie zalecenie sprowadza się do tego, żeby przekazywać

rekordy DNS bez interpretowania ich w trakcie transmisji. Jeśli konieczne jest obcięcie pakietu, moduł proxy musi ustawić bit TC, co wskazuje, że część danych została usunięta. Ponadto jednostki pośredniczące muszą być przygotowane do obsługi żądań TCP. Protokół ten jest standardowym mechanizmem zapasowym, używanym wówczas, gdy komunikaty dostarczane przez protokół UDP są obcinane. Poza tym zgodnie z dokumentem [RFC5966] jest obowiązkowym elementem systemu DNS.

11.9. Translacja komunikatów DNS IPv4 na IPv6 (DNS64)

W rozdziale 7. opisana została platforma translacji datagramów IP między sieciami IPv4 i IPv6. Analogiczne mechanizmy tłumaczące są proponowane do zapewnienia tłumaczenia między rekordami A i AAAA [RFC6147]. Umożliwiłoby to komputerom wyposażonym jedynie w interfejsy IPv6 korzystanie z informacji DNS zapisanych w rekordach A (np. w Internecie IPv4). Rozwiązanie to zostało nazwane DNS64, a jeden z proponowanych trybów jego działania został pokazany na rysunku 11.25 (mechanizm ten jest określany jako „DNS64 w trybie rekurencyjnego resolvera DNS").



Rysunek 11.25. *Usługa DNS64 przekształca rekordy A w AAAA i współdziała z translatorem IPv4-IPv6 w celu zapewnienia komunikacji między klientami IPv6 i usługami IPv4*

Jak widać na rysunku 11.25, usługa DNS64 współpracuje z translatorem IPv4-IPv6 (patrz rozdział 7.). Każde z urządzeń ma przypisany co najmniej jeden prefiks IPv6 potrzebny do osadzenia adresu IPv4. Każdy z prefiksów musi być prefiksem sieciowym (tj. udostępnionym przez operatora) lub „dobrze znanym prefiksem" (`64:ff9b::/96`). Urządzenie DNS64 pracuje jako serwer buforujący. Dla stacji IPv6 jest on podstawowym serwerem DNS. Obsługuje więc zapytania o rekordy AAAA powiązane z nazwami domenowymi. W rzeczywistości jednak przekształca odbierane żądania na zapytania o rekordy A oraz AAAA i wysyła do sieci IPv4. Jeśli żadna odpowiedź z rekordem AAAA nie zostanie odebrana, urządzenie DNS64 utworzy **syntetyczne** rekordy AAAA na podstawie skonfigurowanego prefiksu oraz zawartości każdego z otrzymanych rekordów A. Serwer odpowiada również na zapytania PTR o dowolny prefiks IPv6, który został użyty do utworzenia rekordów AAAA.

Synteza rekordów AAAA wymaga jedynie zmodyfikowania sekcji odpowiedzi w komunikacie DNS. Pozostałe sekcje pozostają niezmienione (ich zawartość odpowiada danym odebranym w sieci IPv4). Podczas przetwarzania łańcuchów CNAME lub

DNAME poszczególne elementy łańcucha są analizowane rekurencyjnie, aż do wyszukania rekordu A lub AAAA, a do odpowiedzi są dołączane wszystkie ogniwa łańcucha. Dzięki odpowiedniej konfiguracji serwer DNS64 może wykluczać z syntezowania adresy z określonych przestrzeni IPv6 i IPv4. W ten sposób zapobiega się niektórym anomaliom (np. osadzaniu adresów IPv4 o specjalnym przeznaczeniu). Działanie mechanizmu DNS64 oddziałuje w pewien sposób na pracę systemu DNSSEC, co zostało opisane w rozdziale 18.

11.10. Protokoły LLMNR i mDNS

Standardowy system DNS wymaga konfiguracji wielu serwerów DNS, które zapewnią odwzorowanie nazw domenowych na adresy oraz udostępnią inne potrzebne informacje. Często okazuje się to zbyt dużym obciążeniem dla użytkowników; dzieje się to szczególnie wtedy, kiedy w komunikacji uczestniczy zaledwie kilka komputerów. W sytuacjach, w których uruchamianie usług DNS wydaje się nieuzasadnione (np. w szybko formowanej sieci ad-hoc), pomocna bywa lokalna wersja systemu DNS bazująca na protokole LLMNR [RFC4795]. Jest to rozwiązanie niestandardowe, opracowane przez firmę Microsoft, ale bazujące na systemie DNS i znajdujące zastosowanie w sieciach lokalnych. Ułatwia ono wyszukiwanie takich urządzeń jak drukarki czy serwery plików. Jest obsługiwane przez systemy Windows Vista, Windows Server 2008, Windows 7 oraz Windows 8. Wykorzystuje protokół UDP i port 5355 oraz adres multiemisji IPv4 `224.0.0.252` bądź adres IPv6 `ff02::1:3`. Serwery dodatkowo używają portu 5355 w protokole TCP do wysyłania odpowiedzi z klasycznych adresów IP.

Inną formą lokalnego systemu DNS jest **multiemisyjny DNS** (mDNS — *multicast DNS*) [IDMDNS] opracowany przez firmę Apple. Wraz z protokołem wykrywania usług (DNS-SD — *DNS Service Discovery*) tworzy on platformę Bonjour. W rozwiązaniu mDNS komunikaty DNS są przekazywane za pomocą lokalnych adresów multiemisji. W działaniu systemu wykorzystywany jest port UDP 5353 oraz specjalna domena TLD `.local` (o niestandardowym przeznaczeniu). Domena `.local` ma zasięg lokalny. Wszystkie zapytania DNS o nazwy z tej domeny są kierowane na adres IPv4 `224.0.0.251` lub adres IPv6 `ff02::fb`. Zapytania odnoszące się do innych domen również mogą być wysyłane na podane adresy, ale jest to działanie opcjonalne. Umożliwienie lokalnym serwerom odpowiadania na prośby o odwzorowanie globalnych nazw może się okazać zagrożeniem dla sieci. Aby wyeliminować ten problem, warto rozważyć wdrożenie systemu DNSSEC (więcej informacji na jego temat znajduje się w rozdziale 18.). Mechanizm mDNS pozwala jednostkom na przypisywanie sobie nazw w pseudodomenie `.local`, mimo że domena ta nie została zarezerwowana na potrzeby protokołu mDNS [RFC2606]. Urządzenia pracujące w niewielkiej sieci domowej mogą więc być opisywane za pomocą takich nazw jak `drukarka.local`, `serwerplikow.local`, `camera1.local`, `laptopkrzysia.local` itp. Protokół mDNS uwzględnia mechanizmy wykrywania i eliminowania kolizji.

11.11. Usługa LDAP

Dotychczasowe rozważania odnosiły się do systemu DNS i lokalnych usług o charakterze zbliżonym do DNS. Obsługa bardziej złożonych zapytań i algorytmów przetwarzania danych wymaga jednak zastosowania ogólnej usługi katalogowej, takiej jak wspomniana

wcześniej usługa LDAP [RFC4510]. Standard LDAP (obecnie w wersji 3.) opisuje protokół warstwy aplikacji, zapewniający użytkownikom Internetu dostęp do katalogów danych i usług zgodnych z zaleceniem X.500 [X500]. Zapewnia funkcje wyszukiwania, modyfikowania, dodawania, porównywania oraz usuwania obiektów w zależności od wzorca wybranego przez użytkownika. Katalog LDAP ma formę drzewa obiektów, w którym każdemu obiektowi odpowiada pewien zbiór atrybutów. Wraz z upowszechnieniem się protokołów TCP/IP rozwiązanie LDAP zostało przystosowane do współdziałania z usługą DNS. Przykładowo za pomocą narzędzia `ldapsearch` (w systemach Windows działa podobne narzędzie o nazwie `ldp`, które trzeba pobrać ze strony narzędzi dodatkowych) można przesłać następujące żądanie dostarczenia informacji na temat biura kanclerza MIT:

```
Linux% ldapsearch -x -h ldap.mit.edu -b "dc=mit,dc=edu" "(ou=*Chancellor*)"
# extended LDIF
#
# LDAPv3
# base <dc=mit,dc=edu> with scope sub
# filter: (ou=*Chancellor*)
# requesting: ALL
#
.....
```

Polecenie nakazuje nawiązanie połączenia z serwerem `ldap.mit.edu` bez używania żadnego szczególnego protokołu uwierzytelniania (opcja `-x`). Szczegółowe omówienie mechanizmu LDAP wykracza poza ramy tematyczne rozdziału (i książki). Jednak z fragmentu uzyskanego listingu wynika sposób użycia elementów `dc` (*domain component* — element domeny) do powiązania danych LDAP z systemem DNS. Każdy komponent `dc` przechowuje jedną etykietę nazwy DNS, a za pomocą kilku takich elementów można zapisać całą nazwę domenową, która stanowi punkt odniesienia w zapytaniach LDAP. Budowanie zapytań LDAP nie należy do szczególnie skomplikowanych zadań. W prezentowanym przykładzie do tego celu wykorzystany został komponent jednostki organizacyjnej (`ou`) zawierający słowo `Chancellor` otoczone symbolami wieloznacznymi.

Najczęściej serwery LDAP są wykorzystywane w przedsiębiorstwach do przechowywania informacji o lokalizacji, numerach telefonicznych i zależnościach między jednostkami organizacyjnymi. Rozwiązania standardu LDAP zostały uwzględnione w systemie Active Directory firmy Microsoft i znajdują szerokie zastosowanie w zarządzaniu kontami użytkowników, usługami i prawami dostępu w dużych przedsiębiorstwach bazujących na systemach Windows. Niektóre serwery LDAP (jak ten, który pracuje w MIT i wielu innych uniwersytetach) są ogólnie dostępne w Internecie.

11.12. Ataki na usługi DNS

System DNS jest kluczowym elementem Internetu i w czasie ostatnich lat był wielokrotnie celem różnych ataków, a także przedmiotem prac nad rozwojem zabezpieczeń [RFC3833]. Stosunkowo niedawno, w ramach globalnego programu o nazwie DNS Security (DNSSEC), poczyniono znaczne postępy we wdrażaniu mechanizmów uwierzytelniania w operacjach DNS. Szczegółowe omówienie systemu DNSSEC znajduje się w rozdziale 18., gdyż zrozumienie zasad jego działania wymaga poznania podstawowych zagadnień z zakresu kryptografii. Celem bieżącego podrozdziału jest natomiast przedstawienie ataków wymierzonych przeciw systemowi DNS.

Usługi DNS są narażone na dwa rodzaje ataków. Pierwszy obejmuje techniki DoS, których celem jest przeciążenie najważniejszych serwerów (np. serwerów głównych lub obsługujących domeny TLD) i doprowadzenie ich do stanu, w którym przestaną odpowiadać na żądania klienckie. Drugi rodzaj ataków polega na zmianie treści rekordów zasobów lub przesłonięciu oficjalnego serwera DNS i generowaniu odpowiedzi z fałszywymi danymi. Rezultatem tego działania jest podesłanie stacjom klienckim niewłaściwego adresu IP, a tym samym wymuszenie połączenia z innym systemem (np. przejęcie komunikacji z serwisem bankowym).

Pierwszy istotny atak na system DNS odnotowano na początku 2001 roku. Polegał on na generowaniu wielu zapytań o rekordy MX serwisu *aol.com*. Osoba atakująca wykorzystywała do przesyłania żądań sfałszowane adresy IP. Zapytanie jest relatywnie małym pakietem, natomiast odpowiedź ma znacznie większy rozmiar (ok. dwudziestokrotnie). Był to więc atak ze **wzmocnieniem**, ponieważ szerokość pasma zajęta w wyniku ataku była znacznie większa niż potrzebna do jego przeprowadzenia. Odpowiedzi DNS są dostarczane pod adresy IP zapisane w pakietach żądań. Osoba atakująca mogła więc skierować ruch z serwerów nazw do dowolnego komputera. Całe zdarzenie zostało opisane szczegółowo w informacji przygotowanej przez organizację CERT [CIN].

Pierwszy atak uwzględniający modyfikację danych został zarejestrowany pod koniec 2008 roku [CKB] i nazwany **atakiem Kamińskiego**. Jego celem było **zatrucie pamięci podręcznej** — zbuforowane dane dostarczone przez serwer zostały zastąpione błędnymi lub sfałszowanymi informacjami, które następnie były przekazane do resolverów stacji końcowych. W jednym z wariantów rozwiązania osoba atakująca odpowiada na kierowane przez serwer buforujący zapytania o rekord A rekordami NS z nazwą domenową wybranej jednostki. Adres IP stacji (wskazany przez atakującego) jest również uwzględniany w sekcji informacji dodatkowych odpowiedzi DNS. Nazwa domenowa stacji może, ale nie musi, należeć do tej samej poddomeny, co pierwotne żądanie. Zagrożenie wynika z tego, że klient polegający na odwzorowaniu nazwy na adres może zostać przekierowany do fałszywych serwerów. Jeśli wspomniane serwery są celowo skonfigurowane w taki sposób, aby przypominały prawdziwe serwisy (np. strony bankowe), użytkownicy mogą nieświadomie ujawnić tajne informacje. Sposoby rozwiązywania takich i podobnych problemów zostały opisane w dokumencie [RFC5452]. Jedna z metod (nieuwzględniona w opracowaniu [RFC5452]), o nazwie **DNS-0x20** [D08], bazuje na wartościach jednorazowych, które są zapisywane na bitach znajdujących się na pozycji 0x20 każdego znaku, który wchodzi w skład nazwy zapisanej w sekcji zapytania. Bity te muszą zostać skopiowane do analogicznego obszaru odpowiedzi. Jest to możliwe, ponieważ — mimo że podczas porównywania nazw domenowych nie jest uwzględniana wielkość liter — serwery starają się zwrócić w sekcji odpowiedzi dokładną nazwę z zapytania. Jeśli do zapytania celowo zostanie włączona nazwa właściciela, odtworzenie wartości jednorazowej w odpowiedzi będzie bardzo utrudnione, a fałszerstwo łatwe do wykrycia.

11.13. Podsumowanie

System DNS jest kluczowym elementem Internetu, a stosowane w nim rozwiązania są wdrażane również w sieciach prywatnych. Przestrzeń nazw DNS ma zasięg ogólnoświatowy i została podzielona w sposób hierarchiczny z domenami najwyższego poziomu na szczycie drzewa. Nazwy domenowe mogą mieć charakter narodowy, jeśli do ich zapisu

wykorzystuje się technikę IDN. Aplikacje klienckie korzystają z resolverów, które odwołując się do przynajmniej jednego serwera DNS, przeszukują bazę danych strefy. Operacja ta skutkuje przekształceniem nazwy jednostki na odpowiadający jej adres IP lub odwrotnie. Resolvery kontaktują się z lokalnymi serwerami nazw, a te — działając rekurencyjnie — mogą odwołać się do jednego z serwerów głównych lub do innego serwera, który zrealizuje zadanie. Większość serwerów DNS i część resolverów buforuje pozyskane informacje, aby przyspieszyć odpowiadanie na podobne żądania innych klientów. Okres przechowywania informacji w buforze jest określony przez czas ważności danych (*TTL*). Zapytania i odpowiedzi są przesyłane za pomocą specjalnego protokołu DNS, który współdziała zarówno z protokołem TCP, jak i z protokołem UDP. Te z kolei są przenoszone przez protokoły IPv4 lub IPv6 bądź przez oba jednocześnie.

Wszystkie zapytania i odpowiedzi mają taki sam format obejmujący sekcje zapytań, odpowiedzi, informacji o pełnomocnictwach oraz informacji dodatkowych. Większość danych DNS jest przechowywana w rekordach zasobów. Istnieje wiele typów rekordów zasobów, w tym typy adresu, przekaźnika poczty i wskaźników na nazwy. Większość komunikatów DNS przekazywanych w Internecie bazuje na protokołach UDP/IPv4. Z tego powodu ich rozmiar jest ograniczony do 512 bajtów. Jednak zastosowanie specjalnego rozszerzenia (EDNS0) pozwala na wydłużenie komunikatów, co jest niezbędne w systemie DNSSEC, który został opisany w rozdziale 18.

Standard DNS definiuje pewne specjalne funkcje — np. transfer strefy oraz dynamiczne aktualizacje. Transfery stref (pełne lub przyrostowe) zapewniają synchronizację serwerów podrzędnych z serwerami nadrzędnymi, przechowującymi dane strefy. Rozwiązanie to pozwala na zwiększenie niezawodności systemu. Dynamiczne aktualizacje umożliwiają modyfikowanie wpisów strefy przez aplikacje korzystające ze specjalnego protokołu. W praktyce stosowane są dwie techniki realizacji tego zadania — formalna [RFC2136] i nieformalna. Pierwsza jest wykorzystywana w przedsiębiorstwach, natomiast druga zyskuje popularność wśród osób korzystających z tymczasowych adresów IP (np. w sieciach kablowych lub w połączeniach DSL). Pozwala ona na dynamiczne dodanie wpisu DNS, który umożliwia odszukanie określonej nazwy w ogólnoświatowej sieci.

System DNS jest celem wielu ataków — od ataków typu DoS (ograniczających dostępność serwerów w sieci) po zatruwanie pamięci podręcznej DNS (skutkujące skierowaniem ruchu do fałszywych serwerów). Na szczęście, opracowano kilka technik eliminowania tego rodzaju problemów, w tym techniki kryptograficzne (opisane w rozdziale 18.) oraz zmiany w konfiguracji serwerów DNS (zmniejszające ich podatność na fałszywe odpowiedzi).

11.14. Bibliografia

[AL06] P. Albitz, C. Liu, *DNS and BIND, Fifth Edition*, O'Reilly Media, Inc., 2006.

[CIN] *http://www.cert.org/incident_notes/IN-2000-04.html*

[CKB] *http://www.kb.cert.org/vuls/id/800113*

[D08] D. Dagon i in., „Increased DNS Forgery Resistance Through 0x20-bit Encoding", *Proc. ACM CCS*, październik 2008.

[DNSPARAM] *http://www.iana.org/assignments/dns-parameters*

[DR97] D. Dougherty, A. Robbins, *sed & awk, Second Edition*, O'Reilly Media, 1997.

[DYNDNS] *http://dyn.com/about*

[ENUM] *http://www.iana.org/assignments/enum-services*

[GTLD] *http://www.iana.org/domains/root/db*

[ICANN] *http://www.icann.org/en/tlds*

[IDDN] S. Rose, W. Wijngaards, *Update to DNAME Redirection in the DNS*, Internet draft-ietf-dnsext-rfc2672bis-dname, w opracowaniu, lipiec 2011.

[IDMDNS] S. Cheshire, M. Krochmal, *Multicast DNS*, Internet draftcheshire-dnsext-multicastdns, w opracowaniu, luty 2011.

[IIDN] *http://www.icann.org/en/topics/idn*

[ISO3166] International Organization for Standardization, *International Standard for Country Codes*, ISO 3166-1, 2006.

[ISPR] *http://www.iana.org/assignments/service-names-port-numbers*

[J02] J. Jung i in., *DNS Performance and the Effectiveness of Caching*, IEEE/ACM Transactions on Networking, 10(5), październik 2002.

[MD88] P. Mockapetris, K. Dunlap, „Development of the Domain Name System", *Proc. ACM SIGCOMM*, sierpień 1988.

[P10] N. Paskin, „Digital Object Identifier (DOI©) System", *Encyclopedia of Library and Information Sciences*, wydanie trzecie, Taylor and Francis, 2010.

[R06] H. Rice, *ENUM— The Mapping of Telephone Numbers to the Internet*, „The Telecommunications Review", 17, sierpień 2006.

[RFC1035] P. Mockapetris, *Domain Names — Implementation and Specification*, Internet RFC 1035/STD 0013, listopad 1987.

[RFC1464] R. Rosenbaum, *Using the Domain Name System to Store Arbitrary String Attributes*, Internet RFC 1464 (experimental), maj 1993.

[RFC1536] A. Kumar i in., *Common DNS Implementation Errors and Suggested Fixes*, Internet RFC 1536 (informational), październik 1993.

[RFC1912] D. Barr, *Common DNS Operational and Configuration Errors*, Internet RFC 1912 (informational), luty 1996.

[RFC1918] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, *Address Allocation for Private Internets*, RFC 1918/BCP 0005, luty 1996.

[RFC1995] M. Ohta, *Incremental Zone Transfer in DNS*, Internet RFC 1995, sierpień 1996.

[RFC1996] P. Vixie, *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)*, Internet RFC 1996, sierpień 1996.

[RFC2136] P. Vixie, (red.), S. Thomson, Y. Rekhter, J. Bound, *Dynamic Updates in the Domain Name System (DNS UPDATE)*, Internet RFC 2136, kwiecień 1997.

[RFC2141] R. Moats, *URN Syntax*, Internet RFC 2141, maj 1997.

[RFC2181] R. Elz, R. Bush, *Clarifications to the DNS Specification*, Internet RFC 2181, lipiec 1997.

[RFC2308] M. Andrews, *Negative Caching of DNS Queries (DNS NCACHE)*, Internet RFC 2308, luty 1998.

[RFC2317] H. Eidnes, G. de Groot, P. Vixie, *Classless IN-ADDR.ARPA Delegation*, Internet RFC 2317/BCP 0020, luty 1998.

[RFC2606] D. Eastlake 3rd, A. Panitz, *Reserved Top Level DNS Names*, Internet RFC 2606/BCP 0032, czerwiec 1999.

[RFC2671] P. Vixie, *Extension Mechanisms for DNS (EDNS0)*, Internet RFC 2671, sierpień 1999.

[RFC2672] M. Crawford, *Non-Terminal DNS Name Redirection*, Internet RFC 2672, sierpień 1999.

[RFC2782] A. Gulbrandsen, P. Vixie, L. Esibov, *A DNS RR for Specifying the Location of Services (DNS SRV)*, Internet RFC 2782, luty 2000.

[RFC2845] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington, *Secret Key Transaction Authentication for DNS (TSIG)*, Internet RFC 2845, maj 2000.

[RFC2930] D. Eastlake 3rd, *Secret Key Establishment for DNS (TKEY RR)*, Internet RFC 2930, wrzesień 2000.

[RFC3172] G. Huston, (red.), *Management Guidelines and Operational Requirements for the Address and Routing Parameter Area Domain (arpa)*, Internet RFC 3172/BCP 0052, wrzesień 2001.

[RFC3263] J. Rosenberg, H. Schulzrinne, *Session Initiation Protocol (SIP): Locating SIP Servers*, Internet RFC 3263, czerwiec 2002.

[RFC3401] M. Mealling, *Dynamic Delegation Discovery System (DDDS) — Part One: The Comprehensive DDDS*, Internet RFC 3401 (informational), październik 2002.

[RFC3402] M. Mealling, *Dynamic Delegation Discovery System (DDDS) — Part Two: The Algorithm*, Internet RFC 3402, październik 2002.

[RFC3403] M. Mealling, *Dynamic Delegation Discovery System (DDDS) — Part Three: The Domain Name System (DNS) Database*, Internet RFC 3403, październik 2002.

[RFC3404] M. Mealling, *Dynamic Delegation Discovery System (DDDS) — Part Four: The Uniform Resource Identifiers (URI) Resolution Application*, Internet RFC 3404, październik 2002.

[RFC3492] A. Costello, *Punycode: A Bootstring Encoding of Unicode for Internationalized Domain Names in Applications (IDNA)*, Internet RFC 3492, luty 2003.

[RFC3596] S. Thomson, C. Huitema, V. Ksinant, M. Souissi, *DNS Extensions to Support IP Version 6*, Internet RFC 3596, październik 2003.

[RFC3597] A. Gustafsson, *Handling of Unknown DNS Resource Record (RR) Types*, Internet RFC 3597, wrzesień 2003.

[RFC3833] D. Atkins, R. Austein, *Threat Analysis of the Domain Name System (DNS)*, Internet RFC 3833 (informational), sierpień 2004.

[RFC3958] L. Daigle, A. Newton, *Domain-Based Application Service Location Using SRV RRs and the Dynamic Delegation Discovery Service (DDDS)*, Internet RFC 3958, styczeń 2005.

[RFC3981] A. Newton, M. Sanz, *IRIS: The Internet Registry Information Service (IRIS) Core Protocol*, Internet RFC 3981, styczeń 2005.

[RFC3986] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, Internet RFC 3986/STD 0066, styczeń 2005.

[RFC4193] R. Hinden, B. Haberman, *Unique Local IPv6 Unicast Addresses*, Internet RFC 4193, październik 2005.

[RFC4343] D. Eastlake 3rd, *Domain Name System (DNS) Case Insensitivity Clarification*, Internet RFC 4343, styczeń 2006.

[RFC4406] J. Lyon, M. Wong, *Sender ID: Authenticating E-Mail*, Internet RFC 4406 (experimental), kwiecień 2006.

[RFC4592] E. Lewis, *The Role of Wildcards in the Domain Name System*, Internet RFC 4592, lipiec 2006.

[RFC4408] M. Wong, W. Schlitt, *Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1*, Internet RFC 4408 (experimental), kwiecień 2006.

[RFC4510] K. Zeilenga, (red.), *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*, Internet RFC 4510, Czerwiec 2006.

[RFC4690] J. Klensin, P. Falstrom, C. Karp, *Review and Recommendations for Internationalized Domain Names (IDNs)*, Internet RFC 4690 (informational), wrzesień 2006.

[RFC4698] E. Gunduz, A. Newton, S. Kerr, *IRIS: An Address Registry (areg) Type for the Internet Registry Information Service*, Internet RFC 4698, październik 2006.

[RFC4795] B. Aboba, D. Thaler, L. Esibov, *Link-Local Multicast Name Resolution (LLMNR)*, Internet RFC 4795 (informational), styczeń 2007.

[RFC4848] L. Daigle, *Domain-Based Application Service Location Using URIs and the Dynamic Delegation Discovery Service (DDDS)*, Internet RFC 4848, kwiecień 2007.

[RFC4960] R. Stewart, (red.), *Stream Control Transmission Protocol*, Internet RFC 4960, wrzesień 2007.

[RFC5001] R. Austein, *DNS Name Server Identifier (NSID) Option*, Internet RFC 5001, sierpień 2007.

[RFC5222] T. Hardie i in., *LoST: A Location-to-Service Translation Protocol*, Internet RFC 5222, sierpień 2008.

[RFC5321] J. Klensin, *Simple Mail Transfer Protocol*, Internet RFC 5321, październik 2008.

[RFC5452] A. Hubert, R. van Mook, *Measures for Making DNS More Resilient against Forged Answers*, Internet RFC 5452, styczeń 2009.

[RFC5483] L. Conroy, K. Fujiwara, *ENUM Implementation Issues and Experiences*, Internet RFC 5483 (informational), luty 2009.

[RFC5507] P. Falstrom, R. Austein, P. Koch, (red.), *Design Choices When Expanding the DNS*, Internet RFC 5507 (informational), kwiecień 2009.

[RFC5509] S. Loreto, *Internet Assigned Numbers Authority (IANA) Registration of Instant Messaging and Presence DNS SRV RRs for the Session Initiation Protocol (SIP)*, Internet RFC 5509, kwiecień 2009.

[RFC5625] R. Bellis, *DNS Proxy Implementation Guidelines*, Internet RFC 5625/BCP 0152, sierpień 2009.

[RFC5782] J. Levine, *DNS Blacklists and Whitelists*, Internet RFC 5782 (informational), luty 2010.

[RFC5855] J. Abley, T. Manderson, *Nameservers for IPv4 and IPv6 Reverse Zones*, Internet RFC 5855/BCP 0155, maj 2010.

[RFC5890] J. Klensin, *Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework*, Internet RFC 5890, sierpień 2010.

[RFC5891] J. Klensin, *Internationalized Domain Names in Applications (IDNA): Protocol*, Internet RFC 5891, sierpień 2010.

[RFC5936] E. Lewis, A. Hoenes, (red.), *DNS Zone Transfer Protocol (AXFR)*, Internet RFC 5936, czerwiec 2010.

[RFC5966] R. Bellis, *DNS Transport over TCP — Implementation Requirements*, Internet RFC 5966, sierpień 2010.

[RFC6116] S. Bradner, L. Conroy, K. Fujiwara, *The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)*, Internet RFC 6116, luty 2011.

[RFC6117] B. Hoeneisen, A. Majrhofer, J. Livingood, *IANA Registration of Enumservices: Guide, Template and IANA Considerations*, Internet RFC 6117, luty 2011.

[RFC6147] M. Bagnulo, A. Sullivan, P. Matthews, I. van Beijnum, *DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers*, Internet RFC 6147, kwiecień 2011.

[RFC6168] W. Hardaker, *Requirements for Management of Name Servers for the DNS*, Internet RFC 6168 (informational), maj 2011.

[RFC6186] C. Daboo, *Use of SRV Records for Locating Email Submission/Access Services*, Internet RFC 6186, luty 2011.

[RFC6195] D. Eastlake 3rd, *Domain Name System (DNS) IANA Considerations*, Internet RFC 6195/BCP 0042, luty 2011.

[RFC6303] M. Andrews, *Locally Served DNS Zones*, Internet RFC 6303/BCP 0163, lipiec 2011.

[ROOTS] *http://www.root-servers.org*

[RSYNC] *http://rsync.samba.org*

[SNP] *http://www.iana.org/assignments/s-naptr-parameters*

[U11] The Unicode Consortium, *The Unicode Standard*, wersja 6.0.0, The Unicode Consortium, 2011.

[URI] *http://www.iana.org/assignments/uri-schemes*

[URN] *http://www.iana.org/assignments/urn-namespaces*

[X500] International Telecommunication Union — Telecommunication Standardization Sector, *The Directory— Overview of Concepts, Models and Services*, ITU-T X.500, 1993.

Rozdział 12.
TCP — protokół sterowania transmisją (zagadnienia wstępne)

12.1. Wprowadzenie

Dotychczas analizowaliśmy protokoły, które nie zawierają własnych mechanizmów niezawodnego dostarczania danych. Mogą *wykrywać* fakt otrzymania błędnych danych, używając funkcji matematycznej, takiej jak suma kontrolna, lub cyklicznego kodu nadmiarowego **CRC**, ale nie zrobią zbyt wiele, by naprawić błędy. W IP i UDP nie jest wykonywana żadna naprawa błędów. W Ethernecie i innych protokołach na nim opartych przewidziano, co prawda, pewną liczbę prób ponownego przesłania, ale protokoły te poddają się w razie braku sukcesu.

Problem komunikacji w środowiskach, gdzie medium komunikacyjne może gubić lub zmieniać dostarczane przez siebie komunikaty, jest studiowany od lat. Jedna z najważniejszych prac teoretycznych na ten temat została opublikowana przez Claude'a Shannona w 1948 roku [S48]. Praca ta, która spopularyzowała termin „bit" i położyła podwaliny pod **teorię informacji**, pozwala zrozumieć podstawowe ograniczenia dotyczące ilości informacji, którą można przesłać przez kanał informacyjny dopuszczający straty (który może gubić lub zmieniać bity). Teoria informacji jest ściśle powiązana z **teorią kodowania**, która dostarcza takich sposobów kodowania informacji, jakie są — najbardziej jak to możliwe — odporne na błędy powstające w kanale komunikacyjnym. Używanie **kodów z korekcją błędów** (polegających zasadniczo na dodaniu nadmiarowych bitów, tak że prawdziwa informacja może być odzyskana nawet wtedy, gdy niektóre bity są uszkodzone) do naprawiania problemów powstałych w trakcie komunikacji to jedna z bardzo ważnych metod obsługi błędów. Drugą można nazwać po prostu „spróbuj wysłać jeszcze raz", aż w końcu informacja zostanie prawidłowo odebrana. To podejście, znane pod nazwą **ARQ** (*Automatic Repeat Request* — automatyczne żądanie ponownego przesłania), stanowi podstawę wielu protokołów komunikacyjnych, w tym TCP.

12.1.1. ARQ i retransmisja

Kiedy analizujemy nie tylko pojedynczy kanał komunikacyjny, ale kilka, tworzących kaskadę przeskoków, z węzłami pośredniczącymi, zdajemy sobie sprawę, że możemy mieć do czynienia nie tylko z już wspomnianymi typami błędów (błędy bitów w pakietach), ale także z innymi. Problemy mogą się pojawić w routerze pośredniczącym, a są to te typy problemów, które omawialiśmy, opisując protokół IP, czyli zmiana kolejności pakietów, powielanie pakietów, usuwanie (gubienie) pakietów. Protokół z korektą błędów, przeznaczony do obsługi kanału komunikacyjnego z wieloma przeskokami (taki jak IP), musi radzić sobie z tymi problemami. Zbadamy teraz mechanizmy protokołów, które będzie można wykorzystać. Zanim jednak omówimy je na poziomie ogólnym, zobaczymy, jak są używane przez protokół TCP w Internecie.

Prostą metodą radzenia sobie z gubieniem pakietów (i błędami bitów) jest ponowne przesyłanie pakietu, aż zostanie poprawnie odebrany. Wymaga to ustalenia, po pierwsze, czy odbiorca otrzymał pakiet i po drugie, czy odebrany pakiet był identyczny z pakietem wysłanym przez nadawcę. Metoda używana przez odbiorcę w celu zasygnalizowania nadawcy, że pakiet został odebrany, nazywa się **potwierdzeniem** (*acknowledgement*), czyli **ACK**. W jej najbardziej podstawowej formie nadawca wysyła pakiet i oczekuje na potwierdzenie ACK. Kiedy odbiorca otrzymuje pakiet, wysyła sygnał ACK. Kiedy nadawca odbierze ACK, wysyła następny pakiet i ta sama procedura się powtarza. Interesujące są w tym momencie następujące pytania. Po pierwsze, jak długo powinien nadawca oczekiwać na otrzymanie potwierdzenia ACK? Po drugie, co się stanie w przypadku utraty ACK? I po trzecie, a co będzie, jeśli pakiet został wprawdzie otrzymany, ale zawierał błędy?

Jak się przekonamy, pierwsze pytanie okazuje się być nietrywialne. Decydowanie, jak długo czekać, wiąże się z tym, jak długiego czasu oczekiwania na ACK powinien się nadawca *spodziewać*. Ustalenie tego może być trudne; odłóżmy analizę stosownych technik na później, kiedy będziemy omawiać protokół TCP szczegółowo (patrz rozdział 14.). Odpowiedź na drugie pytanie jest łatwiejsza: jeśli pakiet z ACK zostanie zgubiony, nadawca nie może w łatwy sposób odróżnić tego przypadku od sytuacji, gdy utracony został oryginalny pakiet, więc przesyła go ponownie. Oczywiście, w tym przypadku odbiorca może otrzymać dwa lub więcej jednakowych pakietów, więc musi być przygotowany na obsługę tej sytuacji (patrz następny punkt). Jeśli chodzi o trzecie pytanie, możemy odwołać się do systemów kodowania wspomnianych w podrozdziale 12.1. Na ogół dużo łatwiej używa się kodów do *wykrywania* (z wysokim prawdopodobieństwem) błędów w dużym pakiecie, korzystając tylko z kilku bitów, niż do ich naprawiania. Prostsze kody nie są zazwyczaj zdolne do korygowania błędów, ale są w stanie je wykrywać. To dlatego sumy kontrolne i kody CRC są tak popularne. Tak więc, w celu wykrywania błędów w pakiecie korzystamy z jakiejś formy sumy kontrolnej. Kiedy odbiorca otrzymuje pakiet zawierający błąd, powstrzymuje się od wysłania ACK. W końcu nadawca przesyła ponownie pakiet, który dociera do odbiorcy, najlepiej w stanie nieuszkodzonym.

Nawet w prostym scenariuszu przedstawionym powyżej istnieje możliwość, że odbiorca otrzyma zdublowane egzemplarze przesyłanego pakietu. Problem ten jest rozwiązywany przy użyciu **numeru sekwencyjnego** (*sequence number*). Zasadniczo, każdy unikalny pakiet otrzymuje w punkcie źródłowym nowy numer sekwencyjny, kiedy jest wysyłany, i ten numer sekwencyjny jest przenoszony w pakiecie do odbiorcy, który używa

go do ustalenia, czy ten pakiet już przedtem nie został otrzymany; jeśli okaże się, że został otrzymany, odrzuca go.

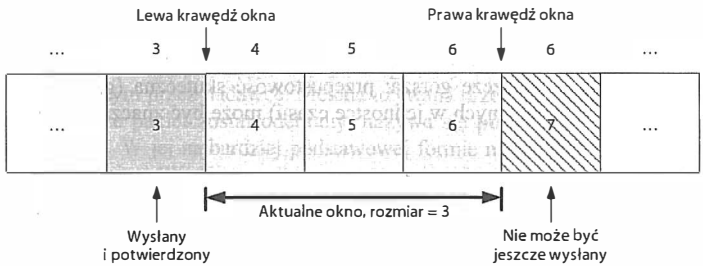
Protokół opisany do tej pory jest niezawodny, ale mało wydajny. Zastanówmy się, co się stanie, gdy czas potrzebny na dostarczenie nawet małego pakietu od nadawcy do odbiorcy (zwany opóźnieniem lub latencją) jest duży (np. jedna lub dwie sekundy, co nie jest niezwykłe w przypadku łączy satelitarnych), a na przesłanie czeka kilka pakietów. Nadawca jest w stanie umieścić pojedynczy pakiet w ścieżce komunikacyjnej, ale potem musi zatrzymać się i czekać, aż odbierze sygnał ACK. Protokół w ten sposób działający jest nazywany protokołem *stop and wait* — zatrzymaj się i czekaj. Jego przepustowość (ilość danych przesłanych przez sieć na jednostkę czasu) jest proporcjonalna do *M*/*R*, gdzie *M* oznacza rozmiar pakietu, a *R* jest to **czas RTT** (*Round-Trip Time* — czas podróży w obie strony, mierzony jako czas potrzebny na przesłanie niewielkiego pakietu od klienta do serwera i z powrotem), przy założeniu, że żadne pakiety nie są tracone lub nieodwracalnie uszkodzone w trakcie przepływu. Przy ustalonym rozmiarze pakietu, w miarę jak rośnie R, przepustowość spada. Jeśli pakiety są tracone lub uszkadzane, sytuacja jest jeszcze gorsza: przepustowość skuteczna (*goodput* — ilość użytecznych danych przesłanych w jednostce czasu) może być znacząco niższa niż przepustowość teoretyczna.

W sieci, która nie zniekształca ani nie gubi wielu pakietów, przyczyną niskiej przepustowości jest zazwyczaj to, że sieć ta nie jest wykorzystywana w sposób ciągły. Sytuację tę można porównać do linii montażowej, na którą nie można wprowadzić kolejnego egzemplarza do zmontowania, zanim montaż poprzedniego nie zostanie całkowicie zakończony. Większa część linii pozostaje bezczynna. Jeśli pójdziemy z tym porównaniem krok dalej, wydaje się oczywiste, że byłoby lepiej, gdyby więcej egzemplarzy znajdowało się jednocześnie na linii montażowej. To samo dotyczy komunikacji sieciowej — gdybyśmy mogli mieć więcej niż jeden pakiet w sieci, moglibyśmy utrzymywać ją w stanie „większej zajętości", co doprowadziłoby do uzyskania wyższej przepustowości.

Dopuszczenie do przebywania w sieci jednocześnie więcej niż jednego pakietu znacznie komplikuje sprawy. Teraz nadawca musi zadecydować nie tylko o tym, kiedy umieścić pakiet w sieci, ale także o tym, jak dużo tych pakietów może umieścić. Musi także wymyśleć, jak utrzymywać zegary odliczające czas oczekiwania na ACK oraz zachowywać kopię każdego jeszcze niepotwierdzonego pakietu na wypadek, gdy niezbędne będą retransmisje. Odbiorca natomiast może potrzebować bardziej wyrafinowanego mechanizmu ACK: takiego, który potrafi odróżnić, które pakiety zostały już odebrane, a których brakuje. Odbiorca może również potrzebować bardziej złożonego mechanizmu buforowania (zapamiętywania pakietów) — takiego, który umożliwi przechowanie pakietów otrzymanych poza kolejnością (tj. tych pakietów, które dotarły wcześniej od aktualnie oczekiwanych, co może wynikać z utraty pakietów lub zmiany kolejności ich przesyłania przez sieć), chyba że decyduje się po prostu na odrzucanie takich pakietów, co jest jednak bardzo nieefektywne. Są także inne kwestie, które mogą nie być oczywiste. Co się stanie, jeśli odbiorca jest wolniejszy od nadawcy? Jeżeli nadawca bezwarunkowo wprowadza do sieci dużą liczbę pakietów, z bardzo dużą szybkością, odbiorca może je odrzucać z powodu własnych ograniczeń w szybkości przetwarzania i wielkości dostępnej pamięci. To samo pytanie dotyczy pośredniczących routerów. Co się dzieje, jeśli infrastruktura sieci nie może obsłużyć szybkości transmisji, która odpowiadałaby nadawcy i odbiorcy?

12.1.2. Okna pakietów i okna przesuwne

By poradzić sobie z tymi wszystkimi problemami, zaczniemy od przyjęcia założenia, że każdy unikalny pakiet posiada numer sekwencyjny, jak pisaliśmy wcześniej. Definiujemy **okno pakietów** jako zbiór pakietów (lub ich numerów sekwencyjnych), które zostały umieszczone w sieci przez nadawcę, ale nie zostały jeszcze całkowicie potwierdzone (tzn. nadawca nie otrzymał dla nich potwierdzenia ACK). Liczbę pakietów w oknie nazywamy **rozmiarem okna**. Termin *okno* wywodzi się z stąd, że kiedy ustawimy wszystkie pakiety wysłane w trakcie sesji komunikacyjnej w długi rząd, ale mamy tylko niewielką szczelinę do ich oglądania, będziemy widzieli tylko ich podzbiór — tak jak to jest przy patrzeniu przez okno. Okno nadawcy (i rząd pozostałych pakietów) można przedstawić graficznie (patrz rysunek 12.1).



Rysunek 12.1. *Okno nadawcy pokazujące, które pakiety nadają się do wysłania (lub już zostały wysłane), które jeszcze nie mogą być wysłane i które już zostały wysłane i potwierdzone. W tym przykładzie rozmiar okna został ustalony na trzy pakiety*

Na rysunku pokazujemy aktualne okno trzech pakietów przy całkowitym rozmiarze okna równym 3. Pakiet numer 3 już został wysłany i potwierdzony, więc jego kopia zachowywana przez nadawcę może być teraz zwolniona. Pakiet numer 7 jest gotowy u nadawcy, ale nie może być wysłany, bo jeszcze nie znajduje się w oknie. Jeśli teraz wyobrazimy sobie, że dane zaczynają przepływać od nadawcy do odbiorcy, a potwierdzenia ACK płyną w przeciwnym kierunku, widzimy, że nadawca mógłby otrzymać potwierdzenie ACK dla pakietu numer 4 jako następne. Kiedy to się zdarzy, okno „przesunie się" w prawo o jeden pakiet, co oznacza, że kopia pakietu numer 4 może zostać zwolniona, a pakiet numer 7 może zostać wysłany. To przesunięcie okna stało się źródłem jeszcze jednej nazwy dla tego typu protokołu — **protokół okna przesuwnego**.

Podejście oparte na zastosowaniu okna przesuwnego można wykorzystać do walki z wieloma dotychczas opisanymi problemami. Zazwyczaj struktura okna jest utrzymywana zarówno po stronie nadawcy, jak i odbiorcy. Po stronie nadawcy kontroluje ona na bieżąco, które pakiety mogą zostać zwolnione, które oczekują na ACK, a które jeszcze nie mogą być wysłane. Zaś po stronie odbiorcy umożliwia śledzenie, które pakiety już zostały odebrane i potwierdzone, które są spodziewane (i jak wiele pamięci zostało przydzielonej do ich przechowania) i które pakiety, nawet jeśli odebrane, nie mogą zostać zachowane z powodu ograniczonej wielkości pamięci. Chociaż struktura okna jest wygodnym narzędziem do bieżącej kontroli przepływu danych między nadawcą i odbiorcą, nie dostarcza wskazówek, jak duże powinno być okno lub co zrobić, jeśli odbiorca lub sieć nie poradzą sobie z szybkością transmisji nadawcy. Teraz zobaczymy, jak te dwie rzeczy są powiązane.

12.1.3. Okna o zmiennym rozmiarze: sterowanie przepływem i kontrola przeciążenia

Jak poradzić sobie z problemem, który powstaje, gdy odbiorca jest zbyt wolny w stosunku do nadawcy? Przedstawimy teraz, jak można zmusić nadawcę do spowolnienia szybkości transmisji, której odbiorca nie może sprostać. Omówimy **sterowaniem przepływem** zwykle realizowane na jeden z dwóch sposobów. Jeden sposób, nazywany **sterowaniem przepływem opartym na szybkości transmisji**, daje nadawcy do dyspozycji pewien limit szybkości transmisji i zapewnia, że dane nigdy nie będą mogły być wysyłane z szybkością, która przekracza przydzielony limit. Ten typ sterowania przepływem jest najwłaściwszy dla aplikacji korzystających ze strumieniowej transmisji danych i może być wykorzystany przy dostarczaniu danych w trybie rozgłoszeniowym lub rozsyłania grupowego (patrz rozdział 9.).

Drugą, dominującą formę sterowania przepływem danych nazywamy **sterowaniem przepływem opartym na oknie danych**; jest to najpopularniejsze podejście korzystające z okien przesuwnych. W tym podejściu rozmiar okna nie jest stały, lecz może się zmieniać w czasie. Aby skutecznie sterować przepływem danych przy użyciu tej techniki, odbiorca musi dysponować metodą przesłania nadawcy sygnału, jak dużego okna należy użyć. Nazywa się to zwykle **propozycją okna** (*window advertisement*) lub **aktualizacją okna** (*window update*). Przesłana wartość jest używana przez nadawcę (tj. odbiorcę propozycji okna) w celu dostosowania rozmiaru własnego okna. Pod względem logicznym aktualizacja okna jest oddzielna w stosunku do potwierdzeń ACK, które omawialiśmy poprzednio, ale w praktyce aktualizacja okna i ACK są przesyłane w pojedynczym pakiecie z uwagi na to, że nadawca na ogół koryguje rozmiar swojego okna w tym samym czasie, kiedy przesuwa je w prawo.

Jeśli zastanowimy się nad efektem zmiany rozmiaru okna u nadawcy, staje się jasne, w jaki sposób osiągana jest kontrola przepływu danych. Nadawcy wolno umieścić W pakietów w sieci, zanim odbierze ACK dla któregokolwiek z nich. Jeśli nadawca i odbiorca są odpowiednio szybcy, a sieć nie traci pakietów i ma nieskończoną pojemność, szybkość transferu jest proporcjonalna do (SW/R) b/s, gdzie W jest rozmiarem okna, S rozmiarem pakietu wyrażonym w bitach, a R to RTT. Kiedy propozycja okna otrzymana od odbiorcy wymaga zmniejszenia wartości W u nadawcy, całkowita szybkość transmisji nadawcy może zostać ograniczona, by nie zalać odbiorcy przesłanymi pakietami. To podejście działa znakomicie w aspekcie ochrony odbiorcy, ale co z siecią znajdującą się pośrodku? Między nadawcą a odbiorcą mogą znajdować się routery dysponujące ograniczonymi zasobami pamięci, które w dodatku muszą zmagać się z wolnymi łączami w sieci. Kiedy taka sytuacja ma miejsce, istnieje możliwość, że szybkość transmisji nadawcy przekroczy zdolność nadążania jakiegoś routera, co doprowadzi do utraty pakietów. Sytuacja ta jest obsługiwana przez specjalną formę sterowania przepływem nazywaną **kontrolą przeciążeń**. Kontrola przeciążeń wiąże się ze spowolnieniem nadawcy, tak aby nie zalewał pakietami sieci pośredniczącej między nim a odbiorcą. Pamiętajmy, że w analizowanej przez nas koncepcji sterowania przepływem danych używaliśmy mechanizmu propozycji okna, by zasygnalizować nadawcy, żeby spowolnił transmisję ze względu na odbiorcę. Nazywamy to **sygnalizacją jawną**, ponieważ protokół używa specjalnego pola, by poinformować nadawcę, co się dzieje. Inną opcją dla nadawcy byłoby **domyślenie się**, że należy zwolnić. Takie podejście zakłada **sygnalizację niejawną** — tzn. wiążącą się z podejmowaniem decyzji o spowolnieniu transmisji na podstawie pewnych innych czynników.

Problem kontroli przeciążenia w sieciach datagramowych, a bardziej ogólnie **teoria kolejek**, z którą jest ściśle powiązany, pozostaje od lat ważnym tematem badań i jest mało prawdopodobne, by kiedykolwiek został rozwiązany całkowicie, dla wszelkich możliwych sytuacji. Analizowanie wszystkich opcji i metod sterowania przepływem danych nie jest celem tej książki. Zainteresowanego czytelnika odsyłamy do [J90], [K97] i [K75]. W rozdziale 16. bardziej szczegółowo zbadamy konkretną technikę kontroli przeciążenia używaną przez protokół TCP, wraz z pewną liczbą wariantów, które powstały w ciągu lat.

12.1.4. Ustalanie czasu oczekiwania na retransmisję

Jedną z najważniejszych kwestii dotyczących wydajności, przed jaką staje projektant niezawodnego protokołu opartego na retransmisji, jest określenie, jak długo należy czekać, zanim uzna się, że pakiet został utracony i powinien być przesłany ponownie. Inaczej mówiąc: jaki powinien być **czas oczekiwania na retransmisję** (*retransmission timeout*). Intuicyjnie uznajemy, że ilość czasu, przez jaki nadawca powinien czekać, zanim wyśle ponownie pakiet, jest mniej więcej sumą następujących czasów: czasu przesłania pakietu, czasu potrzebnego odbiorcy na przetworzenie go i wysłanie ACK, czasu podróży sygnału ACK z powrotem do nadawcy i czasu potrzebnego nadawcy na przetworzenie ACK. Niestety, w praktyce żaden z tych czasów nie jest znany z pewnością. Co gorsza, każdy z nich zmienia się wraz ze zmianą obciążenia hostów końcowych i routerów.

Ponieważ samodzielne podawanie przez użytkownika implementacji protokołu wartości tych wszystkich czasów (lub też aktualizowanie ich na bieżąco) w każdych okolicznościach nie jest praktyczne, lepszą strategią jest wymaganie podjęcia próby ich oszacowania od implementacji protokołu. Nazywa się to **oszacowaniem czasu RTT** i jest procesem statystycznym. Zasadniczo rzeczywisty RTT będzie prawdopodobnie bliski średniej statystycznej dla zbioru próbek czasów RTT. Zauważmy, że średnia ta w naturalny sposób zmienia się w czasie (nie jest stała), jako że ścieżki, po których pakiety wędrują przez sieć, mogą się zmieniać.

Kiedy już uzyskamy pewne oszacowanie RTT, pozostaje wciąż kwestia ustalenia samej wartości czasu oczekiwania, używanej do uruchomienia retransmisji. Po przypomnieniu sobie definicji średniej widzimy, że nigdy nie może ona przyjąć największej wartości ze zbioru próbek (chyba że wszystkie próbki są jednakowe). Zatem nie byłoby rozsądne ustawienie zegara retransmisji na wartość równą wartości estymatora średniej i w ten sposób indukowanie niepożądanych retransmisji, ponieważ prawdopodobnie wiele rzeczywistych czasów RTT będzie od niej większych. Jest oczywiste, że czas oczekiwania powinien być ustawiony na jakąś wartość większą niż średnia, ale jaka dokładnie powinna być relacja między tymi wartościami (lub nawet czy średnia powinna być bezpośrednio użyta) nie jest jeszcze jasne. Ustawienie zbyt dużego czasu oczekiwania jest również niewskazane, gdyż prowadzi do pozwolenia na bezczynność sieci, przy zmniejszonej szybkości przesyłania danych. Odłóżmy dalsze zgłębianie tego tematu do rozdziału 14., w którym zbadamy, jak TCP podchodzi do tego problemu.

12.2. Wprowadzenie do TCP

Korzystając z przygotowania, jakie teraz mamy w kwestiach wpływających na niezawodne dostarczanie danych na poziomie ogólnym, przyjrzyjmy się, jaką rolę one spełniają w TCP i jakich usług ten protokół dostarcza aplikacjom internetowym. Przyjrzymy się także polom znajdującym się w nagłówku TCP (zauważymy przy tym, jak wiele spośród pojęć dotychczas poznanych, np. potwierdzenie ACK, propozycja okna, jest zawartych w opisie nagłówka). W następnych rozdziałach zbadamy te pola nagłówka bardziej szczegółowo.

Nasz opis protokołu TCP rozpoczyna się w tym rozdziale i będzie kontynuowany w następnych pięciu rozdziałach. W rozdziale 13. piszemy, jak połączenie TCP jest ustanawiane i kończone. W rozdziale 14. wyjaśniamy szczegółowo, jak TCP szacuje RTT dla połączenia i w jaki sposób czas oczekiwania na retransmisję zostaje oparty na tym oszacowaniu. W rozdziale 15. przyglądamy się zwykłemu transferowi danych, a zaczynamy od aplikacji interaktywnych (takich jak czat). Następnie opisujemy zarządzanie oknami i sterowanie przepływem, które ma zastosowanie i do interaktywnych aplikacji, i do aplikacji przesyłających dane masowe (np. transfer plików), a także obsługiwany przez TCP **tryb pilnych danych**, który pozwala na oznaczenie pewnych danych w strumieniu jako mających specjalne znaczenie. W rozdziale 16. opisujemy algorytmy kontroli przeciążeń w TCP, które pozwalają zredukować utratę pakietów, kiedy sieć jest bardzo zajęta. Analizujemy także pewne modyfikacje, które zostały zaproponowane w celu zwiększenia przepustowości w szybkich sieciach lub poprawienia odporności w stratnych (np. bezprzewodowych) sieciach. Na koniec, w rozdziale 17. pokazujemy, w jaki sposób TCP utrzymuje połączenia w stanie aktywnym, w sytuacji gdy nie ma przepływu danych.

Oryginalna specyfikacja protokołu TCP jest zawarta w [RFC0793], chociaż pewne błędy znajdujące się w tym dokumencie RFC zostały poprawione w RFC podającym wymagania dotyczące hostów [RFC1122]. Od tamtego czasu specyfikacje dla TCP zostały zrewidowane i rozszerzone przez włączenie wyjaśnionego i ulepszonego działania kontroli przeciążeń [RFC5681] [RFC3782] [RFC3517] [RFC3390] [RFC3168], czasów oczekiwania na retransmisje [RFC6298] [RFC5682] [RFC4015], stosowania NAT [RFC5382], funkcjonowania potwierdzeń [RFC2883], bezpieczeństwa [RFC6056] [RFC5927] [RFC5926], zarządzania połączeniem [RFC5482] i wytycznych dotyczących implementacji trybu pilnych danych [RFC6093]. Pojawił się także bogaty wachlarz modyfikacji eksperymentalnych obejmujących funkcjonowanie retransmisji [RFC5827] [RFC3708], wykrywanie i kontrolę przeciążeń [RFC5690] [RFC5562] [RFC4782] [RFC3649] [RFC2861] oraz inne elementy. Na koniec, czynione są wysiłki, by zbadać, jak TCP mógłby skorzystać z wielu jednoczesnych ścieżek warstwy sieciowej [RFC6182].

12.2.1. Model usług TCP

Jeśli nawet TCP i UDP wykorzystują tę samą warstwę sieci (IPv4 lub IPv6), TCP oferuje warstwie aplikacji całkowicie inną usługę niż protokół UDP. TCP dostarcza usługę **połączeniową**, niezawodną, operującą na strumieniach bajtów. Termin **połączeniowa** oznacza, że dwie aplikacje stosujące TCP muszą ustanowić połączenie TCP, nawiązując kontakt, zanim będą mogły wymieniać dane. Typową analogią jest wykręcanie numeru

telefonicznego, czekanie, aż druga strona podniesie słuchawkę i powie „Halo", a potem „Kto mówi?". Tak samo zachowują się dwa punkty końcowe komunikujące się w połączeniu TCP; takie pojęcia jak rozgłoszenie (*broadcasting*) czy rozsyłanie grupowe (*multicasting*) (patrz rozdział 9.) nie mają zastosowania w TCP.

Protokół TCP dostarcza abstrakcji strumienia bajtów aplikacjom, które go używają. Konsekwencją tej decyzji projektowej jest to, że żadne znaczniki końca rekordu czy granic komunikatu nie są automatycznie wstawiane przez TCP (patrz rozdział 1.). Znacznik końca rekordu wskazuje rozmiar danych wyjściowych pojedynczej operacji zapisu wykonanej przez aplikację. Jeśli aplikacja na jednym końcu połączenia przekaże 10 bajtów, następnie 20 bajtów, wreszcie 50 bajtów w kolejnych operacjach zapisu, aplikacja na drugim końcu połączenia nie będzie mogła rozpoznać, jakiej długości były poszczególne zapisy. Przykładowo drugi koniec mógłby odczytać te 80 bajtów w czterech operacjach odczytu po 20 bajtów lub jakoś inaczej. Jeden punkt końcowy wysyła strumień bajtów do TCP i identyczny strumień bajtów pojawia się na drugim końcu połączenia. Każdy punkt końcowy niezależnie wybiera rozmiary swoich odczytów i zapisów.

TCP w ogóle nie interpretuje zawartości bajtów w strumieniu. Nie ma pojęcia, czy wymieniane między węzłami końcowymi bajty danych są danymi binarnymi, znakami kodu ASCII, znakami kodu EBCDIC czy czymś jeszcze innym. Interpretacja tego strumienia bajtów należy do aplikacji operujących na każdym końcu połączenia. TCP obsługuje jednak wspomniany wcześniej tryb pilnych danych, chociaż nie zaleca się już jego stosowania.

12.2.2. Niezawodność w TCP

TCP zapewnia niezawodność przy użyciu specyficznych odmian technik właśnie opisanych. Przy dostarczaniu interfejsu strumieniowego TCP musi wykonywać konwersję strumienia bajtowego wysyłanego przez aplikację na zestaw pakietów, które mogą być przenoszone przez protokół IP. Nazywa się to **pakietowaniem** (*packetization*). Pakiety te zawierają numery sekwencyjne, które w TCP w rzeczywistości reprezentują offset bajtowy (czyli przesunięcie względem określonej bazy wyrażone w bajtach) pierwszego bajta każdego pakietu w całym strumieniu danych, a nie numer pakietu. To pozwala, żeby pakiety posiadały zmienny rozmiar w trakcie transferu, co umożliwia również ich łączenie. Nazywamy to **przepakietowaniem** (*repacketization*). Dane aplikacji zostają rozbite na porcje, które według TCP mają optymalny rozmiar do przesłania przez sieć, zazwyczaj przez wpasowanie każdego segmentu w pojedynczy datagram warstwy IP, który nie będzie poddany fragmentacji. Różni się to od UDP, gdzie każda operacja zapisu wykonana przez aplikację zwykle generuje datagram o rozmiarze dostosowanym do wielkości tego zapisu (plus nagłówki). Porcja przekazana przez TCP protokołowi IP nazywa się **segmentem** (patrz rysunek 12.2). W rozdziale 15. zobaczymy, jak TCP decyduje o tym, jaki rozmiar powinien mieć segment.

TCP utrzymuje obowiązkową sumę kontrolną obejmującą własny nagłówek, wszystkie dane przekazane przez obsługiwaną aplikację oraz niektóre pola nagłówka IP. Jest to kompleksowa suma kontrolna obejmująca cały segment i tzw. **pseudonagłówek** (zawierający pola z nagłówka IP), której celem jest wykrycie wszystkich błędów bitów powstałych w trakcie transferu. Jeśli segment przybywa z nieprawidłową sumą kontrolną, TCP go odrzuca bez wysyłania potwierdzenia dla odrzuconego pakietu. TCP po stronie

odbiorcy może jednakże potwierdzić **poprzedni** (już potwierdzony) segment, aby w ten sposób pomóc nadawcy w jego obliczeniach związanych z kontrolą przeciążenia (patrz rozdział 16.). Do obliczenia sumy kontrolnej TCP używa tej samej funkcji matematycznej, co inne protokoły Internetu (UDP, ICMP itd.). Dla dużych transferów danych istnieje pewna obawa, że ta suma kontrolna nie jest wystarczająco silna [SP00], tak więc ostrożne aplikacje powinny stosować swoje własne metody ochrony przed błędami (np. silniejsze sumy kontrolne lub kody CRC) albo skorzystać z usługi warstwy pośredniej dla osiągnięcia tego samego rezultatu (np. patrz [RFC5044]).

Kiedy TCP wysyła grupę segmentów, zwykle ustawia jeden zegar retransmisji, czekając na potwierdzenie odbioru przez drugą stronę. TCP nie ustawia oddzielnego zegara retransmisji dla każdego segmentu. Zazwyczaj ustawia zegar wtedy, gdy wysyła okno danych, i aktualizuje czas oczekiwania na retransmisję, kiedy napływają potwierdzenia ACK. Jeśli potwierdzenie nie zostanie odebrane w przewidzianym czasie, segment jest transmitowany ponownie. W rozdziale 14. bardziej szczegółowo przyjrzymy się stosowanej przez TCP adaptacyjnej strategii określania czasu oczekiwania na retransmisję.

Kiedy TCP otrzymuje dane od drugiej strony połączenia, wysyła potwierdzenie. To potwierdzenie nie zawsze jest wysyłane natychmiast, zwykle jest opóźnione o ułamek sekundy. Potwierdzenia ACK są **zbiorcze** w tym sensie, że ACK, które wskazuje numer bajta N, implikuje, że wszystkie bajty do numeru N (z wyłączeniem numeru N) zostały odebrane z sukcesem. Zapewnia to jakąś odporność na utratę potwierdzeń ACK — jeśli jakieś ACK zostanie zgubione, jest wielce prawdopodobne, że kolejne ACK wystarczy do potwierdzenia poprzednich segmentów.

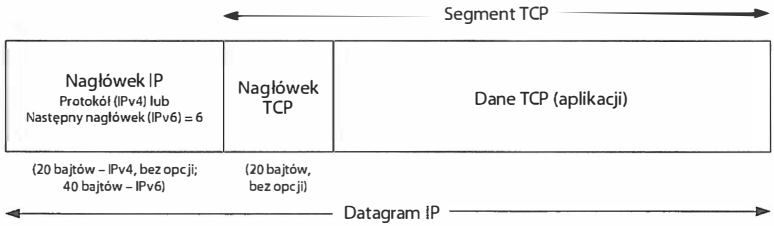
TCP zapewnia warstwie aplikacji usługę **pełnego dupleksu**. Oznacza to, że dane mogą płynąć w każdym kierunku w sposób niezależny od drugiego kierunku. Dlatego każda strona połączenia musi utrzymywać numer sekwencyjny danych dla obu kierunków przepływu. Kiedy połączenie zostanie już ustanowione, każdy segment TCP zawierający dane płynące w jednym kierunku połączenia zawiera również potwierdzenie ACK dla segmentów przepływających w kierunku odwrotnym. Każdy segment zawiera także propozycję okna służącą implementacji sterowania przepływem w odwrotnym kierunku. W ten sposób, w momencie nadejścia pakietu mogą nastąpić trzy zdarzenia naraz: przesunięcie okna do przodu (czyli w prawo), zmiana rozmiaru okna i przyjęcie nowych danych. Jak się przekonamy w rozdziale 13., w pełni aktywne połączenie TCP jest dwukierunkowe i symetryczne; dane mogą przepływać równie dobrze w każdym kierunku.

Używając numerów sekwencyjnych, TCP po stronie odbiorcy odrzuca zdublowane segmenty i zmienia porządek segmentów, które przychodzą w niewłaściwej kolejności. Pamiętajmy, że każda z tych nieprawidłowości może się zdarzyć z tego powodu, że TCP korzysta z protokołu IP, dostarczając swoje segmenty, a IP nie zapewnia eliminacji dublowanych pakietów i nie gwarantuje ich właściwej kolejności. Ponieważ jednak TCP jest protokołem strumieniowym, nigdy nie dostarcza danych odbierającej je aplikacji w niewłaściwym porządku. Dlatego TCP po stronie odbiorcy może być zmuszony do zatrzymania danych z większymi numerami sekwencyjnymi przed przekazaniem ich aplikacji, dopóki brakujący segment z niższym numerem sekwencyjnym (czyli „dziura") nie wypełni luki.

Teraz zaczniemy analizować pewne szczegóły protokołu TCP. W tym rozdziale przedstawimy jedynie enkapsulację i strukturę nagłówka TCP. Inne szczegóły pojawią się w następnych pięciu rozdziałach. Protokół TCP może być używany zarówno z IPv4, jak i z IPv6, a suma kontrolna z włączeniem pseudonagłówka, której TCP używa (podobna do sumy kontrolnej UDP), jest stosowana obowiązkowo bez względu na wersję protokołu IP.

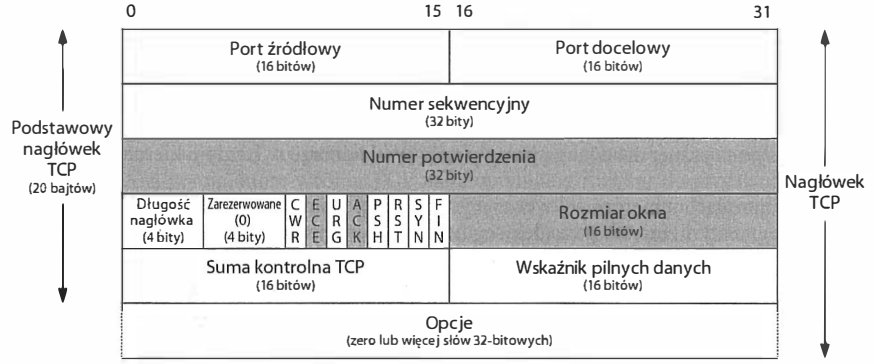
12.3. Nagłówek TCP i enkapsulacja

Segmenty TCP są poddane enkapsulacji w datagramach IP, co pokazano na rysunku 12.2.



Rysunek 12.2. *Nagłówek TCP występuje bezpośrednio po nagłówku IP lub po ostatnim nagłówku dodatkowym IPv6 i często ma długość 20 bajtów (bez opcji TCP). Z opcjami wielkość nagłówka może wynosić do 60 bajtów. Do powszechnie występujących opcji należą: Maksymalny rozmiar segmentu, Znacznik czasu, Skalowanie okna i Selektywne potwierdzenia ACK*

Sam nagłówek jest znacznie bardziej skomplikowany niż nagłówek UDP, który poznaliśmy w rozdziale 10. Nie jest to zbyt zaskakujące, jako że TCP jest znacząco bardziej skomplikowanym protokołem, który musi stale informować obie strony połączenia (inaczej synchronizować je) o aktualnym stanie. Jest to pokazane na rysunku 12.3.



Rysunek 12.3. *Nagłówek TCP. Jego normalny rozmiar wynosi 20 bajtów, chyba że występują opcje. Pole Długość nagłówka podaje rozmiar nagłówka w 32-bitowych słowach (wartość minimalna wynosi 5). Pola zacienione (Numer potwierdzenia, Rozmiar okna plus bity ECE i ACK) odnoszą się do danych przepływających w kierunku przeciwnym, tzn. w kierunku do nadawcy tego segmentu*

Każdy nagłówek TCP zawiera numery portu źródłowego i docelowego. Te dwie wartości, razem z adresami IP źródłowym i docelowym w nagłówku protokołu IP, w jednoznaczny sposób identyfikują każde połączenie. Kombinacja adresu IP i numeru portu jest czasem nazywana **punktem końcowym** lub **gniazdem** w literaturze dotyczącej protokołu TCP. Ten drugi termin pojawił się w [RFC0793] i został w końcu przyjęty jako nazwa interfejsu programowego do komunikacji sieciowej pochodzącego z Uniwersytetu Berkeley (nazywanego teraz często gniazdami Berkeley, *Berkeley Sockets*). To właśnie *para* gniazd lub punktów końcowych (poczwórna wartość składająca się z adresu IP klienta, numeru portu klienta, adresu IP serwera i numeru portu serwera) jednoznacznie identyfikuje każde połączenie TCP. Fakt ten stanie się ważny, kiedy popatrzymy, jak serwer TCP może komunikować się z wieloma klientami (patrz rozdział 13.).

Pole numeru sekwencyjnego identyfikuje bajt w strumieniu danych przepływających od TCP nadającego do TCP odbierającego, który reprezentuje pierwszy bajt danych zawartych w konkretnym segmencie. Jeśli postrzegamy dane jako strumień bajtów przepływających w jednym kierunku miedzy dwoma aplikacjami, widzimy, że TCP nadaje numer sekwencyjny każdemu pojedynczemu bajtowi. Ten numer sekwencyjny jest 32-bitową liczbą całkowitą bez znaku, która przyjmuje na powrót wartość 0 po osiągnięciu wartości 2^{32}–1. Ponieważ każdy wymieniany między stronami połączenia bajt jest numerowany, pole *Numer potwierdzenia* (nazywane także *Numer ACK* lub krótko polem *ACK*) zawiera następny numer sekwencyjny, który nadawca potwierdzenia spodziewa się otrzymać. Dlatego jest to numer sekwencyjny ostatniego skutecznie odebranego bajta danych plus 1. Pole to jest istotne tylko wtedy, gdy pole bitu *ACK* (opisane później w tym podrozdziale) ma ustawioną wartość 1, co zwykle ma miejsce dla wszystkich segmentów z wyjątkiem segmentu początkowego i końcowego. Wysłanie ACK nie wymaga nic więcej, jak tylko przesłania dowolnego segmentu TCP, ponieważ 32-bitowe pole *Numer ACK* jest zawsze częścią nagłówka, podobnie jak pole bitu *ACK*.

Kiedy jest ustanawiane nowe połączenie, pole bitu *SYN* w pierwszym segmencie wysłanym przez klienta do serwera zawiera wartość 1. Takie segmenty nazywamy segmentami *SYN*. Pole *Numer sekwencyjny* zawiera wtedy pierwszy numer sekwencyjny, który zostanie użyty w tym kierunku połączenia, i stanowi bazę dla kolejnych numerów sekwencyjnych i zwracanych numerów ACK (pamiętajmy, że wszystkie połączenia są dwukierunkowe). Zauważmy, że numer ten nie jest równy 0 ani 1, ale jest całkiem inną liczbą, często wybraną losowo, która nazywa się **początkowym numerem sekwencyjnym (ISN** — *Initial Sequence Number*). Powodem tego, że ISN nie jest równy 0 lub 1, są względy bezpieczeństwa, będzie to analizowane w rozdziale 13. Numer sekwencyjny pierwszego bajta danych przesyłanych w tym kierunku połączenia jest równy ISN plus 1, ponieważ pole bitu *SYN* **konsumuje** jeden numer sekwencyjny. Jak zobaczymy później, konsumowanie numeru sekwencyjnego implikuje również niezawodne dostarczanie danych przy użyciu retransmisji. Dlatego bity *SYN* i bajty danych aplikacji (i bity *FIN*, o czym przekonamy się później) są niezawodnie dostarczane. Potwierdzenia ACK, które nie konsumują numerów sekwencyjnych, nie są dostarczane w niezawodny sposób.

TCP można opisać jako „protokół okna przesuwnego z kumulatywnymi pozytywnymi potwierdzeniami". Pole *Numer ACK* jest tak skonstruowane, by wskazywać najwyższy numer bajta otrzymanego przez odbiorcę w swojej kolejności (tzn. gdy wszystkie bajty o niższych numerach sekwencyjnych zostały już prawidłowo odebrane). Jeśli np. bajty od 1 do 1024 są odebrane prawidłowo, a następny segment zawiera bajty 2049–3072,

odbiorca nie może użyć zwykłego pola *Numer ACK*, by dać sygnał nadawcy, że już odebrał ten nowy segment. Nowoczesne implementacje TCP mają jednak opcję **potwierdzenia selektywnego** (*selective acknowledgement*, **SACK**), która pozwala odbiorcy na przesłanie do nadawcy informacji o danych otrzymanych poza kolejnością, ale odebranych prawidłowo. Jeśli opcję tę połączyć ze zdolnością nadawcy do używania **powtórzeń selektywnych**, można uzyskać znaczącą korzyść w aspekcie wydajności [FF96]. W rozdziale 14. zobaczymy, jak TCP wykorzystuje **zduplikowane potwierdzenia** (poprzez powielanie segmentów z tym samym potwierdzeniem ACK) w celu ulepszenia swoich procedur kontroli przeciążeń i kontroli błędów.

Pole *Długość nagłówka* podaje długość nagłówka wyrażoną w 32-bitowych słowach. Jest to wymagane, ponieważ długość pola *Opcje* jest zmienna. Ze względu na 4-bitowy rozmiar tego pola, długość nagłówka jest ograniczona do 60 bajtów. Rozmiar nagłówka, który nie zawiera pola *Opcje*, wynosi 20 bajtów.

Aktualnie w nagłówku TCP jest zdefiniowanych osiem pól 1-bitowych, chociaż niektóre starsze implementacje protokołu rozumieją tylko ostatnie sześć[1]. Bity jednego lub więcej z nich mogą być jednocześnie „włączone" (czyli ustawione na 1). W tym miejscu podajemy w skrócie ich zastosowanie, ale zapowiadamy bardziej szczegółowe omówienie w dalszych rozdziałach.

1. ***CWR*** (*Congestion Window Reduced*) — nadawca zmniejszył swoją szybkość transmisji; patrz rozdział 16.
2. ***ECE*** (*ECN Echo*) — nadawca otrzymał wcześniej powiadomienie o przeciążeniu; patrz rozdział 16.
3. ***URG*** (*Urgent*) — pole *Wskaźnik pilnych danych* jest istotne — rzadko używane; patrz rozdział 15.
4. ***ACK*** (*Acknowledgement*) — pole *Numer potwierdzenia* jest istotne — zawsze włączone po ustanowieniu połączenia; patrz rozdziały 13. i 15.
5. ***PSH*** (*Push*) — odbiorca powinien przekazać te dane aplikacji tak szybko, jak to możliwe — niezaimplementowane w niezawodny sposób i nieużywane; patrz rozdział 15.
6. ***RST*** (*Reset*) — zresetowanie połączenia, zwykle z powodu błędu; patrz rozdział 13.
7. ***SYN*** (*Synchronize*) — synchronizacja numerów sekwencyjnych w celu zainicjowania połączenia; patrz rozdział 13.
8. ***FIN*** (*Finished*) — nadawca segmentu skończył wysyłanie danych do swojego odbiorcy; patrz rozdział 13.

Sterowanie przepływem w TCP odbywa się za pomocą zaproponowania przez każdą stronę połączenia rozmiaru okna przy wykorzystaniu pola *Rozmiar okna*. Jest to liczba bajtów, począwszy od bajta określonego przez numer ACK, jaką odbiorca jest skłonny przyjąć. To pole 16-bitowe ograniczające wielkość okna do 65 535 bajtów i w ten sposób ograniczające wydajność TCP w odniesieniu do przepustowości. W rozdziale 15. przy-

[1] Zauważmy, że [RFC3540], eksperymentalny RFC, definiuje także najmniej znaczący bit pola *Zarezerwowane* jako *NS* (*Nonce Sum* — suma identyfikatorów jednorazowych).

patrzymy się opcji ***Skalowanie okna*** (*Window Scale*), która umożliwia skalowanie zawartości pola *Rozmiar okna*, co pozwala na definiowanie dużo większych okien i zapewnia zwiększoną wydajność w szybkich sieciach i sieciach z dużym opóźnieniem.

Pole *Suma kontrolna TCP* obejmuje nagłówek TCP i dane oraz niektóre pola z nagłówka IP; jest obliczana z uwzględnieniem utworzonego wcześniej pseudonagłówka, podobnego do tego, który jest stosowany w protokołach ICMPv6 i UDP, co omawialiśmy w rozdziałach 8. i 10. Suma kontrolna jest obowiązkowo obliczana i zapisywana przez nadawcę, a potem weryfikowana przez odbiorcę. Sumę kontrolną TCP obliczamy przy użyciu tego samego algorytmu, co sumy kontrolne używane w protokołach IP, ICMP i UDP.

Pole *Wskaźnik pilnych danych* jest istotne tylko przy ustawionym bicie *URG*. Wskaźnik ten stanowi wartość dodatniego offsetu, który należy dodać do zawartości pola *Numer sekwencyjny* segmentu, by uzyskać numer sekwencyjny ostatniego bajta pilnych danych. Tryb pilnych danych używany przez TCP daje nadawcy możliwość dostarczenia specjalnie oznaczonych danych drugiej stronie połączenia.

Najczęściej spotykanym składnikiem pola *Opcje* jest opcja *Maksymalny rozmiar segmentu*, w skrócie ***MSS*** (*Maximum Segment Size*). Każda strona połączenia zazwyczaj określa wartość tej opcji w pierwszym wysłanym przez siebie segmencie (czyli zawierającym ustawiony bit *SYN* służący do nawiązania połączenia). Opcja *MSS* określa, jak duży może być największy segment, jaki nadawca tej opcji może otrzymać od drugiej strony. Opisujemy opcję *MSS* bardziej szczegółowo w rozdziale 13., a niektóre inne opcje TCP w rozdziałach 14. i 15. Inne często występujące opcje to *SACK*, *Znacznik czasu* (*Timestamp*) i *Skalowanie okna* (*Window Scale*).

Odnosząc się do rysunku 12.2, zauważmy, że część segmentu TCP zawierająca dane jest opcjonalna. Zobaczymy w rozdziale 13., że kiedy połączenie jest ustanawiane i kiedy połączenie jest kończone, są wymieniane segmenty, które zawierają tylko nagłówki TCP (z opcjami lub bez), ale żadnych danych. Segment niezawierający danych jest także używany do potwierdzenia otrzymanych danych, jeśli brak jest danych do przesłania w danym kierunku (nazywamy go **czystym ACK**), lub do powiadomienia partnera komunikacji o zmianie rozmiaru okna (wtedy nazywany jest aktualizacją okna). Są też pewne przypadki wynikające z upływania czasów oczekiwania, w których segment może być przesłany bez danych.

12.4. Podsumowanie

Problem zapewnienia niezawodnej komunikacji przez stratne kanały komunikacyjne jest studiowany od lat. Dwie główne metody radzenia sobie z błędami obejmują kody z korekcją błędów i retransmisję danych. Protokoły używające retransmisji powinny także obsługiwać utratę danych, zwykle przez ustawienie zegara odliczającego czas oczekiwania, i muszą również ustalić jakiś sposób, którego użyje odbiorca do zasygnalizowania nadawcy, co od niego otrzymał. Rozstrzyganie, jak długo czekać na potwierdzenie ACK, jest skomplikowane, jako że odpowiedni czas może się zmieniać w ślad za zmianami routingu w sieci czy obciążenia systemów końcowych. Współczesne protokoły oszacowują czas podróży w obie strony i ustawiają zegar oczekiwania na retransmisję w oparciu o pewną funkcję tych pomiarów.

Z wyjątkiem procedury ustawiania zegara retransmisji, protokoły bazujące na retransmisji są proste, gdy tylko jeden pakiet może znajdować się w sieci w tym samym czasie, ale osiągają słabą wydajność w sieciach z dużym opóźnieniem. W celu uzyskania większej wydajności w sieci musi zostać umieszczonych wiele pakietów, zanim zostanie odebrane potwierdzenie ACK. To podejście jest bardziej efektywne, ale i bardziej złożone. Typową metodą radzenia sobie z tą złożonością jest użycie okien przesuwnych, gdzie pakiety są oznaczone numerami sekwencyjnymi, a rozmiar okna ogranicza liczbę tych pakietów. Kiedy rozmiar okna można zmieniać, w oparciu o informację zwrotną od odbiorcy lub o inne sygnały (takie jak utracone pakiety), mogą zostać wdrożone i sterowanie przepływem, i kontrola przeciążeń.

TCP oferuje niezawodną, połączeniową, strumieniową usługę warstwy transportowej utworzoną przy użyciu wielu opisanych technik. Wykonaliśmy krótki przegląd wszystkich pól nagłówka TCP, zauważając, że większość z nich jest bezpośrednio związana z przedstawionymi wyżej abstrakcyjnymi koncepcjami dotyczącymi niezawodnego dostarczania danych. Zbadamy je szczegółowo w następnych rozdziałach. TCP pakietuje dane aplikacji w postaci segmentów, ustawia czas oczekiwania na retransmisję, ilekroć wysyła dane, potwierdza dane otrzymane przez drugą stronę, porządkuje dane, które dotarły do węzła odbiorcy poza kolejnością, odrzuca zduplikowane dane, zapewnia całościową kontrolę przepływu oraz oblicza i weryfikuje obowiązkową, kompleksową sumę kontrolną. Jest najszerzej używanym protokołem w Internecie. Jest wykorzystywany przez większość popularnych aplikacji, takich jak HTTP, SSH/TLS, NetBIOS (NBT — NetBIOS przez TCP), Telnet, FTP i pocztę elektroniczną (SMTP). Wiele rozproszonych aplikacji dystrybucji plików (np. BitTorrent, Shareaza) także używa TCP.

12.5. Bibliografia

[FF96] K. Fall, S. Floyd, *Simulation-Based Comparisons of Tahoe, Reno and SACK TCP*, „ACM Computer Communications Review", lipiec 1996.

[J90] R. Jain, *Congestion Control in Computer Networks: Issues and Trends*, „IEEE Network Magazine", maj 1990.

[K75] L. Kleinrock, *Queuing Systems, Volume I: Theory*, Wiley-Interscience, 1975.

[K97] S. Keshav, *An Engineering Approach to Computer Networking*, Addison-Wesley, 1997. (Drugie wydanie w przygotowaniu).

[RFC0793] J. Postel, *Transmission Control Protocol*, Internet RFC 0793/STD 0007, wrzesień 1981.

[RFC1122] R. Braden, ed., *Requirements for Internet Hosts—Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC2861] M. Handley, J. Padhye, S. Floyd, *TCP Congestion Window Validation*, Internet RFC 2861 (experimental), czerwiec 2000.

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, Internet RFC 2883, lipiec 2000.

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, Internet RFC 3168, wrzesień 2001.

[RFC3390] M. Allman, S. Floyd, C. Partridge, *Increasing TCP's Initial Window*, Internet RFC 3390, październik 2002.

[RFC3517] E. Blanton, M. Allman, K. Fall, L. Wang, *A Conservative Selective Acknowledgment (SACK)-Based Loss Recovery Algorithm for TCP*, Internet RFC 3517, kwiecień 2003.

[RFC3540] N. Spring, D. Wetherall, D. Ely, *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*, Internet RFC 3540 (experimental), czerwiec 2003.

[RFC3649] S. Floyd, *HighSpeed TCP for Large Congestion Windows*, Internet RFC 3649 (experimental), grudzień 2003.

[RFC3708] E. Blanton, M. Allman, *Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions*, Internet RFC 3708 (experimental), luty 2004.

[RFC3782] S. Floyd, T. Henderson, A. Gurtov, *The NewReno Modification to TCP's Fast Recovery Algorithm*, Internet RFC 3782, kwiecień 2004.

[RFC4015] R. Ludwig, A. Gurtov, *The Eifel Response Algorithm for TCP*, Internet RFC 4015, luty 2005.

[RFC4782] S. Floyd, M. Allman, A. Jain, P. Sarolahti, *Quick-Start for TCP and IP*, Internet RFC 4782 (experimental), styczeń 2007.

[RFC5044] P. Culley, U. Elzur, R. Recio, S. Bailey, J. Carrier, *Marker PDU Aligned Framing for TCP Specification*, Internet RFC 5044, październik 2007.

[RFC5382] S. Guha, red., K. Biswas, B. Ford, S. Sivakumar, P. Srisuresh, *NAT Behavioral Requirements for TCP*, Internet RFC 5382/BCP 0142, październik 2008.

[RFC5482] L. Eggert, F. Gont, *TCP User Timeout Option*, Internet RFC 5482, marzec 2009.

[RFC5562] A. Kuzmanovic, A. Mondal, S. Floyd, K. Ramakrishnan, *Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets*, Internet RFC 5562 (experimental), czerwiec 2009.

[RFC5681] M. Allman, V. Paxson, E. Blanton, *TCP Congestion Control*, Internet RFC 5681, wrzesień 2009.

[RFC5682] P. Sarolahti, M. Kojo, K. Yamamoto, M. Hata, *Forward RTORecovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP*, Internet RFC 5682, wrzesień 2009.

[RFC5690] S. Floyd, A. Arcia, D. Ros, J. Iyengar, *Adding Acknowledgement Congestion Control to TCP*, Internet RFC 5690 (informational), luty 2010.

[RFC5827] M. Allman, K. Avrachenkov, U. Ayesta, J. Blanton, P. Hurtig, *Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)*, Internet RFC 5827 (experimental), maj 2010.

[RFC5926] G. Lebovitz, E. Rescorla, *Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)*, Internet RFC 5926, czerwiec 2010.

[RFC5927] F. Gont, *ICMP Attacks against TCP*, Internet RFC 5927 (experimental), lipiec 2010.

[RFC6056] M. Larsen, F. Gont, *Recommendations for Transport-Protocol Port Randomization*, Internet RFC 6056/BCP 0156, styczeń 2011.

[RFC6093] F. Gont, A. Yourtchenko, *On the Implementation of the TCP Urgent Mechanism*, Internet RFC 6093, styczeń 2011.

[RFC6182] A. Ford, C. Raiciu, M. Handley, S. Barre, J. Iyengar, *Architectural Guidelines for Multipath TCP Development*, Internet RFC 6182 (informational), marzec 2011.

[RFC6298] V. Paxson, M. Allman, J. Chu, M. Sargent, *Computing TCP's Retransmission Timer*, Internet RFC 6298, czerwiec 2011.

[S48] C. Shannon, *A Mathematical Theory of Communication*, „Bell System Technical Journal", lipiec/październik 1948.

[SP00] J. Stone, C. Partridge, „When the CRC and TCP Checksum Disagree", *Proc. ACM SIGCOMM*, sierpień/wrzesień 2000.

Rozdział 13.
Zarządzanie połączeniem TCP

13.1. Wprowadzenie

TCP jest **połączeniowym** protokołem obsługującym transmisje pojedyncze (*unicast*). Zanim którakolwiek strona połączenia będzie mogła przesłać dane drugiej, musi być ustanowione między nimi połączenie. W tym rozdziale rozważymy szczegółowo, co to jest połączenie TCP, jak jest ono ustanawiane i jak kończone. Przypomnijmy sobie, że model usług TCP przewiduje dostarczanie danych do aplikacji w postaci strumienia bajtów. TCP wykrywa i naprawia zasadniczo wszystkie problemy związane z przesyłaniem danych, które mogą się zdarzyć na skutek utraty czy powielania pakietów lub błędów występujących na poziomie warstwy IP (lub poniżej).

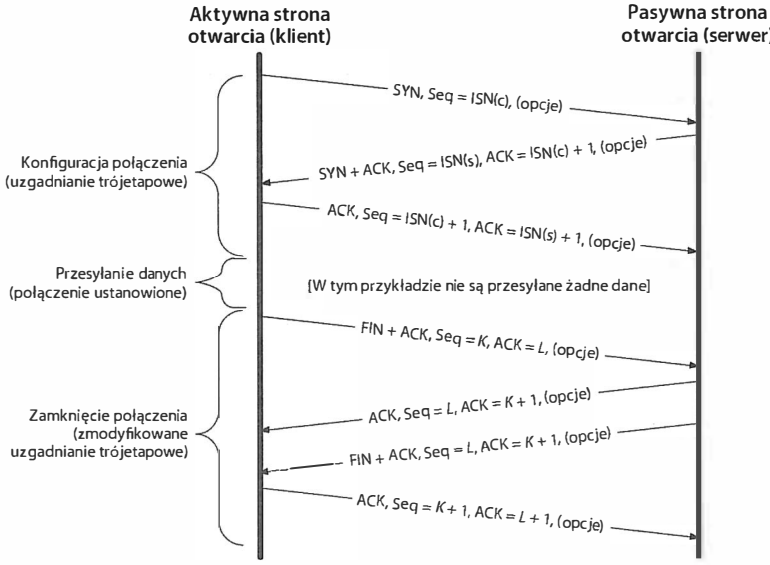
Ze względu na wykonywane zarządzanie **stanem połączenia** (informacją o połączeniu utrzymywaną przez obydwa punkty końcowe) TCP jest protokołem znacznie bardziej skomplikowanym niż UDP (patrz rozdział 10.). UDP jest protokołem **bezpołączeniowym**, który nie wymaga żadnego ustanawiania lub kończenia połączenia. Jedną z głównych różnic, które dostrzegamy między tymi dwoma protokołami, jest ilość szczegółów wymaganych do właściwej obsługi różnych stanów protokołu TCP i występujących, kiedy połączenia są tworzone, kończone w normalny sposób lub resetowane bez ostrzeżenia. W innych rozdziałach zobaczymy, co się dzieje, kiedy połączenie jest już ustanowione i są przesyłane dane.

Podczas ustanawiania połączenia między obydwoma punktami końcowymi następuje wymiana kilku **opcji** odnoszących się do parametrów połączenia. Niektóre opcje mogą zostać wysłane tylko wtedy, gdy połączenie jest ustanawiane, a inne mogą być wysyłane później. Pamiętamy z rozdziału 12., że nagłówek TCP ma ograniczoną przestrzeń na przechowywanie opcji (40 bajtów).

13.2. Ustanawianie i kończenie połączenia TCP

Połączenie TCP jest zdefiniowane jako 4-członowy identyfikator składający się z dwóch adresów IP i dwóch numerów portów. Mówiąc bardziej precyzyjnie, jest to para **punktów końcowych**, czyli **gniazd**, z których każde jest identyfikowane przez parę adres IP-numer portu.

Połączenie zazwyczaj przechodzi przez trzy fazy, czyli konfigurację, przesyłanie danych (stan ESTABLISHED — połączenie ustanowione) i likwidację (zamykanie). Jak zobaczymy, jedną z trudności w tworzeniu solidnej implementacji protokołu TCP jest poradzenie sobie z odpowiednią obsługą wszystkich przejść między poszczególnymi fazami. Typowy przebieg ustanowienia połączenia TCP i jego zamknięcia (bez żadnego transferu danych) został pokazany na rysunku 13.1.



Rysunek 13.1. *Normalne ustanowienie i zakończenie połączenia TCP. Zwykle klient inicjuje uzgadnianie trójetapowe w celu wymiany początkowych numerów sekwencyjnych klienta (ISN(c)) i serwera (ISN(s)) przesyłanych w segmentach SYN (skrót Seq oznacza pole numeru sekwencyjnego w nagłówku segmentu). Połączenie zostaje zakończone, kiedy każda strona wyśle segment FIN i otrzyma dla niego potwierdzenie*

Na powyższym rysunku ukazujemy chronologicznie to, co się dzieje podczas ustanawiania połączenia. Przy ustanawianiu połączenia zachodzą zwykle następujące zdarzenia.

1. **Aktywna strona otwarcia** (zwykle zwana klientem) wysyła segment SYN (tzn. pakiet TCP/IP z ustawionym bitem SYN w nagłówku TCP), podając numer portu strony, z którą chce się połączyć, oraz początkowy numer sekwencyjny klienta, czyli **ISN(c)** (patrz punkt 13.2.3). W tym momencie klient przesyła zazwyczaj jedną lub więcej opcji (patrz podrozdział 13.3). To jest segment numer 1.
2. Serwer odpowiada, wysyłając swój własny segment SYN zawierający początkowy numer sekwencyjny serwera (**ISN(s)**). To segment numer 2. Serwer potwierdza także segment SYN klienta, przekazując wartość ISN(c) plus 1 w polu *Numer ACK*. Segment SYN zużywa jedną wartość numeru sekwencyjnego i jest retransmitowany, jeśli zostanie utracony.
3. Klient musi potwierdzić otrzymanie segmentu SYN od serwera przez przekazanie wartości ISN(s) plus 1 w polu *Numer ACK*. Jest to segment numer 3.

Te trzy segmenty wypełniają procedurę ustanowienia połączenia. Jest ona często nazywana **uzgadnianiem trójetapowym** (*three-way handshake*). Jej główne cele to poinformowanie każdej ze stron o rozpoczęciu połączenia i o pewnych specjalnych szczegółach przekazywanych poprzez opcje oraz wymiana numerów ISN.

O stronie, która wysyła pierwszy segment SYN, mówi się, że wykonuje **otwarcie aktywne** (*active open*). Jak już zaznaczaliśmy, jest to zazwyczaj klient. Druga strona, która odbiera ten segment SYN i wysyła następny segment SYN, wykonuje **otwarcie pasywne** (*passive open*). Jest najczęściej nazywana serwerem. W punkcie 13.2.2 opisujemy obsługiwane przez protokół, ale rzadko występujące **otwarcie jednoczesne** (*simultaneous open*), przy którym obie strony mogą wykonać otwarcie aktywne w tym samym czasie i stać się zarówno klientem, jak i serwerem.

TCP udostępnia możliwość przesyłania danych aplikacji w segmentach SYN. Jest to jednak rzadko wykorzystywane, ponieważ nie jest obsługiwane przez interfejs API gniazd Berkeley.

Na rysunku 13.1 pokazujemy również, w jaki sposób połączenie TCP jest zamykane (usuwane lub kończone). Każdy koniec połączenia może zainicjować operację zamknięcia, obsługiwane są także równoczesne zamknięcia, choć występują rzadko. Tradycyjnie to najczęściej klient inicjuje zamknięcie (co pokazujemy na rysunku 13.1). Jednakże są serwery (np. serwery WWW), które inicjują zamknięcie po wykonaniu żądania klienta. Zwykle operacja zamknięcia rozpoczyna się, kiedy aplikacja zgłasza chęć zakończenia swojego połączenia (np. przez wywołanie funkcji systemowej `close()`). Wykonujący zamknięcie protokół TCP inicjuje tę operację przez wysłanie **segmentu FIN** (tzn. segmentu TCP z ustawionym bitem FIN). Całkowita operacja zamknięcia zajdzie, kiedy obie strony połączenia ją wykonają.

- Strona wykonująca **zamknięcie aktywne** (*active closer*) wysyła segment FIN z bieżącym numerem sekwencyjnym oczekiwanym przez odbiorcę (*K* na rysunku 13.1). Segment FIN zawiera również potwierdzenie ACK dla ostatniej partii danych przesłanych w przeciwnym kierunku (oznaczone literą *L* na rysunku 13.1).
- Strona wykonująca **zamknięcie pasywne** (*passive closer*) odpowiada, wysyłając w polu *Numer potwierdzenia* wartość $K + 1$, by zasygnalizować skuteczne odebranie segmentu FIN wysłanego przez stronę aktywną. W tym momencie aplikacja strony pasywnej jest powiadamiana, że druga strona połączenia wykonała zamknięcie. Zazwyczaj powoduje to zainicjowanie przez tę aplikację swojej własnej operacji zamknięcia. Strona pasywna staje się wtedy w gruncie rzeczy drugą stroną aktywną i wysyła swój własny segment FIN z numerem sekwencyjnym równym *L*.
- By dopełnić zamknięcie, ostatni segment wymiany zawiera potwierdzenie ACK dla ostatniego segmentu FIN. Zwróćmy uwagę, że jeśli segment FIN zostanie zgubiony, jest retransmitowany, aż do skutecznego otrzymania potwierdzenia ACK.

Do ustanowienia połączenia trzeba przesłać trzy segmenty, ale jego zakończenie wymaga aż czterech. Jest również możliwe, że połączenie znajdzie się w stanie częściowo otwartym (patrz punkt 13.6.3), chociaż nie występuje to często. Wynika to z tego, że model transmisji danych protokołu TCP jest dwukierunkowy, a to oznacza dopuszczenie możliwości, że tylko jeden z dwóch kierunków transmisji jest aktywny. Operacja **częściowego**

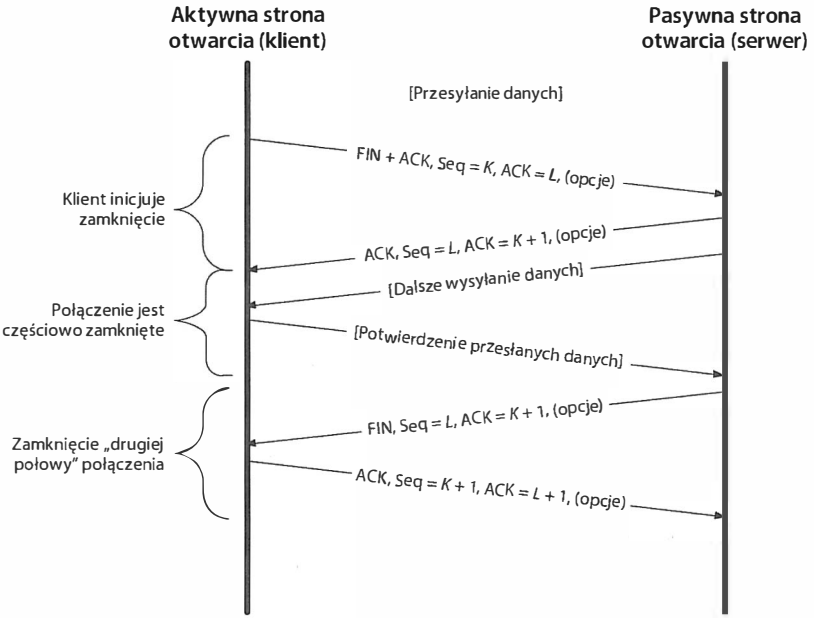
zamknięcia (*half-close*) w TCP zamyka tylko pojedynczy kierunek przepływu danych. Dwie operacje częściowego zamknięcia łącznie powodują zamknięcie całego połączenia. Obowiązuje reguła, że każda strona połączenia może wysłać segment FIN, kiedy zakończy przesyłać dane. Kiedy TCP otrzyma segment FIN, musi powiadomić właściwą aplikację o tym, że druga strona zakończyła przesyłanie danych w danym kierunku. Wysłanie własnego segmentu FIN jest zwykle wynikiem wykonania operacji zamknięcia przez obsługiwaną aplikację, co zazwyczaj powoduje ostateczne zamknięcie obu kierunków transmisji.

Te siedem segmentów, które omówiliśmy, stanowi podstawowy dodatkowy koszt dla każdego połączenia, które jest ustanawiane i usuwane w sposób „łagodny" (*gracefully*). Istnieją bardziej gwałtowne sposoby przerywania połączenia TCP przy użyciu specjalnych segmentów z żądaniem resetowania, które omówimy później. Widzimy teraz, dlaczego niektóre aplikacje wolą korzystać z protokołu UDP, z jego zdolnością do wysyłania i odbierania danych bez nawiązywania połączeń, w sytuacji gdy trzeba wymienić niewielką ilość danych. Jednak takie aplikacje muszą sobie same radzić w kwestii mechanizmów naprawiania błędów, kontroli przeciążeń i sterowania przepływem.

13.2.1. Częściowe zamknięcie połączenia TCP

Jak już wspomnieliśmy, TCP obsługuje operację częściowego zamknięcia. Niewiele aplikacji wymaga tej możliwości, toteż nie spotkamy jej zbyt często. Aby aplikacja mogła skorzystać z tej opcji, interfejs API musi dostarczyć jej sposobu na wyrażenie czegoś takiego: „Skończyłam wysyłanie danych, więc prześlij drugiej stronie segment FIN, ale w dalszym ciągu chcę otrzymywać od niej dane, do momentu aż mi wyśle swój własny segment FIN". Interfejs API gniazd Berkeley realizuje częściowe zamknięcie, jeśli aplikacja wywoła funkcję `shutdown()` zamiast bardziej typowego wywołania funkcji `close()`. Jednak większość aplikacji zamyka oba kierunki połączenia, wywołując funkcję `close`. Na rysunku 13.2 pokazujemy przykład, w którym używane jest zamknięcie częściowe. W tym przykładzie klient po lewej stronie inicjuje zamknięcie częściowe, ale może to zrobić każda strona połączenia.

Pierwsze dwa segmenty są takie same jak przy zamknięciu regularnym: segment FIN wysłany przez inicjatora, następnie segment zawierający ACK dla segmentu FIN wysłany przez odbiorcę. Dalsze działanie różni się od tego, co pokazujemy na rysunku 13.1, ponieważ strona, która odebrała sygnał częściowego zakończenia połączenia, może nadal wysyłać dane. Pokazujemy tylko jeden segment danych, po którym następuje segment ACK, ale w rzeczywistości może zostać przesłana dowolna liczba segmentów. (Więcej o wymianie segmentów danych i potwierdzeń piszemy w rozdziale 15.). Kiedy odbiorca sygnału częściowego zamknięcia zakończy wysyłanie danych, zamknie połączenie ze swojej strony, powodując wysłanie segmentu FIN i w konsekwencji dostarczenie znacznika końca pliku do aplikacji, która zainicjowała zamknięcie częściowe. Kiedy ten drugi segment FIN zostanie potwierdzony, połączenie zostanie zamknięte całkowicie.

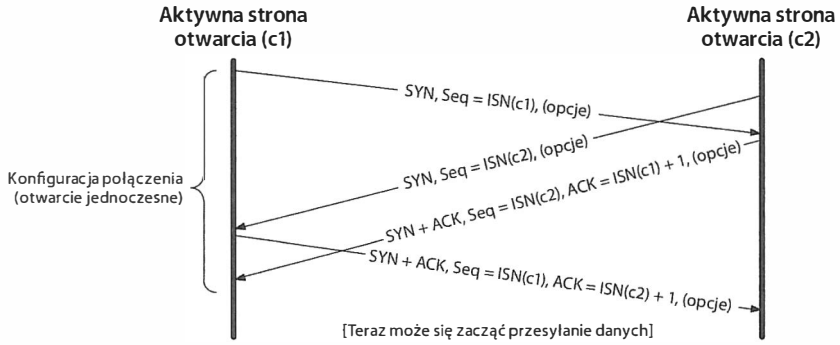


Rysunek 13.2. *W operacji częściowego zamknięcia połączenia TCP połączenie w jednym z kierunków zostaje zakończone, podczas gdy połączenie w drugim kierunku jest kontynuowane, dopóki nie zostanie zamknięte przez drugą stronę po przesłaniu pozostałych danych i otrzymaniu potwierdzenia. W niewielu aplikacjach używa się tej opcji*

13.2.2. Jednoczesne otwarcie i jednoczesne zamknięcie

Jest możliwe, choć wielce nieprawdopodobne, z wyjątkiem sytuacji specjalnie zaaranżowanych, że dwie aplikacje wykonają wzajemne aktywne otwarcie w tym samym czasie. Obie strony musiałyby wysłać segment SYN przed otrzymaniem segmentu SYN od drugiej strony, czyli segmenty musiałyby się minąć w sieci. Scenariusz ten wymaga również, aby każda strona posiadała adres IP i numer portu, które są znane drugiej stronie, co zdarza się rzadko (z wyjątkiem zastosowania technik „wybijania dziur" — *hole-punching* — w zaporze sieciowej, które poznaliśmy w rozdziale 7.). Jeśli coś takiego się zdarzy, nazywamy to **otwarciem jednoczesnym** (*simultaneous open*).

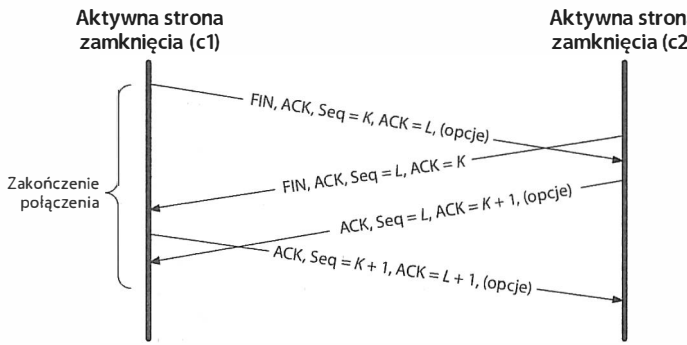
Przykładowo otwarcie jednoczesne zachodzi, kiedy aplikacja na hoście A, używająca lokalnego portu 7777, wykonuje otwarcie aktywne adresowane do portu 8888 na hoście B, a w tym samym czasie aplikacja na hoście B, wykorzystując lokalny port 8888, wykonuje aktywne otwarcie z docelowym portem 7777 na hoście A. To **nie jest to samo**, co łączenie się klienta na hoście A z serwerem na hoście B w sytuacji, gdy w tym samym czasie klient na hoście B łączy się z konwencjonalnym serwerem na hoście A. W takim przypadku oba serwery wykonują otwarcia pasywne, a nie otwarcia aktywne, a klienty przydzielają sobie inne, tzw. efemeryczne numery portów. Skutkuje to nawiązaniem dwóch oddzielnych połączeń TCP. Na rysunku 13.3 pokazujemy segmenty wymieniane podczas otwarcia jednoczesnego.



Rysunek 13.3. *Segmenty wymieniane podczas otwarcia jednoczesnego. W stosunku do zwykłej procedury ustanawiania połączenia wymagany jest jeden dodatkowy segment. Bit SYN jest włączony w każdym przesyłanym segmencie, dopóki nie zostanie odebrane dla niego potwierdzenie ACK*

Otwarcie jednoczesne wymaga wymiany czterech segmentów, jednego więcej niż przy normalnym uzgadnianiu trójetapowym. Zauważmy także, że nie nazywamy żadnej strony klientem czy serwerem, ponieważ oba końce funkcjonują i jako klient, i jako serwer. **Zamknięcie jednoczesne** nie różni się zbytnio. Powiedzieliśmy wcześniej, że jedna strona (często, ale nie zawsze, klient) wykonuje zamknięcie aktywne, powodując wysłanie pierwszego segmentu `FIN`. W przypadku zamknięcia jednoczesnego zamknięcie aktywne wykonują obie strony. Na rysunku 13.4 pokazujemy segmenty wymieniane podczas zamknięcia jednoczesnego.

Rysunek 13.4. *Segmenty wymieniane podczas zamknięcia jednoczesnego spełniają podobne funkcje, jak przy zamknięciu konwencjonalnym, ale przeplatają się pod względem kolejności*



W zamknięciu jednoczesnym wymieniana jest taka sama liczba segmentów jak przy normalnym zamknięciu. Jedyną różnicą jest to, że sekwencyjność przesyłania segmentów ustępuje ich przeplataniu. Zobaczymy później, że operacje jednoczesnego otwarcia i jednoczesnego zamknięcia używają szczególnych stanów w implementacji protokołu TCP, które nie są często wykorzystywane.

13.2.3. Początkowy numer sekwencyjny (ISN)

Kiedy połączenie jest już otwarte, każdy segment z odpowiednimi dwoma adresami IP i numerami portów jest akceptowany jako prawidłowy, o ile zawiera prawidłowy numer sekwencyjny (tzn. mieszczący się w oknie odbiorczym) i suma kontrolna się zgadza. Rodzi to pytanie, czy jest możliwa sytuacja, w której pewne, wędrujące przez sieć segmenty docierają do odbiorcy z opóźnieniem i dezorganizują połączenie. Odpowiedzią na tę obawę jest staranny wybór wartości ISN, który teraz przeanalizujemy.

Zanim każda ze stron wyśle swój segment SYN, by ustanowić połączenie, najpierw wybierze wartość ISN dla tego połączenia. Wartość ISN powinna być zmienna w czasie, tak aby dla każdego połączenia była inna. Dokument [RFC0793] stanowi, że numer ISN powinien być traktowany jako 32-bitowy licznik, który zwiększa się o 1 co 4 μs. Celem zastosowania tej reguły jest zapewnienie, aby zakres wartości numerów sekwencyjnych segmentów z jednego połączenia nie zachodził na zakres wartości numerów sekwencyjnych używanych w kolejnym (nowym) identycznym połączeniu. Szczególnie nie można dopuścić do pokrywania się numerów sekwencyjnych w różnych **instancjach** (*instantiations*), czyli **wcieleniach** (*incarnations*) **tego samego** połączenia.

Koncepcja różnych instancji tego samego połączenia staje się jasna, jeśli przypomnimy sobie, że połączenie TCP jest identyfikowane za pomocą pary punktów końcowych, tworzących czwórkę złożoną z dwóch par adres-port. Jeśli jeden z segmentów jakiegoś połączenia miał opóźnienie trwające przez długi okres czasu, a połączenie zostało w międzyczasie zamknięte i otwarte ponownie z tym samym 4-członowym identyfikatorem, można sobie wyobrazić sytuację, w której opóźniony segment pojawia się w strumieniu danych nowego połączenia i zastępuje prawidłowe dane. Byłoby to w najwyższym stopniu kłopotliwe. Podejmując odpowiednie kroki mające na celu uniknięcie powtarzania się numerów sekwencyjnych segmentów pochodzących z różnych instancji tego samego połączenia, możemy starać się zminimalizować to ryzyko. Stąd jednak wynika sugestia, że w aplikacji szczególnie wymagającej pod względem integralności danych należy zastosować swoje własne kody CRC lub sumy kontrolne na poziomie warstwy aplikacji, by mieć pewność, że jej własne dane zostały przekazane bez błędu. Jest to generalnie dobra praktyka w każdym przypadku, a powszechnie stosowana dla dużych plików.

Jak się przekonamy, znajomość 4-członowego identyfikatora połączenia oraz aktualnego okna numerów sekwencyjnych to wszystko, co jest wymagane, by utworzyć segment TCP, który będzie uznany za prawidłowy przez punkt końcowy połączenia TCP. Stanowi to pewnego rodzaju lukę bezpieczeństwa w protokole TCP: każdy może podrobić segment TCP i, dobierając odpowiednio numery sekwencyjne, adresy IP oraz numery portów, zakłócić (lub przerwać) połączenie TCP [RFC5961]. Jednym ze sposobów przeciwdziałania temu zagrożeniu jest spowodowanie, że początkowy numer sekwencyjny (lub efemeryczny numer portu — patrz [RFC6056]) będzie stosunkowo trudny do odgadnięcia. Innym sposobem jest szyfrowanie (patrz rozdział 18.).

We współczesnych systemach wartość ISN jest zazwyczaj wybierana w sposób półlosowy. Interesująca analiza subtelności związanych z jej właściwym wyborem jest zawarta w dokumencie CERT Advisory CA-2001-09 (patrz [CERTISN]). W Linuksie używa się dość skomplikowanej procedury przy wyborze wartości ISN. Wykorzystywany jest schemat oparty na zegarze, ale uruchamia się zegar z losowym przesunięciem (*offset*)

dla każdego połączenia. Losowe przesunięcie jest wyznaczane przez zastosowanie kryptograficznej funkcji skrótu do 4-członowego identyfikatora połączenia. Sekretna wartość wejściowa do funkcji skrótu zmienia się co 5 minut. Z 32 bitów numeru ISN 8 najwyższych bitów stanowi numer kolejny sekretnej wartości, a pozostałe bity są generowane przez funkcję skrótu. W ten sposób powstaje numer ISN, który jest trudny do odgadnięcia i który ponadto zwiększa swoją wartość wraz z upływem czasu. W Windows używa się ponoć podobnej procedury opartej na kryptosystemie RC4 [S96].

13.2.4. Przykład

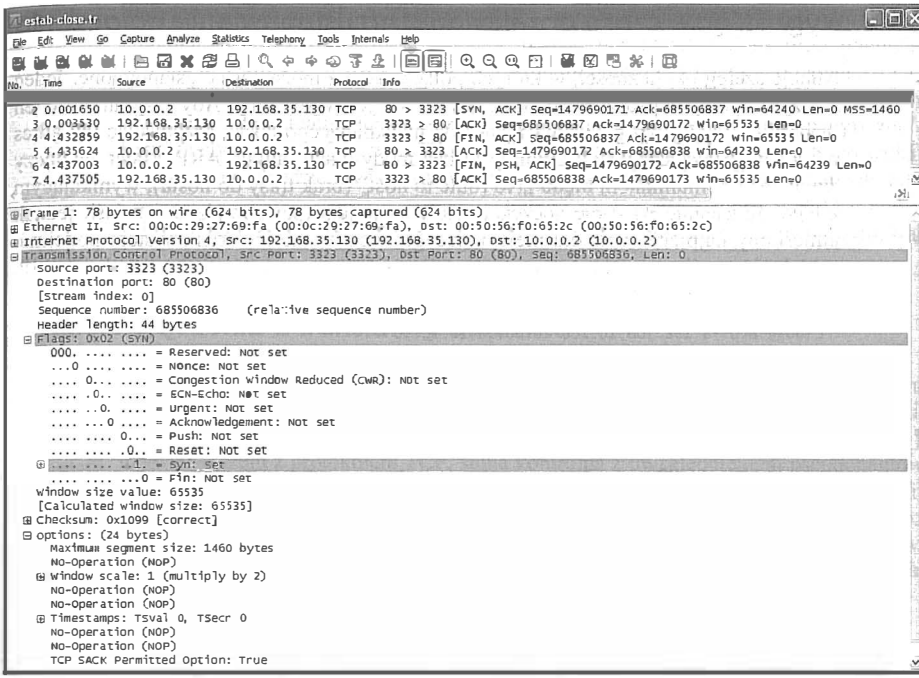
Teraz, kiedy mamy już podstawowe wyobrażenie o tym, jak połączenie TCP jest ustanawiane i usuwane, przypatrzymy się szczegółom na poziomie pakietu. W tym celu ustanawiamy połączenie TCP z jednym z pobliskich serwerów WWW, który jest uruchomiony na komputerze o adresie IPv4 10.0.0.2. Klientem jest aplikacja Telnet w systemie Windows:

```
C:\> telnet 10.0.0.2 80
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL+]'
... odczekujemy ok. 4.4 sekund...
Microsoft Telnet> quit
```

Polecenie telnet ustanawia połączenie TCP z hostem o adresie IPv4 10.0.0.2 na porcie właściwym dla http, czyli usługi WWW (port 80). Kiedy program Telnet łączy się z portem innym niż 23 (dobrze znany numer portu dla protokołu Telnet; patrz [RFC0854]), nie rozpoczyna działania zgodnego ze swoim protokołem. Zamiast tego jedynie kopiuje bajty ze swego wejścia do połączenia TCP i na odwrót. Kiedy serwer WWW odbiera przychodzące żądanie ustanowienia połączenia, pierwszą rzeczą, którą robi, jest oczekiwanie na żądanie przesłania strony WWW. W tym przypadku nie wysyłamy takiego żądania, więc serwer nie przesyła żadnych danych. Jest to dla nas sytuacja idealna, ponieważ na razie jesteśmy zainteresowani tylko wymianą pakietów związanych z ustanowieniem i zakończeniem połączenia. Na rysunku 13.5 pokazujemy wyświetlone przez program Wireshark dane o segmentach wygenerowanych poleceniem telnet.

Ze zrzutu ekranu widać, że klient rozpoczyna od wysłania segmentu SYN zawierającego numer ISN o wartości 685506836 oraz propozycję okna równą 65535. Segment ten zawiera również kilka opcji, które omówimy w podrozdziale 13.3. Drugi segment jest zarazem segmentem SYN serwera i segmentem ACK dla klienta. Numer sekwencyjny (ISN serwera) jest równy 1479690171, a numer ACK ma wartość 685506837, o 1 większą niż ISN klienta. To świadczy o poprawnym odebraniu numeru ISN klienta. Segment ten zawiera także propozycję okna wskazującą, że serwer jest gotów przyjąć do 64 240 bajtów danych. Zakończenie uzgadniania trójetapowego ma miejsce wraz z wysłaniem segmentu numer 3, który zawiera numer ACK o wartości 1479690172. Pamiętajmy, że numery ACK mają zbiorczy (kumulatywny) charakter i zawsze wskazują, jaki numer sekwencyjny jest oczekiwany przez nadawcę ACK jako **następny** (nie jest to numer sekwencyjny ostatnio odebranego segmentu).

Po przerwie wynoszącej ok. 4,4 s aplikacja Telnet otrzymuje polecenie zamknięcia połączenia. Skutkuje to wysłaniem przez TCP klienta sygnału FIN w segmencie numer 4.



Rysunek 13.5. *Połączenie TCP między hostami o adresach 192.168.35.130 i 10.0.0.2 zostało ustanowione i usunięte bez wysyłania żadnych danych. Bit PSH (push) wskazuje, że segment numer 6 zawiera wszystkie dane z bufora nadawcy (w tym przypadku nie ma żadnych danych)*

Numer sekwencyjny segmentu FIN wynosi 685506837, co jest potwierdzone przez numer ACK w segmencie numer 5 (który ma wartość 685506838). Wkrótce potem serwer wysyła swój własny segment FIN z numerem sekwencyjnym 1479690172. Ten segment potwierdza jeszcze raz (redundancyjnie) segment FIN klienta, ustawiając flagę i numer ACK. Zauważmy, że jest włączony bit PSH. Nie ma to realnego wpływu na zamykanie połączenia, ale zwykle pokazuje, że serwer nie ma żadnych dodatkowych danych do przesłania. Ostatni z przesyłanych segmentów przekazuje potwierdzenie ACK dla segmentu FIN serwera, z numerem ACK wynoszącym 1479690173.

Dokument [RFC1025] nazywa segment z maksymalną liczbą włączonych funkcji (tzn. flag i opcji) pakietem „kamikaze". Inne barwne określenia to „obelgogram" (*nastygram*), „pakiet pod choinkę" i „segment kontroli świateł" (*lamp test segment*).

Jak możemy zauważyć na rysunku 13.5, segmenty SYN przesyłają jedną lub więcej opcji. Opcje zajmują dodatkowy obszar w nagłówku TCP. Przykładowo długość pierwszego nagłówka TCP wynosi 44 bajty, czyli o 24 bajty więcej niż minimalny rozmiar nagłówka. Protokół TCP obsługuje szereg opcji, które wyszczególnimy po zobaczeniu, co się dzieje, kiedy połączenie nie może być ustanowione.

13.2.5. Wygaśnięcie czasu oczekiwania na ustanowienie połączenia

Istnieje szereg okoliczności, w których połączenie nie może zostać ustanowione. Jeden z oczywistych przypadków zachodzi, gdy serwer jest wyłączony. Aby zasymulować ten scenariusz, wydajemy polecenie `telnet` do nieistniejącego hosta, przypisując mu adres w tej samej podsieci. Jeżeli zrobimy to bez modyfikacji tablicy ARP, klient zakończy działanie z komunikatem błędu „No route to host" (brak trasy do hosta), wynikającym z tego, że żądanie ARP nie doczekało się odpowiedzi (patrz rozdział 4.). Jeśli jednak umieścimy najpierw wpis dla nieistniejącego hosta w tablicy ARP, system podejmie natychmiast próbę kontaktu z tym nieistniejącym hostem przy użyciu protokołu TCP/IP. Najpierw wprowadźmy polecenia:

```
Linux# arp -s 192.168.10.180 00:00:1a:1b:1c:1d
Linux% date; telnet 192.168.10.180 80; date
Tue June 7 21:16:34 PDT 2009
Trying 192.168.10.180...
telnet: connect to address 192.168.10.180: Connection timed out
Tue June 7 21:19:43 PDT 2009
Linux%
```

Użyty w przykładzie adres MAC (`00:00:1a:1b:1c:1d`) jest dowolnie wybranym adresem MAC niewystępującym w prezentowanej sieci LAN; nie ma to szczególnych konsekwencji. Przekroczenie czasu oczekiwania następuje po ok. 3,2 min od wprowadzenia polecenia `telnet`. Ponieważ nie ma hosta, który mógłby odpowiadać, wszystkie wygenerowane segmenty pochodzą od klienta. Na listingu 13.1 pokazujemy dane wyjściowe wyświetlane przez program Wireshark w trybie podsumowania pakietu (tekstowym).

Listing 13.1. *Dane wyjściowe programu Wireshark ilustrujące próby ustanowienia połączenia, które kończą się niepowodzeniem z powodu przeterminowania*

```
No. Time      Source          Destination     Protocol  Info
 1  0.000000  192.168.10.144  192.168.10.180  TCP       32787 > http
 2  2.997928  192.168.10.144  192.168.10.180  TCP       32787 > http
 3  8.997962  192.168.10.144  192.168.10.180  TCP       32787 > http
 4  20.997942 192.168.10.144  192.168.10.180  TCP       32787 > http
 5  44.997936 192.168.10.144  192.168.10.180  TCP       32787 > http
 6  92.997937 192.168.10.144  192.168.10.180  TCP       32787 > http
```

Interesującym aspektem zaprezentowanych danych jest informacja o tym, jak często klient TCP wysyła segment `SYN` w kolejnych próbach ustanowienia połączenia. Drugi segment zostaje wysłany 3 s po pierwszym, trzeci 6 s po drugim, czwarty jest wysłany 12 s po trzecim itd. Ten sposób postępowania nazywany jest **odczekiwaniem wykładniczym** (*exponential backoff*), a spotkaliśmy się z czymś podobnym, omawiając zachowanie ethernetowego protokołu kontroli dostępu do nośnika CSMA/CD (patrz rozdział 3.). Jednak w tamtym przypadku algorytm był nieco inny — tu mamy do czynienia z deterministycznym ustalaniem każdego kolejnego czasu odczekiwania przez podwojenie poprzedniego czasu odczekiwania, podczas gdy w przypadku Ethernetu podwojenie dotyczyło **maksymalnego** czasu odczekiwania, a rzeczywisty czas był wybierany losowo.

Ilość prób ponowienia przesłania inicjującego segmentu `SYN` może w niektórych systemach podlegać konfiguracji i zazwyczaj otrzymuje dość małą wartość, np. 5. W Linuksie zmienna konfiguracyjna systemu, czyli `net.ipv4.tcp_syn_retries`, określa maksymalną

liczbę ponowień przesłania segmentu SYN w czasie otwarcia aktywnego. Analogiczna zmienna, net.ipv4.tcp_synack_retries, podaje maksymalną ilość prób ponowienia przesłania segmentu SYN+ACK (tzn. z ustawionymi bitami SYN i ACK) w odpowiedzi na żądanie aktywnego otwarcia ze strony partnera. Opisywany parametr może być również określany indywidualnie dla poszczególnych połączeń przez ustawienie specyficznej dla Linuksa opcji gniazda o nazwie TCP_SYNCNT. Jej wartość domyślna wynosi 5, co widzimy w naszym przykładzie. Czasy przerw między retransmisjami wynikające z reguły odczekiwania wykładniczego stanowią część strategii używanej przez protokół TCP przy obsłudze przeciążeń. Zbadamy ją szczegółowo, gdy będziemy analizować algorytm Karna (patrz rozdział 16.).

13.2.6. Połączenia a translatory adresów

W rozdziale 7. analizowaliśmy, jak konwencjonalny NAT przekształca adresy i numery portów używane przez protokoły, takie jak TCP i UDP. Badaliśmy również, jak pakiety IP mogą być tłumaczone między standardami IPv6 i IPv4. Kiedy mechanizm NAT jest używany razem z protokołem TCP, suma kontrolna z uwzględnieniem pseudonagłówka wymaga zwykle skorygowania (z wyjątkiem przypadków, gdy używany jest modyfikator adresów neutralny względem sumy kontrolnej). Dotyczy to również innych protokołów, w których używa się sum kontrolnych obliczanych z włączeniem pseudonagłówka, ponieważ obliczenia te muszą uwzględnić informacje zarówno z poziomu warstwy transportowej, jak i warstwy sieciowej.

Kiedy połączenie TCP jest ustanawiane, system NAT (lub NAT64) może wykryć ten fakt dzięki obecności ustawionego bitu SYN w segmencie. Może on również ustalić, kiedy połączenie zostało w pełni ustanowione, szukając kolejnych segmentów SYN+ACK i ACK, zawierających odpowiednie numery sekwencyjne. To samo odnosi się do zakończenia połączenia. Po zaimplementowaniu części automatu stanów protokołu TCP w technologii NAT (patrz np. sekcje 3.5.2.1 i 3.5.2.2 dokumentu [RFC6146]) połączenie może być śledzone łącznie z bieżącymi stanami, numerami sekwencyjnymi dla każdego kierunku i odpowiadającymi im numerami ACK. Takie śledzenie stanów jest typowe dla implementacji NAT.

Dalsze komplikacje powstają, kiedy NAT działa jak edytor i zmienia zawartość pola danych użytkowych protokołu transportowego. W TCP może się to wiązać z usuwaniem lub dodawaniem bajtów do strumienia danych i w konsekwencji rzutować na długość segmentów i wartości numerów sekwencyjnych. Takie działanie wpływa siłą rzeczy na sumę kontrolną, ale także na sekwencjonowanie danych. Jeżeli dane są wstawiane lub usuwane ze strumienia przez NAT, odpowiednie wartości mogą zostać skorygowane. Takie postępowanie jest jednak nieco zawodne — jeśli NAT utraci synchronizację swojego stanu ze stanem przechowywanym w hostach końcowych, połączenie nie będzie działać poprawnie.

13.3. Opcje TCP

Nagłówek TCP może zawierać opcje (patrz rysunek 12.3). Jedynymi opcjami zdefiniowanymi w oryginalnej specyfikacji protokołu TCP są: *Koniec listy opcji* (*End of Option List*, czyli ***EOL***), *Brak akcji* (*No Operation*, czyli ***NOP***) i *Maksymalny rozmiar segmentu*

(*Maximum Segment Size*, czyli ***MSS***). Od tego czasu zdefiniowano szereg nowych opcji. Pełna lista jest utrzymywana przez organizację IANA (patrz [TPARAMS]); w tabeli 13.1 przedstawiamy aktualne opcje warte uwagi (tzn. te ze standardowej ścieżki, opisane w dokumentach RFC).

Tabela 13.1. *Opcje nagłówka TCP. Na przechowywanie opcji udostępniony jest obszar do 40 bajtów*

Rodzaj	Długość	Nazwa	Pełna nazwa (angielska)	Źródło	Opis i zastosowanie
0	1	EOL	*End of Option List*	[RFC0793]	Koniec listy opcji
1	1	NOP	*No Operation*	[RFC0793]	Brak akcji (używana w roli wypełniacza)
2	4	MSS	*Maximum Segment Size*	[RFC0793]	Maksymalny rozmiar segmentu
3	3	WSOPT	*Window Scale Option*	[RFC1323]	Czynnik skalujący okna (o ile bitów należy przesunąć w lewo zawartość pola *Rozmiar okna*)
4	2	SACK-Permitted		[RFC2018]	Nadawca obsługuje opcje SACK
5	Zmienna	SACK	*Selective Acknowledgement*	[RFC2018]	Blok SACK (otrzymano dane poza kolejnością)
8	10	TSOPT	*Timestamps Option*	[RFC1323]	Opcja znaczników czasu
28	4	UTO	*User Timeout Option*	[RFC5482]	Czas oczekiwania użytkownika (awaryjne zakończenie po zadanym czasie bezczynności)
29	Zmienna	TCP-AO	*TCP Authentication Option*	[RFC5925]	Opcja uwierzytelniania (przy użyciu różnych algorytmów)
253	Zmienna	Ekspery-mentalna		[RFC4727]	Zarezerwowana do użytku eksperymentalnego
254	Zmienna	Ekspery-mentalna		[RFC4727]	Zarezerwowana do użytku eksperymentalnego

Każdą opcję rozpoczyna 1-bajtowy *rodzaj* (*kind*), który określa typ opcji. Opcje, które nie są rozpoznawane, są po prostu ignorowane, zgodnie z dokumentem [RFC1122]. Opcje, których rodzaj ma wartość 0 lub 1, zajmują pojedynczy bajt. W pozostałych opcjach za bajtem *rodzaju* występuje bajt określający *długość*, która jest długością całkowitą, z uwzględnieniem bajtów *rodzaj* i *długość*. Powodem istnienia opcji *NOP* jest umożliwienie nadawcy dopełnienia pól do najbliższej wielokrotności 4 bajtów, jeśli tak będzie trzeba. Pamiętajmy, że wymagane jest, aby długość nagłówka TCP była zawsze równa wielokrotności 32 bitów, ponieważ taka właśnie jednostka jest używana w polu *Długość nagłówka*. Opcja EOL wskazuje koniec listy opcji i sygnalizuje, że w tym miejscu można zakończyć ich przetwarzanie. Teraz przyjrzyjmy się pozostałym opcjom.

13.3.1. Opcja maksymalnego rozmiaru segmentu (MSS)

Maksymalny rozmiar segmentu (MSS — *Maximum Segment Size*) określa rozmiar największego segmentu, jaki protokół TCP jest gotów przyjąć od swojego odpowiednika na innym hoście, a w konsekwencji maksymalny rozmiar segmentu, który może zostać użyty przez drugą stronę połączenia przy wysyłaniu danych. Wartość MSS uwzględnia tylko bajty danych i nie obejmuje rozmiarów żadnego z towarzyszących im nagłówków, TCP czy IP [RFC0879]. Kiedy połączenie jest ustanawiane, każda jego strona ogłasza swoje MSS w postaci opcji MSS przekazywanej w segmencie SYN. Opcja udostępnia 16 bitów do określenia wartości MSS. Jeśli opcja MSS nie zostanie przekazana, przyjmowana jest wartość domyślna wynosząca 536 bajtów. Przypomnijmy sobie regułę, która wymaga, aby każdy host był zdolny do przetworzenia datagramów IPv4 przynajmniej o wielkości 576 bajtów. Przy minimalnych rozmiarach nagłówków IPv4 i TCP protokół TCP wysyłający 536 bajtów danych, czyli tyle, na ile pozwala domyślna wartość MSS, tworzy datagram IPv4 o rozmiarze 20 + 20 + 536 = 576 bajtów.

Wszystkie wartości MSS na rysunku 13.5 wynoszą 1460, co jest typowe dla IPv4. Wynikający z tego rozmiar datagramu IPv4 jest zwykle o 40 bajtów większy (w sumie 1500 bajtów, typowy rozmiar MTU dla Ethernetu i MTU ścieżki dla Internetu) po uwzględnieniu 20 bajtów nagłówka TCP i 20 bajtów nagłówka IPv4. Jeżeli używany jest protokół IPv6, MSS zwykle wynosi 1440, czyli o 20 bajtów mniej, z powodu większego rozmiaru nagłówka IPv6. W jumbogramach IPv6 używana jest specjalna wartość MSS równa 65535, oznaczająca praktycznie nieskończoność [RFC2675]. W tym przypadku parametr SMSS (MSS nadawcy, *send MSS*) zostanie wyznaczony jako PMTU (MTU ścieżki, *path MTU*) minus 60 bajtów (40 bajtów nagłówka IPv6 i 20 bajtów nagłówka TCP). Zauważmy, że opcja MSS nie jest elementem negocjacji między stronami protokołu TCP; jest to ograniczenie. Kiedy jedna strona TCP przekazuję swoją opcję MSS drugiej stronie, to pokazuje, że przez cały okres trwania połączenia nie zamierza akceptować jakichkolwiek segmentów przekraczających wskazany rozmiar.

13.3.2. Opcje selektywnego potwierdzenia (SACK)

W rozdziale 12. wprowadziliśmy pojęcie okna przesuwnego i opisaliśmy, jak TCP obsługuje numery sekwencyjne i potwierdzenia. Ponieważ protokół TCP używa skumulowanych potwierdzeń ACK, nie może nigdy potwierdzić prawidłowo odebranych danych, które nie są, pod względem numerów sekwencyjnych, ciągłą kontynuacją danych odebranych poprzednio. W takich przypadkach mówi się, że odbiorca TCP posiada **luki** w kolejce odebranych danych. Odbierający protokół TCP nie pozwala aplikacjom na skonsumowanie danych znajdujących się za luką ze względu na abstrakcję strumienia danych, którą musi zapewnić.

Gdyby nadawca TCP mógł się dowiedzieć o istnieniu luk (i odebranych poza kolejnością blokach danych, poprzedzonych dziurami w przestrzeni numerów sekwencyjnych) u odbiorcy, lepiej wybrałby, które konkretne segmenty należy retransmitować, w sytuacji gdy segmenty zostały utracone lub z innego powodu nieodebrane przez odbiorcę. Opcja selektywnego potwierdzenia protokołu TCP (**SACK**, *Selective Acknowledgement*; patrz [RFC2018] [RFC2883]) oferuje taką możliwość. Rozwiązanie to działa efektywnie jednak tylko wtedy, gdy logika protokołu TCP u nadawcy jest w stanie zrobić skuteczny

użytek z informacji zawartej w opcji SACK otrzymanej od zdolnego do jej przekazania odbiorcy.

Protokół TCP hosta dowiaduje się, że jego odpowiednik na drugim hoście jest zdolny do przekazywania opcji SACK, kiedy odbierze opcję SACK-Permitted przekazaną w segmencie SYN (lub SYN+ACK). Kiedy warunek ten zostanie spełniony, strona, która odbierze dane w niewłaściwej kolejności, może przesłać opcję SACK opisującą te dane, aby pomóc partnerowi w bardziej efektywnym wykonywaniu retransmisji. Informacja zawarta w opcji SACK składa się zakresów numerów sekwencyjnych reprezentujących prawidłowo odebrane bloki danych. Każdy taki zakres nazywa się **blokiem SACK** i jest reprezentowany przez parę 32-bitowych numerów sekwencyjnych. Dlatego opcja SACK zawierająca *n* bloków SACK ma w bajtach długość (8*n*+2). Dwa dodatkowe bajty zawierają *rodzaj* i *długość* opcji SACK.

Ze względu na ograniczoną ilość miejsca dostępnego w polu opcji nagłówka TCP maksymalna liczba bloków SACK możliwych do przesłania w pojedynczym segmencie wynosi 3 (zakładając, że jest także używana opcja *Znaczniki czasu*, opisana w punkcie 13.3.4, co zazwyczaj ma miejsce we współczesnych implementacjach protokołu TCP). Chociaż opcja SACK-Permitted jest zawsze przesyłana w segmencie SYN, same bloki SACK mogą być przekazywane w dowolnym segmencie, pod warunkiem że nadawca przesłał już wcześniej opcję SACK-Permitted. Ponieważ działanie opcji SACK jest w sposób najbardziej oczywisty (i najbardziej istotny) powiązane z operacjami kontroli błędów i przeciążeń protokołu TCP, przeanalizujemy je bardziej szczegółowo, kiedy będziemy omawiać te tematy w rozdziałach 14. i 16.

13.3.3. Opcja skalowania rozmiaru okna (WSCALE lub WSOPT)

Opcja *Skalowanie okna* (oznaczana jako WSCALE lub WSOPT; patrz [RFC1323]) zwiększa efektywną pojemność pola *Rozmiar okna* z 16 do 30 bitów. Dzieje się to jednak bez zwiększania rozmiaru pola w nagłówku TCP, które wciąż przechowuje wartość 16-bitową, lecz zostaje zdefiniowana nowa opcja, która dokłada **czynnik skalujący** do tej 16-bitowej wartości. Zawartość pola *Rozmiar okna* jest przed użyciem przesuwana bitowo w lewo o wartość tego czynnika. W rezultacie wartość rozmiaru okna zostaje pomnożona przez liczbę 2^s, gdzie *s* oznacza czynnik skalujący. Jednobajtowy wyznacznik przesunięcia przyjmuje wartości od 0 do 14 (włącznie). Wartość 0 oznacza brak przesunięcia. Maksymalna wartość skalowania równa 14 umożliwia osiągnięcie maksymalnego rozmiaru okna w wysokości 1 073 725 440 bajtów (65 535 × 2^{14}), bliskiej wartości 1 073 741 823 (2^{30}–1), a więc 1 GB. Protokół TCP przechowuje wewnętrznie faktyczny rozmiar okna jako wartość 32-bitową.

Opcja WSCALE może wystąpić tylko w segmencie SYN, tak więc czynnik skalujący jest stały w ramach każdego kierunku połączenia, kiedy jest już ustanowione. Aby włączyć skalowanie okna, obie strony połączenia muszą przesłać tę opcję w swoich segmentach SYN. Strona wykonująca otwarcie aktywne zawsze może wysyłać opcję WSCALE w swoim segmencie SYN, ale strona wykonująca otwarcie pasywne może wysłać swoją opcję WSCALE tylko wówczas, gdy otrzymany segment SYN też zawiera specyfikację tej opcji. Czynnik skalujący może być inny dla każdego kierunku. Jeżeli strona wykonująca otwarcie aktywne wyśle czynnik skalujący o niezerowej wartości, ale nie otrzyma w odpowiedzi opcji *Skalowanie okna* od drugiej strony, ustawia swoje czynniki skalujące

na wartość 0 i dla nadawania, i dla odbioru. Pozwala to systemom „nierozumiejącym" tej opcji na współdziałanie z systemami, które ją obsługują.

Załóżmy, że używamy opcji *Skalowanie okna* z wartością przesunięcia równą *S* dla wysyłania i *R* dla odbioru. Wtedy każda 16-bitowa wartość propozycji okna otrzymana od drugiej strony musi zostać przesunięta w lewo o *R* bitów, aby otrzymać rzeczywistą wartość proponowanego rozmiaru okna. Z kolei zawsze wtedy, kiedy wysyłamy naszą propozycję okna do drugiej strony, bierzemy nasz rzeczywisty 32-bitowy rozmiar okna i przesuwamy go w prawo o *S* bitów, umieszczając uzyskaną w ten sposób wartość 16-bitową w nagłówku TCP.

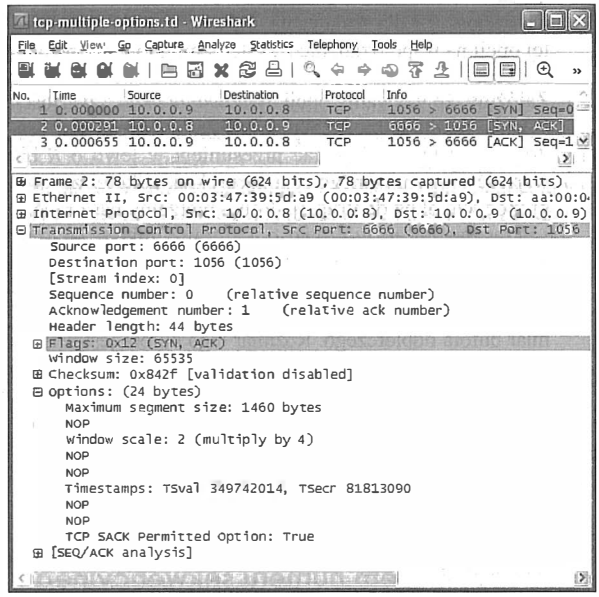
Wartość przesunięcia jest automatycznie dobierana przez protokół TCP w oparciu o rozmiar bufora odbiorczego. Rozmiar tego bufora jest ustawiany przez system, ale zazwyczaj aplikacje mają możliwość jego zmiany. Opcja *Skalowanie okna* ma największe zastosowanie wtedy, kiedy protokół TCP obsługuje transfer danych masowych poprzez sieci z dużym iloczynem przepustowości i opóźnienia (tzn. ze stosunkowo dużą wartością iloczynu przepustowości i czasu RTT; *large-bandwith-delay product*). Wobec tego znaczenie i sposób użycia tej opcji przeanalizujemy dokładniej w rozdziale 16.

13.3.4. Opcja znaczników czasu i ochrona przed przepełnieniem numeru sekwencyjnego (PAWS)

Opcja *Znaczniki czasu* (*Timestamps*), nazywana niekiedy również *Znacznikiem czasu* (*Timestamp*) i zapisywana w skrócie jako **TSOPT** lub **TSopt** pozwala nadawcy na umieszczenie dwóch 4-bajtowych wartości znacznika czasu w każdym segmencie. Odbiorca odzwierciedla te wartości w potwierdzeniu, pozwalając nadawcy na obliczenie szacunkowej wartości czasu RTT połączenia dla każdego otrzymanego ACK. (Musimy tu powiedzieć „dla każdego otrzymanego ACK", a nie „dla każdego segmentu", ponieważ TCP często potwierdza wiele segmentów pojedynczym ACK; zobaczymy to w rozdziale 15.). Używając opcji *Znaczniki czasu*, nadawca umieszcza 32-bitową wartość w polu *Wartość znacznika czasu* (TSV lub TSval, *Timestamp Value*), w pierwszej części opcji TSOPT, a odbiorca odsyła ją niezmienioną na zasadzie echa w drugim polu opcji o nazwie *Echo znacznika czasu* (TSER lub TSecr, *Timestamp Echo Reply*). Nagłówki TCP zawierające tę opcję zwiększają się o 10 bajtów (8 bajtów na dwie wartości znacznika czasu i 2 bajty wskazujące *rodzaj* i *długość* opcji).

Znacznik czasu to rosnąca monotonicznie wartość. Ponieważ odbiorca po prostu wiernie zwraca wszystko, co otrzymuje, nie dba o to, jakie jednostki i wartości są faktycznie użyte w znacznikach czasu. Opcja ta nie wymaga żadnej formy synchronizacji zegarów między obydwoma hostami. Dokument [RFC1323] zaleca, żeby nadawca zwiększał wartość znacznika czasu przynajmniej o 1 co sekundę. Na rysunku 13.6 pokazujemy opcję *Znaczniki czasu* wyświetloną w programie Wireshark.

W tym przykładzie obie strony uczestniczą w generowaniu własnych znaczników czasu i zwracaniu echa znaczników czasu drugiej strony. Pierwszy segment (SYN klienta) używa początkowej wartości znacznika czasu równej 81813090. Wartość ta jest umieszczona w polu TSV. Drugie pole, czyli TSER, ma w pierwszym segmencie wartość 0, ponieważ klient nie zna jeszcze wartości znacznika czasu serwera.



Rysunek 13.6. *Połączenie TCP używające opcji: Znaczniki czasu, Skalowanie okna i MSS. Nagłówek TCP ma 44 bajty długości. Inicjujący SYN (pakiet numer 1) rozpoczyna od wartości TSV równej 81813090. Drugi pakiet, wyróżniony, przesyła tę wartość z powrotem do aktywnej strony otwarcia, dołączając własną wartość równą 349742014*

Głównym uzasadnieniem potrzeby wyliczenia dobrego oszacowania czasu RTT połączenia jest właściwe ustawienie czasu oczekiwania na retransmisję, który jest dla protokołu TCP informacją, kiedy powinien podjąć próbę ponownego przesłania segmentu, jaki prawdopodobnie został utracony. W rozdziale 12. analizowaliśmy potrzebę określenia czasu oczekiwania w oparciu o jakąś funkcję czasu RTT. Korzystając z opcji *Znaczniki czasu*, możemy uzyskać stosunkowo precyzyjne pomiary czasu RTT. Przed wprowadzeniem opcji *Znaczniki czasu* większość implementacji TCP wykonywała tylko jedno próbkowanie czasu RTT na okno danych. Dzięki tej opcji może zostać pobranych więcej próbek, co prowadzi do potencjalnie lepszego oszacowania czasu RTT (patrz [RFC1323] i [RFC6298]).

Ponieważ opcja *Znaczniki czasu* jest najistotniejsza dla ustawiania zegara retransmisji, omówimy jej użycie do tego celu bardziej szczegółowo, kiedy będziemy opisywać retransmisję w rozdziale 14. Powiedzieliśmy „do tego celu", bo opcja znaczników czasu, oprócz tego, że umożliwia częstsze próbkowanie czasu RTT, dostarcza odbiorcy sposób na unikanie sytuacji, w których, odbierając jakieś stare segmenty, mógłby uznać je za aktualne. Mechanizm ten nazywa się **ochroną przed przepełnieniem numeru sekwencyjnego** (PAWS, *Protection Against Wrapped Sequence Numbers*) i jest opisany w dokumencie [RFC1323] razem z opcją *Znaczniki czasu*. Teraz przyjrzymy się, jak działa.

Rozważmy połączenie TCP, w którym użyto opcji *Skalowanie okna* z największym możliwym rozmiarem okna wynoszącym ok. 1 GB (czyli 2^{30} bajtów). Przyjmijmy też, że zastosowana jest opcja *Znaczniki czasu* i wartość znacznika czasu zwiększa się o 1 dla każdego wysłanego okna. (Jest to ostrożne założenie. Normalnie znacznik czasu zwiększa swą wartość szybciej). W tabeli 13.2 pokazujemy możliwy przepływ danych

Tabela 13.2. *Opcja TCP Znaczniki czasu pozwala na ujednoznacznienie segmentów z tymi samymi numerami sekwencyjnymi, dostarczając dodatkowe 32 bity efektywnej przestrzeni numerów sekwencyjnych*

Czas	Wysłane bajty	Numery sekwencyjne wysłanych danych	Znacznik czasu nadawcy	Odbiór
A	0G:1G	0G:1G	1	OK.
B	1G:2G	1G:2G	2	OK. Jednak jeden utracony segment jest retransmitowany
C	2G:3G	2G:3G	3	OK.
D	3G:4G	3G:4G	4	OK.
E	4G:5G	0G:1G	5	OK.
F	5G:6G	1G:2G	6	OK. Jednak pojawia się ponownie retransmitowany segment

między dwoma hostami podczas transferu 6 GB. Aby uniknąć mnóstwa liczb dziesięciocyfrowych, używamy notacji G do oznaczenia wielokrotności liczby 1 073 741 824. Używamy także notacji pochodzącej z programu `tcpdump`, w której *J*:*K* oznacza bajty od *J* do *K*-1 włącznie.

32-bitowe pole *Numer sekwencyjny* ulega przepełnieniu (i numery sekwencyjne powtarzają się od 0) między czasami D i E. Przyjmujemy, że jeden segment zostaje utracony w czasie B i jest retransmitowany. Zakładamy także, że segment ten pojawia się ponownie u odbiorcy w czasie F. Zakładamy ponadto, że różnica czasowa między utratą segmentu a jego ponownym pojawieniem się u odbiorcy jest mniejsza niż maksymalny czas, przez jaki segment może „żyć" w sieci (tzw. MSL; patrz punkt 13.5.2); gdyż inaczej segment zostałby odrzucony przez jakiś router po wygaśnięciu jego czasu TTL (*Time To Live*, czas życia). Jak wspomnieliśmy wcześniej, problem ten pojawia się w stosunkowo szybkich połączeniach, gdzie stare segmenty mogą pojawiać się ponownie i zawierać numery sekwencyjne, które są aktualnie transmitowane.

Analizując tabelę 13.2, możemy dostrzec, że użycie opcji *Znaczniki czasu* zapobiega temu problemowi. Odbiorca traktuje znacznik czasu jako 32-bitowe rozszerzenie numeru sekwencyjnego. Ponieważ utracony segment, który pojawia się ponownie w czasie F, posiada znacznik czasu równy 2, a więc mniejszy niż ostatnio odebrany prawidłowy znacznik czasu (5 lub 6), zostanie on odrzucony przez algorytm PAWS. Algorytm PAWS nie wymaga żadnej formy synchronizacji między nadawcą i odbiorcą. Wszystko, czego odbiorca wymaga w kwestii wartości znaczników czasu, to to, że mają rosnąć monotonicznie, zwiększając się przynajmniej o 1 dla każdego okna danych.

13.3.5. Opcja czasu oczekiwania użytkownika (UTO)

Opcja *Czas oczekiwania użytkownika* (UTO, *User Timeout*) jest stosunkowo nową możliwością protokołu TCP opisaną w dokumencie [RFC5482]. Wartość UTO (nazywana także `USER_TIMEOUT`) określa ilość czasu, przez jaką nadawca jest gotów oczekiwać

na zaległe potwierdzenie ACK dla wysłanych danych przed dojściem do wniosku, że odległy koniec połączenia już go nie dostarczy. USER_TIMEOUT jest tradycyjnym lokalnym parametrem konfiguracyjnym protokołu TCP (patrz [RFC0793]). Opcja UTO umożliwia jednej stronie protokołu TCP przekazanie swojej wartości parametru USER_TIMEOUT drugiej stronie połączenia. To pozwala odbierającemu TCP na dostosowanie swojego zachowania (np. przez tolerowanie dłuższego okresu przerwanej łączności przed anulowaniem połączenia). Urządzenia obsługujące NAT mogą również interpretować taką informację jako pomoc w ustawieniu własnych zegarów aktywności połączenia.

Wartości opcji UTO mają charakter konsultacyjny; z faktu, że jedna strona połączenia chciałaby używać dużej czy małej wartości UTO, nie wynika, że druga strona musi się do tego zastosować. Dokument [RFC1122] precyzuje definicję parametru USER_TIMEOUT, sugerując, że protokół TCP osiągający próg trzech (*R1*) retransmisji powinien o tym powiadomić obsługiwaną aplikację, a po upływie 100 s (*R2*) połączenie powinno zostać zamknięte. Niektóre aplikacje dysponują funkcją API pozwalającą na zmianę *R1* i *R2*. Ponieważ duże wartości UTO mogłyby doprowadzić do kłopotów związanych z wyczerpaniem zasobów, a krótkie czasy UTO mogłyby skutkować przedwczesnym przerywaniem niektórych połączeń (rodzaj ataku typu DoS), zostały nałożone granice, dolna i górna, na dopuszczalne wartości opcji UTO. Sposób ustalania wartości USER_TIMEOUT jest, wobec tego, następujący:

```
USER_TIMEOUT = min(U_LIMIT, max(ADV_UTO, REMOTE_UTO, L_LIMIT))
```

gdzie ADV_UTO oznacza opcję UTO przekazywaną do odległego protokołu TCP, REMOTE_UTO jest wartością opcji UTO proponowaną przez odległą stronę połączenia, U_LIMIT jest górnym ograniczeniem opcji UTO lokalnego systemu, a L_LIMIT stanowi dolną granicę dla UTO lokalnego systemu. Zauważmy, że powyższy wzór nie gwarantuje, iż każda strona tego samego połączenia otrzyma w efekcie tę samą wartość parametru USER_TIMEOUT. W każdym przypadku wartość L_LIMIT musi być większa od wartości czasu oczekiwania na retransmisję (RTO) ustalonej dla danego połączenia (patrz rozdział 14.) i zaleca się, by wynosiła 100 s w celu zachowania kompatybilności z dokumentem [RFC1122].

Opcje UTO są zawarte w segmentach SYN, kiedy połączenie jest ustanawiane, w pierwszych segmentach, które nie są segmentami SYN, i zawsze, kiedy jest zmieniana wartość parametru USER_TIMEOUT. Wartość opcji jest wyrażona w postaci 15-bitowej liczby jednostek czasu, sekund lub minut, poprzedzonej 1-bitowym polem „granularności", które wskazuje, czy jednostki czasu to minuty (1), czy sekundy (0). Jako stosunkowa nowa opcja, UTO nie jest jeszcze szeroko stosowana.

13.3.6. Opcja uwierzytelniania (TCP-AO)

Istnieje opcja używana do podniesienia bezpieczeństwa połączeń TCP. Została zaprojektowana, aby ulepszyć i zastąpić wcześniejszy mechanizm zwany TCP-MD5 (patrz [RFC2385]). Nazwana jest *Opcją uwierzytelniania TCP* (TCP-AO, *TCP Authentication Option*; patrz [RFC5925]); użyto w niej kryptograficznego algorytmu haszującego (patrz rozdział 18.) razem z sekretną wartością znaną każdej stronie połączenia do uwierzytelnienia każdego segmentu. Opcja TCP-AO jest lepsza od opcji TCP-MD5, dlatego że obsługuje wiele różnych algorytmów kryptograficznych i identyfikuje zmiany kluczy przy użyciu sygnalizacji wewnątrzpasmowej. Nie udostępnia jednak wszechstronnego rozwiązania do zarządzania kluczami, znaczy to, że każda strona musi wciąż posiadać sposób ustanowienia przed rozpoczęciem operacji wspólnego zestawu kluczy.

Wysyłając dane, protokół TCP tworzy klucz szyfrujący dane (*traffic encryption key*) na podstawie wspólnego tajnego klucza i oblicza wartość skrótu zgodnie z konkretnym algorytmem kryptograficznym (patrz [RFC5926]). Odbiorca wyposażony w ten sam tajny klucz potrafi w analogiczny sposób utworzyć klucz szyfrujący dane i użyć go w celu sprawdzenia, czy przychodzący segment nie został zmodyfikowany w drodze (z wysokim prawdopodobieństwem). Opcja TCP-AO została pomyślana jako silny środek zaradczy na różnorodne ataki typu *TCP spoofing* (próby nieuprawnionego zestawienia połączenia TCP przez podszywanie się pod zaufany system; patrz podrozdział 13.8). Ponieważ jednak wymaga utworzenia i dystrybucji wspólnego klucza (i jest stosunkowo nową opcją), nie jest jeszcze szeroko stosowana.

13.4. Odkrywanie MTU ścieżki w protokole TCP

W rozdziale 3. opisaliśmy pojęcie MTU ścieżki (*Path MTU*). Jest to najmniejsza spośród wartości MTU dla wszystkich segmentów sieci aktualnie znajdujących się w ścieżce między dwoma hostami. Znajomość MTU ścieżki może pomóc protokołom, takim jak TCP, w uniknięciu fragmentacji. W rozdziale 10. zobaczyliśmy, jak odkrywanie MTU ścieżki (PMTUD, *Path Maximum Transmission Unit Discovery*) jest realizowane w oparciu o komunikaty protokołu ICMP, chociaż opisywany w tamtym rozdziale protokół UDP nie ma zwykle możliwości dostosowania rozmiaru swojego datagramu, ponieważ to aplikacja określa ten rozmiar (a nie protokół warstwy transportowej). Protokół TCP, dostarczając implementowanej przez siebie abstrakcji strumienia bajtów, sam ustala, jakiego rozmiaru segmentu ma użyć, i w rezultacie posiada znacznie większy zakres kontroli nad rozmiarem datagramów IP, które są ostatecznie generowane.

W tym podrozdziale zbadamy, jak PMTUD jest wykorzystywane w protokole TCP. Nasza analiza będzie odnosiła się zarówno do konfiguracji TCP/IPv4, jak i TCP/IPv6. Więcej szczegółów na ten temat znaleźć można w dokumentach [RFC1191] i [RFC1981]. Zarówno TCP (patrz [RFC4821]), jak i inne protokoły transportowe, mogą skorzystać z metody, która pozwala na uniknięcie użycia protokołu ICMP, a nazywa się po angielsku *Packetization Layer Path MTU Discovery* (PLPMTUD — odkrywanie MTU ścieżki przez warstwę pakietyzacji). Będziemy używać terminu stosowanego w protokole ICMPv6, czyli PTB (*Packet Too Big* — pakiet jest za duży), zarówno dla komunikatu protokołu ICMPv4: *Destination Unreachable (Fragmentation Required)* — punkt docelowy nieosiągalny (wymagana fragmentacja), jak i dla właściwego komunikatu *Packet Too Big* protokołu ICMPv6.

Zwykły proces PMTUD protokołu TCP działa następująco: kiedy połączenie jest ustanawiane, TCP używa mniejszej z dwóch wartości, MTU interfejsu wyjściowego lub parametru MSS ogłoszonego przez drugi koniec połączenia, jako podstawy do określenia własnego maksymalnego rozmiaru segmentu nadawcy (SMSS, *send MSS*). Metoda PMTUD nie pozwala protokołowi TCP na przekroczenie wartości MSS zgłoszonej przez drugą stronę. Jeżeli druga strona nie określi MSS, nadawca przyjmuje wartość domyślną, czyli 536 bajtów, ale taka sytuacja występuje teraz rzadko. Możliwa jest także implementacja, która przechowuje informacje o MTU ścieżki dla poszczególnych punktów docelowych, aby je wykorzystać do określenia swojego rozmiaru segmentu. Zwróćmy uwagę na to, że MTU ścieżki może być inne dla każdego kierunku połączenia.

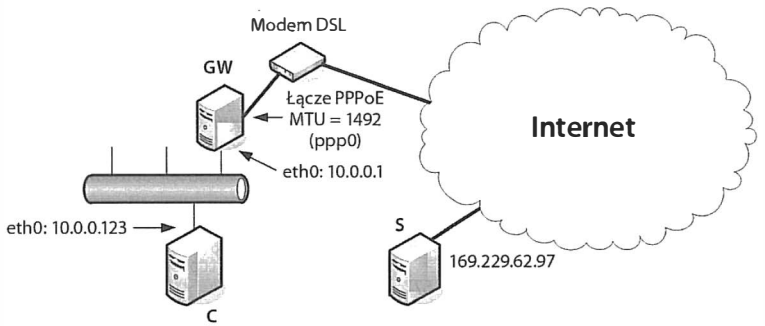
Kiedy początkowa wartość SMSS jest już wybrana, wszystkie datagramy IPv4 wysyłane przez TCP za pośrednictwem danego połączenia mają ustawiony bit DF w nagłówku IPv4. W przypadku zestawu TCP/IPv6 nie jest to wymagane, ponieważ nie ma pola bitu DF; zakłada się domyślnie, że wszystkie datagramy mają tę flagę włączoną. Jeśli zostanie odebrany komunikat PTB, protokół TCP zmniejsza rozmiar segmentu i ponawia transmisję ze zmienionym rozmiarem segmentu. Jeżeli komunikat PTB zawiera zalecenie MTU dla następnego przeskoku (*next-hop MTU*), rozmiar segmentu może być ustalony na wartość równą *next-hop MTU* pomniejszoną o sumę rozmiarów nagłówków IPv4 (lub IPv6) i TCP. Jeśli wartość MTU następnego przeskoku nie jest określona (tzn. został zwrócony komunikat PTB starszej wersji protokołu ICMP, w którym brakuje tego parametru), nadawca może próbować różnych wartości (np. stosując metodę wyszukiwania binarnego w celu znalezienia odpowiedniej wartości). To, co teraz omawiamy, ma również znaczenie dla zarządzania kontrolą przeciążeń przez TCP (patrz rozdział 16.). W przypadku zastosowania techniki PLPMTUD sytuacja jest podobna, oprócz tego, że nie są wykorzystywane komunikaty PTB. Wówczas wykonujący procedurę PMTUD protokół musi być zdolny do szybkiego wykrywania przypadków odrzucania komunikatów i do wykonywania samodzielnej regulacji rozmiaru datagramów.

Ponieważ trasy mogą zmieniać się dynamicznie, po upływie pewnego czasu od ostatniego zmniejszenia rozmiaru segmentu można spróbować użycia większej wartości (nieprzekraczającej początkowego SMSS). Wskazówki zawarte w dokumentach [RFC1191] i [RFC1981] zalecają, aby ten okres czasu wynosił ok. 10 minut.

Występuje szereg problemów z technologią PMTUD, kiedy funkcjonuje ona w środowisku Internetu, z zaporami sieciowymi, które blokują komunikaty PTB (patrz [RFC2923]). Spośród wielu problemów eksploatacyjnych związanych z PMTUD najbardziej kłopotliwym są tzw. **czarne dziury** (*black holes*), chociaż sytuacja ulega poprawie (wg źródła [LS10], 80% badanych systemów potrafiło właściwie przetwarzać komunikaty PTB). Czarne dziury PMTUD powstają w sytuacji, kiedy implementacja TCP, zależna od dostarczania komunikatów ICMP w procesie dostosowania swojego rozmiaru segmentu, nie otrzymuje ich. Może to nastąpić z kilku powodów, łącznie z sytuacją, w której konfiguracja zapory sieciowej lub NAT uniemożliwiają przekazywanie tego rodzaju komunikatów ICMP. W konsekwencji mamy połączenie TCP, które przestaje funkcjonować, jeśli tylko zacznie używać większych pakietów. Może być to sytuacja trudna do zdiagnozowania, ponieważ nie mogą być przekazywane tylko duże pakiety. Mniejsze zaś (takie jak pakiety SYN lub SYN+ACK używane do ustanowienia połączenia) na ogół są transmitowane skutecznie. Niektóre implementacje protokołu TCP posiadają mechanizm wykrywania czarnych dziur, który sprowadza się do prób zastosowania mniejszego rozmiaru segmentu w przypadku, kiedy segment był kilkakrotnie retransmitowany.

13.4.1. Przykład

Możemy prześledzić prawidłowe zachowanie procedury PMTUD w sytuacji, gdy pośredniczący router ma MTU mniejsze niż MSS każdego z punktów końcowych. Aby wykreować taką sytuację, zaczynamy od routera (linuksowego hosta o lokalnym adresie 10.0.0.1), który łączy się z dostawcą usługi DSL poprzez interfejs PPPoE. Łącze PPPoE korzysta z MTU równego 1492 (1500 bajtów dla Ethernetu minus 6 bajtów narzutu protokołu PPPoE i minus dalsze 2 bajty narzutu protokołu PPP; patrz rozdział 3.). Rysunek 13.7 stanowi ilustrację tej topologii.



Rysunek 13.7. *Enkapsulacja protokołu PPPoE obniża MTU ścieżki większości połączeń TCP do 1492 bajtów z wartości, która inaczej wynosiłaby 1500 bajtów (typowe MTU dla Ethernetu). Aby zademonstrować, jak TCP używa PMTUD, ustawiliśmy MTU na jeszcze mniejszą wartość (288 bajtów)*

Aby spowodować, by działanie mechanizmu PMTUD było widoczne wyraźniej, możemy zmniejszyć rozmiar MTU dla łącza PPPoE z 1492 do, powiedzmy, 288 bajtów. To zadanie wykonamy za pomocą następującego polecenia, wydanego na (pełniącym rolę routera) hoście `GW`:

```
Linux (GW) # ifconfig ppp0 mtu 288
```

Dodatkowo musimy poinformować system klienta (`C`), że dopuszcza się małe rozmiary segmentów:

```
Linux (C) $ sysctl -w net.ipv4.route.min_pmtu=68
```

Gdybyśmy nie wykonali tej drugiej operacji, Linux zablokowałby swoje minimalne MTU ścieżki na domyślnej wartości wynoszącej 552 bajty, pozwalającej uniknąć pewnych ataków sieciowych wykorzystujących małe MTU (patrz podrozdział 13.8). W konsekwencji w naszym przykładzie wszystkie segmenty posiadające rozmiar większy niż 288 bajtów byłyby poddane fragmentacji. By tego uniknąć i żeby zademonstrować PMTUD w sposób bardziej efektywny, usuwamy to (linuksowe) minimum. Następnie rozpoczynamy transfer pliku z komputera `C` (adres `10.0.0.123`) do serwera `S` w Internecie (adres `169.229.62.97`). Na listingu 13.2 pokazujemy wyświetlony przez program `tcpdump` zapis śledzenia pakietów wygenerowanych przez tę operację. Kilka wierszy zostało zawiniętych, a nieistotne pola usunęliśmy dla większej klarowności.

Listing 13.2. *Mechanizm odkrywania MTU ścieżki znajduje odpowiedni rozmiar segmentu do użycia przy przesyłaniu przez sieć, w której łącze pośrednie ma mniejsze MTU niż oba punkty końcowe*

```
1 20:20:21.992721 IP (tos 0x0, ttl 45, id 43565, offset 0, flags [DF],
                proto 6, length: 588)
                169.229.62.97.22 > 10.0.0.123.1027: P [tcp sum ok]
                41:577(536) ack 23

2 20:20:21.993727 IP (tos 0x0, ttl 64, id 57659, offset 0, flags [DF],
                proto 6, length: 588)
                10.0.0.123.1027 > 169.229.62.97.22: P [tcp sum ok]
                23:559(536) ack 577
```

```
3 20:20:21.994093 IP (tos 0xc0, ttl 64, id 57547, offset 0, flags
                [none], proto 1, length: 576)
                10.0.0.1 > 10.0.0.123: icmp 556:
                169.229.62.97 unreachable - need to frag (mtu 288) for
                  IP (tos 0x0, ttl 63, id 57659, offset 0, flags [DF],
                  proto 6, length: 588)
                    10.0.0.123.1027 > 169.229.62.97.22:
                    P 23:559(536) ack 577

4 20:20:21.994884 IP (tos 0x0, ttl 64, id 57660, offset 0, flags [DF],
                                 proto 6, length: 288)
                10.0.0.123.1027 > 169.229.62.97.22: . [tcp sum ok]
                23:259(236) ack 577

...
5 20:20:22.488856 IP (tos 0x0, ttl 45, id 6712, offset 0, flags [DF],
                proto 6, length: 836)
                169.229.62.97.22 > 10.0.0.123.1027: P [tcp sum ok]
                857:1641(784)ack 855

...
6 20:20:29.672947 IP (tos 0x8, ttl 64, id 57679, offset 0, flags [DF],
                proto 6, length: 1452)
                10.0.0.123.1027 > 169.229.62.97.22: . [tcp sum ok]
                1431:2831(1400) ack 2105

7 20:20:29.674123 IP (tos 0xc8, ttl 64, id 57548, offset 0, flags
                [none], proto 1, length: 576)
                10.0.0.1 > 10.0.0.123: icmp 556:
                169.229.62.97 unreachable - need to frag (mtu 288) for
                  IP (tos 0x8, ttl 63, id 57679, offset 0, flags [DF],
                  proto 6, length: 1452)
                  10.0.0.123.1027 > 169.229.62.97.22: .
                  1431:2831(1400) ack 2105
8 20:20:29.673751 IP (tos 0x8, ttl 64, id 57680, offset 0, flags [DF],
                proto 6, length: 1452)
                10.0.0.123.1027 > 169.229.62.97.22: . [tcp sum ok]
                2831:4231(1400) ack 2105
9 20:20:29.675180 IP (tos 0xc8, ttl 64, id 57549, offset 0, flags
                [none], proto 1, length: 576)
                10.0.0.1 > 10.0.0.123: icmp 556:
                169.229.62.97 unreachable - need to frag (mtu 288) for
                  IP (tos 0x8, ttl 63, id 57680, offset 0, flags [DF],
                  proto 6, length: 1452)
                  10.0.0.123.1027 > 169.229.62.97.22:
                  2831:4231(1400) ack 2105

10 20:20:29.674932 IP (tos 0x8, ttl 64, id 57681, offset 0, flags
                [DF], proto 6, length: 288)
                10.0.0.123.1027 > 169.229.62.97.22: . [tcp sum ok]
                1431:1667(236) ack 2105

11 20:20:29.675143 IP (tos 0x8, ttl 64, id 57682, offset 0, flags
                [DF], proto 6, length: 288)
                10.0.0.123.1027 > 169.229.62.97.22: . [tcp sum ok]
                1667:1903(236) ack 2105
```

W zaprezentowanym powyżej wydruku programu tcpdump połączenie zostało już ustanowione i opcje MSS zostały wymienione między jego stronami. Wszystkie pakiety w tym połączeniu mają ustawiony bit DF, więc oba końce połączenia wykonują procedurę PMTUD. Pierwszy pakiet strony odległej ma 588 bajtów długości i przechodzi przez router z powodzeniem w jednym kawałku, pomimo naszego ustawienia MTU dla łączy PPPoE wynoszącego 288 bajtów. Powodem tego jest asymetria w konfiguracji MTU. Chociaż lokalny koniec łącza PPPoE używa maksymalnej jednostki transmisyjnej w wysokości 288 bajtów, to drugi koniec używa większego rozmiaru parametru SMSS, przypuszczalnie wynoszącego 1492 bajty. To nas stawia w sytuacji, w której pakiety wychodzące muszą być małe (mieć 288 bajtów długości lub mniej), a pakiety podróżujące w odwrotnym kierunku mogą być większe.

Kiedy koniec lokalny usiłuje przesłać większy pakiet o rozmiarze 588 bajtów z włączonym bitem DF, zostaje wygenerowany przez router (10.0.0.1) komunikat PTB wskazujący, że odpowiednie dla łącza MTU następnego przeskoku wynosi 288 bajtów. Protokół TCP reaguje, wysyłając kolejny pakiet o rozmiarze 288 bajtów, zgodnie z otrzymanym zaleceniem. Następnie, aby przesłać resztę danych, które pierwotnie próbował przesłać w swoim 588-bajtowym pakiecie, wysyła dwa dodatkowe pakiety[1], o rozmiarach 288 i 116. Jak możemy zaobserwować, podobne zmniejszenie rozmiaru pakietów w następstwie odebrania komunikatu PTB powtarza się w trakcie transferu pliku.

Proces odkrywania PTMU jest jedną z niewielu metod, w ramach których protokół TCP w jawny sposób próbuje dostosować swój rozmiar segmentu już po uruchomieniu połączenia, przynajmniej wtedy, kiedy są przesyłane wielkie ilości danych. Rozmiar segmentu może mieć wpływ na całkowitą przepustowość, podobnie jak rozmiar okna. Wpływ tych czynników na całkowitą wydajność omówimy w rozdziale 15.

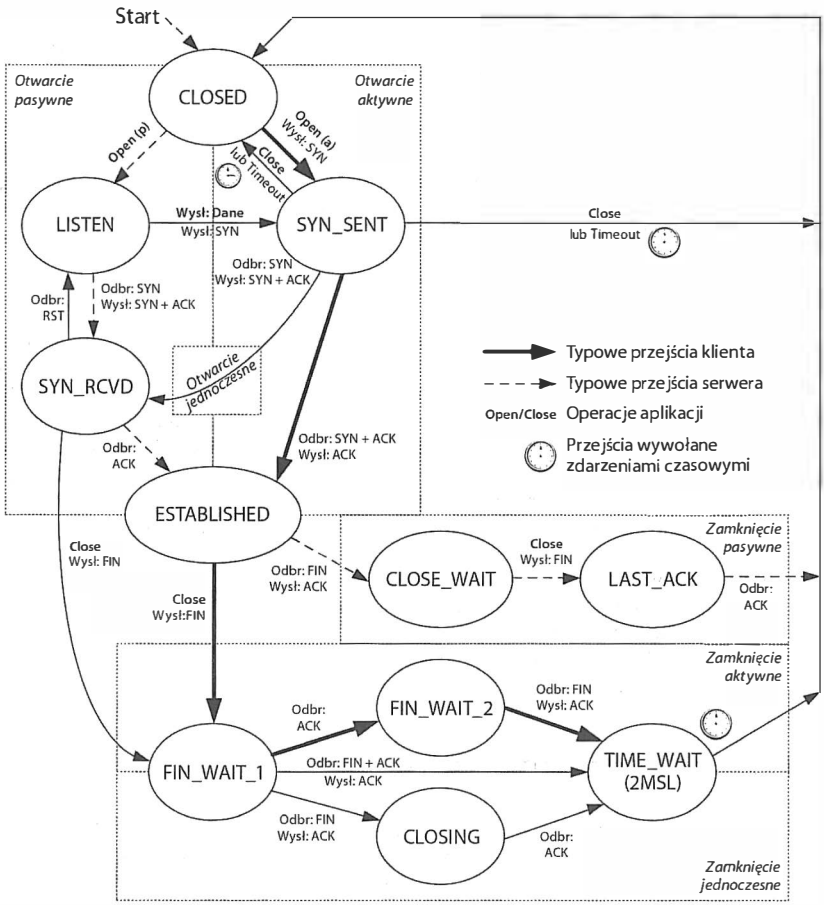
13.5. Przejścia między stanami protokołu TCP

Opisaliśmy liczne reguły dotyczące inicjacji i kończenia połączenia TCP i zobaczyliśmy, jakie typy segmentów są przesyłane podczas różnych faz połączenia. Reguły, które rozstrzygają o tym, co protokół TCP robi, są określone przez stan, w jakim się znajduje. Bieżący stan ulega zmianie na skutek różnych czynników, takich jak transmitowane lub odbierane segmenty, zegary sygnalizujące wygaśnięcie ustawionego czasu, operacje odczytu i zapisu wykonywane przez aplikacje czy informacje pochodzące od innych warstw. Reguły dotyczące tych zmian mogą być streszczone w diagramie stanów protokołu TCP.

13.5.1. Diagram stanów protokołu TCP

Diagram stanów protokołu TCP został pokazany na rysunku 13.8. Elementy owalne oznaczają stany, a strzałki — przejścia między stanami. Każdy z punktów końcowych połączenia przechodzi przez te stany. Niektóre przejścia są powodowane odebraniem segmentu z ustawionymi pewnymi bitami sterującymi (np. SYN, ACK, FIN). Pewne przejścia także powodują wysłanie segmentu z ustawionymi konkretnymi bitami sterującymi.

[1] Te pakiety, które zostały wysłane przez klienta po pakiecie 4., nie zostały pokazane na listingu 13.2 — *przyp. tłum.*



Rysunek 13.8. *Diagram stanów protokołu TCP (nazywany również skończonym automatem stanów). Strzałki przedstawiają przejścia między stanami wynikające z wysyłania i odbierania segmentów lub wygasania limitów czasu. Wytłuszczone strzałki pokazują typowe zachowanie klienta, a strzałki narysowane linią przerywaną przedstawiają typowe zachowanie serwera. Wytłuszczone dyrektywy (np. open, close) oznaczają operacje wykonywane przez aplikacje*

Inne przejścia mogą być wywołane przez działania aplikacji lub przez wygaśnięcie limitu czasu. Każdy z tych przypadków jest pokazany na diagramie w postaci adnotacji tekstowej towarzyszącej strzałce oznaczającej przejście. Na etapie inicjacji protokół TCP rozpoczyna od stanu `CLOSED`. Zwykle pierwsze przejście przenosi go do stanu `SYN_SENT` albo do stanu `LISTEN`, w zależności od tego, czy nasz protokół otrzymał polecenie wykonania, odpowiednio, aktywnego, czy pasywnego otwarcia.

Zauważmy, że w tym diagramie tylko część przejść między stanami jest oznaczona jako „typowe". Oznaczyliśmy normalne przejścia klienta wytłuszczoną strzałką narysowaną linią ciągłą, a normalne przejścia serwera — strzałką narysowaną linią przerywaną.

Dwa przejścia prowadzące do stanu ESTABLISHED odpowiadają otwarciu połączenia, a dwa przejścia wychodzące od stanu ESTABLISHED odpowiadają zakończeniu połączenia. Stan ESTABLISHED to stan, w którym może zachodzić transfer danych między dwoma punktami końcowymi w obu kierunkach. W rozdziałach od 14. do 17. opisujemy to, co się dzieje w tym stanie.

Stany oznaczone etykietami FIN_WAIT_1, FIN_WAIT_2 i TIME_WAIT umieściliśmy (przynajmniej częściowo) w obrębie prostokąta nazwanego „Otwarcie aktywne". Jest to zespół stanów, w które wchodzimy, kiedy aplikacja lokalna inicjuje żądanie zamknięcia. Dwa inne stany (CLOSE_WAIT i LAST_ACK) są zebrane w oznaczonym linią przerywaną prostokącie opatrzonym etykietą „Otwarcie pasywne". Stany te odpowiadają oczekiwaniu na potwierdzenie segmentu FIN i wykonanie procedury zamknięcia przez partnera połączenia. Zamknięcie jednoczesne, które jest formą podwójnego aktywnego zamknięcia, używa stanu CLOSING.

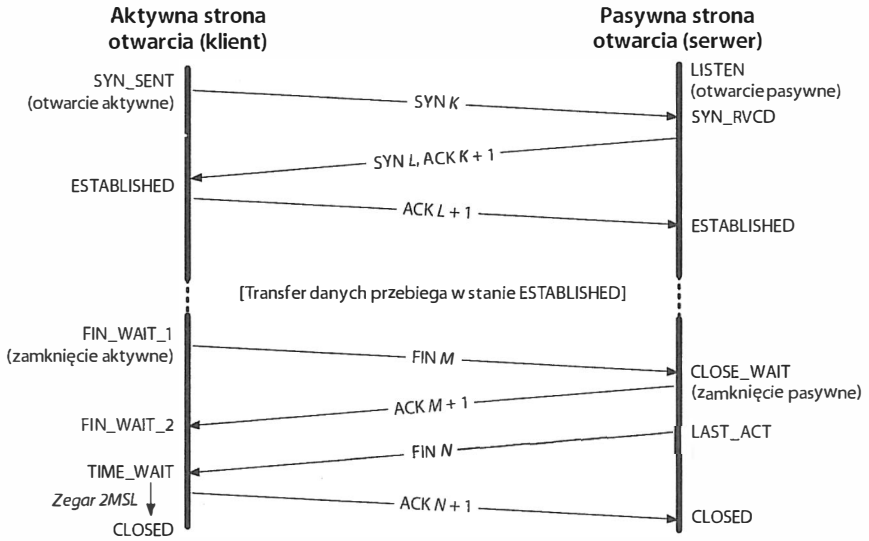
Nazwy 11 stanów (CLOSED, LISTEN, SYN_SENT itd.) przedstawionych na tym rysunku opierają się na nazwach wyświetlanych przez polecenie netstat występujące w systemach UNIX, Linux i Windows, które z kolei zostały utworzone na podstawie nazw po raz pierwszy użytych w dokumencie [RFC0793]. Stan CLOSED nie jest w rzeczywistości „oficjalnym" stanem, lecz został dodany jako użyteczny punkt startowy i końcowy naszego diagramu.

Przejście od stanu LISTEN do stanu SYN_SENT jest dozwolone w protokole TCP, ale nie jest obsługiwane przez gniazda Berkeley i rzadko występuje. Przejście od stanu SYN_RCVD z powrotem do stanu LISTEN jest uprawnione tylko wtedy, gdy stan SYN_RCVD został osiągnięty po przejściu ze stanu LISTEN (normalny scenariusz), a nie ze stanu SYN_SENT (otwarcie jednoczesne). Oznacza to, że jeśli wykonujemy otwarcie pasywne (wchodząc do stanu LISTEN), odbieramy segment SYN, wysyłamy segment SYN z potwierdzeniem ACK (przechodząc do stanu SYN_RCVD), a potem otrzymujemy sygnał resetujący zamiast ACK, nasz punkt końcowy wraca do stanu LISTEN i oczekuje na przyjście następnego żądania połączenia.

Na rysunku 13.9 pokazujemy normalne ustanowienie i zakończenie połączenia TCP, wyszczególniając różne stany, przez jakie przechodzą i klient, i serwer. Jest to prostsza wersja rysunku 13.1 pokazująca odnośne stany, ale już bez opcji czy szczegółów dotyczących numerów ISN. Zakładamy na rysunku 13.9, że klient po lewej stronie wykonuje otwarcie aktywne, a serwer po prawej stronie wykonuje otwarcie pasywne. Chociaż pokazujemy, że to klient wykonuje otwarcie aktywne, to pamiętajmy, że, jak to wspomniano wcześniej, każda strona może wykonać otwarcie aktywne.

13.5.2. Stan TIME_WAIT (odczekiwanie 2MSL)

Stan TIME_WAIT jest także nazywany stanem odczekiwania 2MSL. Jest to stan, w którym protokół TCP odczekuje przez okres czasu równy podwójnej wartości parametru MSL (*Maximum Segment Lifetime*; maksymalny czas życia segmentu), a który bywa też określany angielskim terminem *timed wait* (odczekiwanie ustalonego okresu czasu). Każda implementacja musi określić wartość parametru MSL. Jest to maksymalna ilość czasu, przez jaki segment może istnieć w sieci, zanim zostanie odrzucony. Wiemy, że ten limit czasowy nie jest dowolny, ponieważ segmenty TCP są transmitowane jako datagramy



Rysunek 13.9. *Stany protokołu TCP odpowiadające normalnemu ustanowieniu i zakończeniu połączenia*

protokołu IP, a datagram IP posiada pole *TTL* lub pole *Limit przeskoków* (*Hop Limit*), które ograniczają efektywny czas życia (patrz rozdział 5.). Dokument [RFC0793] określa MSL jako 2 minuty. Wartości przyjmowane w powszechnie występujących implementacjach wynoszą 30 sekund, 1 minuta lub 2 minuty. W większości przypadków wartości te mogą być modyfikowane. W Linuksie parametr `net.ipv4.tcp_fin_timeout` przechowuje wartość limitu czasu odczekiwania `2MSL` (w sekundach). W systemie Windows wartość tę przechowuje następujący klucz rejestru:

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\TcpTimedWaitDelay
```

Zakres dozwolonych wartości wynosi od 30 do 300 s. Dla protokołu IPv6 wyrażenie `Tcpip` należy zastąpić przez `Tcpip6`.

Przy określonej dla danej implementacji wartości parametru `MSL` reguła jest następująca: kiedy protokół TCP wykonuje zamknięcie aktywne i wysyła końcowe `ACK`, dane połączenie musi pozostawać w stanie `TIME_WAIT` przez czas równy podwójnej wartości `MSL`. Pozwala to protokołowi TCP na ponowne wysłanie końcowego potwierdzenia `ACK`, w przypadku gdy zostało one zgubione. Końcowe `ACK` jest przesyłane ponownie nie dlatego, że TCP retransmituje potwierdzenia `ACK` (nie konsumują one numerów sekwencyjnych i nie są retransmitowane przez TCP), ale dlatego, że druga strona wykonuje retransmisję swojego sygnału `FIN` (który konsumuje numer sekwencyjny). Faktycznie, protokół TCP zawsze retransmituje segmenty `FIN`, dopóki nie otrzyma końcowego potwierdzenia `ACK`.

Innym efektem stanu odczekiwania `2MSL` jest to, że kiedy implementacja TCP znajduje się w nim, punkty końcowe definiujące dane połączenie (adres IP klienta, numer portu klienta, adres IP serwera i numer portu serwera) nie mogą być użyte ponownie. Połączenie to może być wykorzystane ponownie tylko wtedy, gdy czas odczekiwania `2MSL` zakończy się lub kiedy nowe połączenie użyje numeru `ISN`, który przekracza najwyższy

numer sekwencyjny zastosowany w poprzedniej instancji połączenia (patrz [RFC1122]), albo kiedy użycie opcji *Znaczniki czasu* pozwala na odróżnienie segmentów pochodzących z poprzedniej instancji połączenia, co zapobiega ich pomyleniu z segmentami należącymi do bieżącego połączenia (patrz [RFC6191]). Niestety, pewne implementacje narzucają bardziej rygorystyczne ograniczenie. W tych systemach lokalny numer portu nie może być wykorzystany ponownie w ramach całego systemu, w sytuacji gdy jest on numerem portu lokalnego **dowolnego** punktu końcowego znajdującego się w stanie odczekiwania 2MSL. Zobaczymy przykłady tego ograniczenia na listingach 13.3 i 13.4.

Większość implementacji i interfejsów API oferuje sposób na obejście tego ograniczenia. W API gniazd Berkeley operację obejścia umożliwia opcja gniazdowa SO_REUSEADDR. Pozwala ona wywołującej aplikacji na przydzielenie sobie numeru portu lokalnego nawet wtedy, gdy ten numer portu jest częścią jakiegoś połączenia znajdującego się w stanie odczekiwania 2MSL. Zobaczymy jednak, że nawet gdy stosujemy ten mechanizm obejścia odnoszący się do pojedynczego gniazda (para złożona z adresu i numeru portu), reguły protokołu TCP wciąż zapobiegają (lub powinny zapobiegać) ponownemu użyciu tego samego numeru portu przez kolejną instancję tego samego połączenia, które znajduje się w stanie odczekiwania 2MSL. Wszystkie opóźnione segmenty, które przychodzą w ramach połączenia będącego w stanie odczekiwania 2MSL, są odrzucane. Ponieważ połączenie zdefiniowane przez dwie pary adres-port, będące w stanie odczekiwania 2MSL, nie może być wykorzystane ponownie w tym okresie czasu, to kiedy w końcu zostanie ustanowione nowe prawidłowe połączenie, wiemy, że opóźnione segmenty pochodzące z wcześniejszej instancji tego połączenia nie mogą być fałszywie zinterpretowane jako część nowego połączenia.

W aplikacjach interaktywnych z reguły klient wykonuje zamknięcie aktywne i przechodzi do stanu TIME_WAIT. Serwer zwykle wykonuje zamknięcie pasywne i nie przechodzi przez stan TIME_WAIT. Z tego wynika, że jeśli zakończymy działanie klienta i zrestartujemy tego samego klienta zaraz potem, nowy klient nie może użyć ponownie tego samego numeru lokalnego portu. Zazwyczaj nie stanowi to problemu, ponieważ klienty z reguły używają portów efemerycznych przydzielonych przez system operacyjny i nie dbają o to, jaką ten przypisany numer portu ma wartość. (Pamiętajmy, że właściwe zapewnienie losowości wyboru numerów portu jest praktyką zalecaną ze względów bezpieczeństwa — patrz [RFC6056]). Należy to wiedzieć, ponieważ nie można wykluczyć, że klient, który wykonuje szybko dużą ilość połączeń (szczególnie z tym samym serwerem), będzie musiał czekać na zakończenie innych połączeń w sytuacji deficytu portów efemerycznych.

Jednak w przypadku serwerów sytuacja jest inna. One prawie zawsze korzystają z dobrze znanych portów. Jeśli zakończymy proces serwera, który ma ustanowione połączenie, i natychmiast próbujemy go zrestartować, serwer nie może skorzystać z przypisanego mu numeru portu, tworząc swój punkt końcowy (otrzymuje błąd wiązania „Address already in use" — adres jest już używany), ponieważ ten numer portu jest częścią połączenia, które znajduje się w stanie odczekiwania 2MSL. Może upłynąć od 1 do 4 minut, zanim serwer będzie mógł zostać uruchomiony ponownie, w zależności od wartości parametru MSL dla lokalnego systemu. Możemy zaobserwować ten scenariusz przy użyciu naszego programu sock. Na listingu 13.3 uruchamiamy serwer, łączymy się z nim w roli klienta, a potem zatrzymujemy serwer.

Listing 13.3. *Połączenie TCP musi zakończyć odczekiwanie 2MSL w stanie TIME_WAIT, zanim numer portu może być ponownie użyty przez inny proces*

```
Linux% sock -v -s 6666
(teraz klient na innym komputerze łączy się z tym serwerem)
connection on 192.168.10.144.6666 from 192.168.10.140.2623
(serwer został zatrzymany po naciśnięciu klawiszy Ctrl+C)
(teraz serwer został zrestartowany)
Linux% sock -v -s 6666
can't bind local address: Address already in use

Linux% netstat -n -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address       Foreign Address     State
tcp    0      0     192.168.10.144:6666 192.168.10.140:2623 TIME_WAIT

(poczekaj jedną minutę i zrestartuj serwer ponownie)
Linux% sock -v -s 6666
```

Kiedy próbujemy zrestartować serwer, program wyprowadza komunikat błędu wskazujący, że nie może powiązać się ze swoim numerem portu, ponieważ adres jest już używany. To faktycznie oznacza, że dana kombinacja adresu i numeru portu jest już wykorzystywana; znajduje się ona w stanie odczekiwania 2MSL z powodu poprzedniego połączenia. Jest to przykład bardziej rygorystycznego ograniczenia dotyczącego ponownego wykorzystywania numerów portów, o którym wspomnieliśmy wcześniej. Informacja wyprowadzona przez polecenie `netstat` pokazuje, że połączenie jest w stanie `TIME_WAIT`. Mimo że klienty nie mają tak wielu kłopotów ze stanami odczekiwania 2MSL jak serwery, możemy i w ich przypadku zademonstrować ten sam problem, w sytuacji gdy klient określa swój własny numer portu, co pokazano na listingu 13.4.

Listing 13.4. *Klient nie może ponownie użyć numeru portu, który wciąż jest wykorzystywany przez inne połączenie znajdujące się w stanie odczekiwania 2MSL*

```
(uruchom serwer w jednym oknie)
Linux% sock -s -v 6666
(połącz się z nim z innego okna)
Linux% sock -v 127.0.0.1 6666
(serwer identyfikuje połączenie przychodzące)
connection on 127.0.0.1.6666 from 127.0.0.1.2091
(klient identyfikuje ustanowienie połączenia i jego działanie zostaje przerwane)
connected on 127.0.0.1.2091 to 127.0.0.1.6666
^C
(serwer odkrywa, że połączenie zostało zakończone i wykonuje operację exit)
connection closed by peer
Linux%
(klient jest restartowany ze specyfikacją tego samego numeru, co użyty poprzednio)
Linux% sock -b 2091 -v 127.0.0.1 6666
bind() error: Address already in use
(odczekanie 30 sekund i ponowienie próby)
Linux% sock -b 2091 -v 192.168.10.144 6666
connect() error: Connection refused
```

Kiedy po raz pierwszy uruchamiamy klienta, używamy opcji `-v`, aby zobaczyć, jaki numer portu lokalnego (efemeryczny) zostanie przypisany do klienta (2091). Kiedy uruchamiamy klienta po raz drugi, używamy opcji `-b`, nakazując klientowi, by przypisał

sobie lokalny numer portu 2091, zamiast korzystać z nadania następnego efemerycznego numeru portu przez system operacyjny. Jak spodziewaliśmy się, klient nie może tego wykonać, ponieważ port 2091 jest częścią połączenia, które znajduje się w stanie odczekiwania 2MSL. Kiedy czas odczekiwania mija (1 minuta na tym komputerze z systemem Linux), klient próbuje połączyć się raz jeszcze, ale ponieważ proces serwera zakończył działanie, kiedy połączenie zostało przerwane za pierwszym razem, próba klienta zostaje odrzucona. Sposób, w jaki resetujące segmenty TCP są używane do sygnalizacji odmowy połączenia, poznamy w podrozdziale 13.6.

Wspomnieliśmy wcześniej, że większość systemów udostępnia sposób na przesłonięcie zachowania domyślnego, który umożliwi procesom wiązanie się z portami nawet wtedy, gdy te porty stanowią część połączeń znajdujących się w stanie odczekiwania 2MSL. Teraz wypróbujemy ten sam scenariusz, co przedtem, ale w wywołaniu programu sock użyjemy opcji -A, która włącza mechanizm obejścia:

```
Linux% sock -A -v -s 6666
Linux% sock -A -v -s 6666
```

W powyższym przykładzie uruchamiamy serwer z opcją -A, która z kolei włącza wcześniej wspomnianą opcję gniazda SO_REUSEADDR. Postępując w ten sposób, pozwalamy serwerowi na powiązanie się ze swoim portem, nawet jeśli jest on częścią połączenia znajdującego się w stanie odczekiwania 2MSL. Jeżeli jednak spróbujemy użyć ponownie klienta natychmiast, z tym samym numerem portu, zdarzy się, co następuje:

```
Linux% sock -b 32840 -v 127.0.0.1 6666
bind() error: Address already in use
```

Raz jeszcze powtarzamy: punkt końcowy jest w użyciu, więc klientowi nie udaje się nawiązać połączenia. Jeśli jednak użyjemy opcji -A również w przypadku klienta, możemy wymusić zadziałanie połączenia:

```
Linux% sock -A -b 32840 -v 127.0.0.1 6666
Connected on 127.0.0.1.32840 to 127.0.0.1.6666
TCP_MAXSEG = 16383
```

Tu widzimy, że chociaż to samo połączenie (posiadające ten sam 4-członowy identyfikator) jest używane ponownie przed upłynięciem czasu odczekiwania 2MSL, to użycie opcji -A wymusiło możliwość zaistnienia tego połączenia. Oczywiście, ma to miejsce na tym samym komputerze, więc system operacyjny jest w stanie ustalić, które procesy reprezentują określone punkty końcowe połączeń znajdujących się w stanie odczekiwania 2MSL i (przynajmniej potencjalnie) oddzielić je. Co się stanie, jeśli znów spróbujemy zrobić to samo, ale tym razem ustanowimy połączenie z innego hosta? Przetestujmy ten pomysł:

```
(uruchom serwer na pierwszym komputerze)
Linux% sock -v -s 6666
(połącz się z nim z drugiego komputera z systemem Windows)
C:\> sock -A -v 10.0.0.1 6666
(serwer identyfikuje połączenie przychodzące)
connection on 10.0.0.1.6666 from 10.0.0.3.2172
(klient identyfikuje ustanowienie połączenia i jego działanie zostaje przerwane)
connected on 10.0.0.3.2172 to 10.0.0.1.6666
^C
C:\>
(serwer identyfikuje zakończenie połączenia i wykonuje operację exit)
connection closed by peer
```

```
Linux%
(klient zostaje uruchomiony ponownie, z tym samym numerem portu, co poprzednio)
C:\> sock -A -b 2091 -v 10.0.0.1 6666
connect() error: Address already in use
C:\> sock -A -b 2091 -v 10.0.0.1 6666
connect() error: Address already in use

(odczekaj 30 sekund i spróbuj ponownie)

C:\> sock -A -b 2091 -v 10.0.0.1 6666
connect() error: Connection refused
```

Przykład ten jest podobny do poprzedniego z tym wyjątkiem, że klient i serwer znajdują się na różnych komputerach. Zauważamy, że niezależnie od flagi -A ustawionej przez klienta, stosowany jest czas odczekiwania 2MSL. W tym przypadku odczekiwanie 2MSL trwa przez 30 s. Po upływie tego czasu klient usiłuje nawiązać kontakt z serwerem, który już zakończył działanie.

Interesująca rzecz wydarzy się, jeśli zamienimy komputery klienta i serwera. Teraz powtórzymy nasz eksperyment, używając systemu Windows jako serwera i Linuksa w charakterze klienta:

```
(uruchom serwer na komputerze z systemem Windows)
C:\> sock -v -s 6666

(połącz się z nim z drugiego komputera z systemem Linux)
Linux% sock -A -v 192.168.10.145 6666

(serwer identyfikuje połączenie przychodzące)
connection on 192.168.10.145.6666 from 192.168.10.145.32843

(klient identyfikuje ustanowienie połączenia i jego działanie zostaje przerwane)
connected on 192.168.10.144.32843 to 192.168.10.145.6666
^C
Linux%

(serwer identyfikuje zakończenie połączenia i wykonuje operację exit)

connection closed by peer
C:\>

(klient jest restartowany z tym samym numerem portu lokalnego, co przedtem)

Linux% sock -A -b 32843 -v 192.168.10.144 6666
bind() error: Connection refused
```

W tym momencie oczekiwalibyśmy, że port lokalny 32843 będzie niedostępny, ale z powodu sposobu, w jaki opcja -A funkcjonuje w systemie Linux, otrzymujemy możliwość skorzystania z niego. To naruszenie oryginalnej specyfikacji protokołu TCP, ale dopuszczone na podstawie dokumentów [RFC1122] i [RFC6191], jak już wspominaliśmy wcześniej. Specyfikacje te pozwalają, aby nowe żądanie połączenia nadeszło i zostało zaakceptowane dla połączenia znajdującego się w stanie TIME_WAIT, jeśli istnieje mocne uzasadnienie pozwalające wierzyć, że segmenty przesyłane w ramach nowego połączenia nie będą mylone z segmentami należącymi do poprzedniej jego instancji, w oparciu o kombinację numerów sekwencyjnych i znaczników czasu. Dokument [RFC1337] i aneks do dokumentu [RFC1323] pokazują pewne pułapki związane z tą zasadą.

13.5.3. Pojęcie czasu ciszy

Odczekiwanie 2MSL stanowi ochronę przed interpretowaniem opóźnionych segmentów pochodzących z wcześniejszej instancji połączenia jako części nowego połączenia, które używa tych samych lokalnych i obcych adresów IP i numerów portów. Ale jest skuteczne tylko wtedy, kiedy host z połączeniami będącymi w stanie odczekiwania 2MSL nie ulegnie awarii.

Co się stanie, jeśli host z połączeniami w stanie TIME_WAIT ulegnie awarii, zostanie uruchomiony ponownie w czasie nieprzekraczającym wartości MSL i niezwłocznie ustanowi nowe połączenia przy użyciu tych samych lokalnych oraz obcych adresów IP i numerów portów, które były wykorzystywane przez lokalne połączenia znajdujące się w stanie TIME_WAIT przed wystąpieniem awarii? W tym scenariuszu opóźnione segmenty pochodzące z połączeń, które istniały przed awarią, mogą być błędnie zinterpretowane jako należące do nowych połączeń utworzonych po ponownym uruchomieniu systemu. Może się tak zdarzyć niezależnie od tego, w jaki sposób został wybrany początkowy numer sekwencyjny po restarcie hosta.

W celu zapewnienia ochrony przed tym scenariuszem dokument [RFC0793] stanowi, że protokół TCP powinien odczekać ilość czasu równą wartości MSL, zanim utworzy jakiekolwiek nowe połączenia po restarcie lub awarii. Okres ten jest nazywany **czasem ciszy** (*quiet time*). Niewiele implementacji przestrzega tej zasady, ponieważ większość hostów potrzebuje na ponowne uruchomienie po awarii więcej czasu niż wartość MSL. Ponadto, jeżeli aplikacje używają swoich własnych sum kontrolnych lub szyfrowania, błędy takie jak te są łatwe do wykrycia.

13.5.4. Stan FIN_WAIT

W stanie FIN_WAIT_2 protokół TCP wysłał już segment FIN, a druga strona go potwierdziła. Z wyjątkiem sytuacji, gdy wykonywane jest zamknięcie częściowe, TCP musi czekać, aż aplikacja po drugiej stronie połączenia uzna, że odebrała sygnał końca pliku, i zamknie swój koniec połączenia, co spowoduje wysłanie segmentu FIN. Dopiero wtedy, kiedy aplikacja wykona to zamknięcie (i jej segment FIN zostanie odebrany), protokół TCP wykonujący aktywne zamknięcie przejdzie ze stanu FIN_WAIT_2 do stanu TIME_WAIT. Oznacza to, że jeden z końców połączenia może pozostać w tym stanie na zawsze. Druga strona pozostaje wciąż w stanie CLOSE_WAIT i może pozostać w nim na zawsze, chyba że aplikacja zadecyduje o wydaniu polecenia zamknięcia.

Wiele implementacji zapobiega temu nieskończonemu oczekiwaniu w stanie FIN_WAIT_2 w sposób następujący. Jeżeli aplikacja wykonująca zamknięcie aktywne realizuje całkowite zamknięcie, a nie zamknięcie częściowe, wskazujące, że przewidywane jest jeszcze odbieranie danych, ustawiany jest zegar. Jeśli połączenie jest w stanie bezczynności w momencie wygaśnięcia zegara, TCP przenosi połączenie do stanu CLOSED. W Linuksie wartość zmiennej net.ipv4.tcp_fin_timeout może być regulowana w celu kontrolowania liczby sekund, na jaką jest ustawiany zegar. Wartość domyślna wynosi 60 s.

13.5.5. Przejścia odpowiadające jednoczesnemu otwarciu i jednoczesnemu zamknięciu

Widzieliśmy już normalne użycie stanów SYN_SENT i SYN_RCVD, które kolejno odpowiadają wysłaniu i odebraniu segmentów SYN. Jak to zilustrowano na rysunku 13.3, protokół TCP został celowo zaprojektowany, by obsługiwać jednoczesne otwarcia, których efektem jest pojedyncze połączenie. Kiedy zachodzi otwarcie jednoczesne, przejścia między stanami różnią się od pokazanych na rysunku 13.9. Obie strony wysyłają segment SYN prawie w tym samym czasie, przechodząc do stanu SYN_SENT. Kiedy każda strona odbierze segment SYN od swojego partnera, przechodzi do stanu SYN_RCVD, po czym każda strona wysyła sygnał SYN ponownie i potwierdza otrzymany SYN. Kiedy każda ze stron odbierze segment SYN z potwierdzeniem ACK, następuje przejście do stanu ESTABLISHED.

W przypadku jednoczesnego zamknięcia, co przedstawia rysunku 13.8, obie strony przechodzą ze stanu ESTABLISHED do stanu FIN_WAIT_1, kiedy aplikacja wydaje polecenie zamknięcia. Powoduje to wysłanie przez obie strony segmentów FIN, które mijają się gdzieś w sieci. Kiedy przychodzi segment FIN od partnera, każdy koniec przechodzi ze stanu FIN_WAIT_1 do stanu CLOSING i każda strona wysyła swoje końcowe potwierdzenie ACK. Po odebraniu końcowego ACK stan każdego punktu końcowego zmienia się na TIME_WAIT i inicjowane jest odczekiwanie 2MSL.

13.6. Segmenty RST

W rozdziale 12. wspominaliśmy o 1-bitowym polu RST w nagłówku TCP. Segment, który ma ten bit ustawiony na wartość 1, jest nazywany segmentem RST. Ogólnie rzecz biorąc, segment RST jest wysyłany przez protokół TCP, ilekroć przychodzi segment, który nie wydaje się być poprawnym segmentem w kontekście wskazanego połączenia. (Używamy terminu „wskazane połączenie", mając na myśli połączenie określone przez dwie pary adresów i numerów portów w nagłówkach TCP i IP pakietu RST). Segmenty RST zazwyczaj powodują szybkie przerwanie połączenia TCP. Możemy skonstruować scenariusze demonstrujące użycie segmentów RST.

13.6.1. Żądanie połączenia z nieistniejącym hostem

Często występujący przypadek, kiedy generowany jest segment RST, to sytuacja, w której przychodzi żądanie połączenia, a żaden proces nie nasłuchuje na porcie docelowym. Zobaczyliśmy to poprzednio, kiedy napotkaliśmy komunikaty błędu „connection refused" (odmowa połączenia). Występują one często w protokole TCP. W przypadku protokołu UDP zobaczyliśmy w rozdziale 10., że generowany jest komunikat ICMP — *Destination Unreachable* (*Port Unreachable*) — kiedy przychodzi datagram z portem docelowym, który nie jest używany. TCP zamiast tego używa segmentu RST.

Wygenerowanie przykładu dla opisanej sytuacji jest trywialne — używamy klienta programu Telnet, podając numer portu, który nie jest wykorzystywany w hoście docelowym. Tym hostem docelowym może być równie dobrze lokalny komputer:

```
Linux% telnet localhost 9999
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
```

Ten komunikat jest wyświetlany przez klienta programu Telnet natychmiast. Na listingu 13.5 pokazujemy wymianę pakietów odpowiadających temu poleceniu.

Listing 13.5. *Resetowanie spowodowane próbą otwarcia połączenia z nieistniejącym portem*

```
1 22:15:16.348064 127.0.0.1.32803 > 127.0.0.1.9999:
       S [tcp sum ok] 3357881819:3357881819(0) win 32767
       <mss 16396,sackOK,timestamp 16945235 0,nop,wscale 0>
       (DF) [tos 0x10] (ttl 64, id 42376, len 60)
2 22:15:16.348105 127.0.0.1.9999 > 127.0.0.1.32803:
       R [tcp sum ok] 0:0(0) ack 3357881820 win 0
       (DF) [tos 0x10] (ttl 64, id 0, len 40)
```

Wartości, którym powinniśmy się przyjrzeć na listingu 13.5, to pola *Numer sekwencyjny* i *Numer ACK* w segmencie RST (drugim). Ponieważ bit ACK nie był włączony w przychodzącym segmencie SYN, numer sekwencyjny segmentu RST jest ustawiony na 0, a numer ACK jest ustawiony na wartość przychodzącego ISN plus liczba bajtów danych w segmencie. Chociaż brakuje danych w segmencie przychodzącym, to bit SYN zajmuje w sensie logicznym 1 bajt przestrzeni numerów sekwencyjnych; dlatego w tym przykładzie numer ACK w segmencie RST jest ustawiony na wartość ISN, plus długość danych (0), plus 1 dla bitu SYN.

Żeby segment RST został zaakceptowany przez TCP, bit ACK musi być ustawiony, a wartość pola *Numer ACK* powinna mieścić się w prawidłowym oknie (patrz rozdział 12.). Pomaga to zapobiec prostemu atakowi, w którym każdy, kto potrafi wygenerować segment RST pasujący do właściwego połączenia (określonego przez 4-członowy identyfikator), mógłby przerwać połączenie (patrz [RFC5961]).

13.6.2. Przerwanie połączenia

Na rysunku 13.1 zobaczyliśmy, że normalnym sposobem zakończenia połączenia jest dla jednej strony wysłanie segmentu FIN. Nazywa się to czasami **zwolnieniem planowym** (*orderly release*), ponieważ segment FIN jest wysyłany po tym, jak wszystkie uprzednio zakolejkowane dane zostały przesłane, i nie ma zwykle żadnej utraty danych. Ale jest również możliwe przerwanie połączenia przez wysłanie segmentu RST zamiast segmentu FIN w dowolnym czasie. Jest ono czasem nazywane **zwolnieniem awaryjnym** (*abortive release*).

Przerwanie połączenia ma dwie cechy istotne dla aplikacji: po pierwsze, jakiekolwiek zakolejkowane dane zostają wyrzucone i natychmiast wysyłany jest segment RST, a po drugie, odbiorca segmentu RST może stwierdzić, że druga strona wykonała przerwanie połączenia zamiast normalnego zamknięcia. Interfejs API używany przez aplikację musi dostarczyć sposób na spowodowanie przerwania połączenia zamiast normalnego zamknięcia.

Interfejs API gniazd oferuje taką możliwość; jest nią użycie opcji gniazda „zwlekaj przy zamknięciu" (SO_LINGER) z wartością zwłoki równą 0. Zasadniczo to oznacza: „Przerwij połączenie natychmiast bez żadnej zwłoki na upewnienie się, że dane dotarły do drugiej strony". W poniższym przykładzie zobaczymy, co się stanie, kiedy zdalne polecenie, które generuje dużą ilość danych wyjściowych, zostaje anulowane przez użytkownika:

```
Linux% ssh linux cat /usr/share/dict/words
Aarhus
Aaron
Ababa
aback
abaft
abandon
abandoned
abandoning
abandonment
abandons
... continues ...
^C
Killed by signal 2.
```

W tym momencie użytkownik zadecydował o przerwaniu wyświetlania danych przez to polecenie. Plik words zawiera 45 427 słów, więc to polecenie było prawdopodobnie jakimś rodzajem pomyłki. Kiedy użytkownik wprowadza znak przerwania, system pokazuje, że proces (w tym wypadku program ssh) został zabity przez sygnał numer 2. Sygnał nazywa się SIGINT i zwykle powoduje zakończenie programu, kiedy zostanie odebrany. Na listingu 13.6 pokazujemy wyjście programu tcpdump dla tego przykładu. (Usunęliśmy wiele pośrednich pakietów, ponieważ nic nie wnoszą do naszej analizy).

Listing 13.6. *Przerwanie połączenia przy użyciu segmentu RST zamiast segmentu FIN*

```
Linux# tcpdump -vvv -s 1500 tcp

1 22:33:06.386747 192.168.10.140.2788 > 192.168.10.144.ssh:
          S [tcp sum ok] 1520364313:1520364313(0) win 65535
          <mss 1460,nop,nop,sackOK>
          (DF) (ttl 128, id 43922, len 48)

2 22:33:06.386855 192.168.10.144.ssh > 192.168.10.140.2788:
          S [tcp sum ok] 181637276:181637276(0) ack 1520364314
          win 5840
          <mss 1460,nop,nop,sackOK>
          (DF) (ttl 64, id 0, len 48)

3 22:33:06.387676 192.168.10.140.2788 > 192.168.10.144.ssh:
          . [tcp sum ok] 1:1(0) ack 1 win 65535
          (DF) (ttl 128, id 43923, len 40)

(... wymiana zaszyfrowanych danych uwierzytelniających i transfer masowych danych wykonane przez ssh ...)
4 22:33:13.648247 192.168.10.140.2788 > 192.168.10.144.ssh:
          R [tcp sum ok] 1343:1343(0) ack 132929 win 0
          (DF) (ttl 128, id 44004, len 40)
```

Segmenty od 1. do 3. pokazują normalne ustanowienie połączenia. W następstwie wprowadzenia znaku przerwania połączenie zostaje awaryjnie zakończone. Segment RST zawiera numer sekwencyjny i numer potwierdzenia. Zauważmy również, że segment RST nie uzyskuje żadnej odpowiedzi drugiej strony — nie jest w ogóle potwierdzany. Odbiorca segmentu RST przerywa połączenie i informuje aplikację, że połączenie zostało zresetowane. To często skutkuje pokazaniem błędu „Connection reset by peer" (połączenie zresetowane przez drugą stronę połączenia) lub podobnym komunikatem.

13.6.3. Połączenia częściowo otwarte

Mówi się, że połączenie TCP jest **częściowo otwarte**, jeśli jedna strona zamknęła lub przerwała połączenie bez wiedzy drugiej. Może się to zdarzyć, ilekroć jedna ze stron ulegnie awarii. Dopóki nie ma próby transferu danych poprzez częściowo otwarte połączenie, strona, która nadal pracuje, nie wykrywa faktu awarii drugiej strony.

Inną częstą przyczyną zaistnienia częściowo otwartego połączenia jest wyłączenie zasilania jednego z hostów bez właściwego zamknięcia systemu. To ma miejsce np. wtedy, gdy komputery PC są używane z uruchomionymi klientami zdalnego logowania i wyłączane pod koniec dnia. Jeżeli nie było żadnego transferu danych trwającego w chwili odcięcia zasilania, serwer nigdy się nie dowie, że klient zniknął (będzie nadal myślał, że połączenie jest w stanie ESTABLISHED). Kiedy użytkownik przychodzi następnego dnia rano, włącza komputer PC i uruchamia nową sesję, nowy proces serwera jest uruchamiany na hoście serwera. (W rozdziale 17. dowiemy się, w jaki sposób przy użyciu opcji podtrzymania aktywności — *keepalive option* — jedna strona połączenia TCP może odkryć, że druga strona znikła).

Możemy łatwo utworzyć połączenie częściowo otwarte. W tym przypadku robimy to po stronie klienta, a nie serwera. Uruchomimy klienta programu Telnet na hoście z adresem IP 10.0.0.1, łącząc się z serwerem usługi Sun RPC (sunrpc, port 111) uruchomionym na komputerze posiadającym adres 10.0.0.7 (patrz listing 13.7). Wprowadzamy jeden wiersz danych wejściowych i widzimy, korzystając z programu tcpdump, że zostaje odebrany przez serwer. Wtedy odłączamy kabel Ethernetu na hoście serwera i przeładowujemy system na tym komputerze. W ten sposób symulujemy awarię hosta serwera. (Odłączamy kabel ethernetowy przed zrestartowaniem serwera, by uniemożliwić mu wysłanie segmentu FIN z otwartych połączeń, co robią niektóre implementacje TCP, gdy są zamykane). Po zrestartowaniu serwera z powrotem podłączamy kabel i próbujemy przesłać następny wiersz tekstu od klienta do serwera. Po restarcie protokół TCP serwera stracił całą pamięć o połączeniach, które istniały przedtem, więc nie wie nic o połączeniu, do którego odwołuje się przesłany segment danych. Zgodnie z regułą protokołu TCP odbiorca odpowiada, wysyłając segment RST.

Listing 13.7. *Host serwera zostaje odłączony od sieci i zrestartowany, co pozostawia częściowo otwarte połączenia u klienta. Kiedy serwer otrzymuje kolejne dane w ramach połączenia, o którym nic już nie wie, odpowiada wysłaniem segmentu RST i powoduje zamknięcie połączenia po obu stronach*

```
Linux% telnet 10.0.0.7 sunrpc
Trying 10.0.0.7...
Connected to 10.0.0.7.
Escape character is '^]'.
foo
```

```
(kabel Ethernetu został odłączony i serwer został zrestartowany)
bar
Connection closed by remote host
```

Na listingu 13.8 pokazujemy dane wyjściowe programu tcpdump dla tego przykładu.

Listing 13.8. *Żądanie resetowania w odpowiedzi na segment danych w połączeniu częściowo otwartym*

```
1 23:15:48.804142 IP (tos 0x10, ttl 64, id 20095, offset 0,
      flags [DF], proto 6, length: 60)
      10.0.0.1.1310 > 10.0.0.7.sunrpc:
      S [tcp sum ok] 2365970104:2365970104(0) win 5840
      <mss 1460,sackOK,timestamp 3849492679 0,nop,wscale 2>
2 23:15:48.804742 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF],
      proto 6, length: 60)
      10.0.0.7.sunrpc > 10.0.0.1.1310:
      S [tcp sum ok] 2093796387:2093796387(0) ack 2365970105 win 5792
      <mss 1460,sackOK,timestamp 654784 3849492679,nop,wscale 0>

3 23:15:48.805028 IP (tos 0x10, ttl 64, id 20097, offset 0,
      flags [DF], proto 6, length: 52)
      10.0.0.1.1310 > 10.0.0.7.sunrpc:
      . [tcp sum ok] 1:1(0) ack 1 win 1460
      <nop,nop,timestamp 3849492680 654784>

4 23:15:51.999394 IP (tos 0x10, ttl 64, id 20099, offset 0,
      flags [DF], proto 6, length: 57)
      10.0.0.1.1310 > 10.0.0.7.sunrpc:
      P [tcp sum ok] 1:6(5) ack 1 win 1460
      <nop,nop,timestamp 3849495875 654784>

5 23:15:51.999874 IP (tos 0x0, ttl 64, id 12773, offset 0,
      flags [DF], proto 6, length: 52)
      10.0.0.7.sunrpc > 10.0.0.1.1310:
      . [tcp sum ok] 1:1(0) ack 6 win 5792
      <nop,nop,timestamp 656421 3849495875>

6 23:17:19.419611 arp who-has 10.0.0.7 (Broadcast) tell 0.0.0.0
7 23:17:20.419142 arp who-has 10.0.0.7 (Broadcast) tell 0.0.0.0
8 23:17:21.427458 arp reply 10.0.0.7 is-at 00:e0:00:88:ad:d6

9 23:17:21.921745 arp who-has 10.0.0.1 tell 10.0.0.7
10 23:17:21.921892 arp reply 10.0.0.1 is-at 00:04:5a:9f:9e:80

11 23:17:23.437114 arp who-has 10.0.0.7 (Broadcast) tell 10.0.0.7

12 23:17:34.804196 arp who-has 10.0.0.7 tell 10.0.0.1
13 23:17:34.804650 arp reply 10.0.0.7 is-at 00:e0:00:88:ad:d6

14 23:17:43.684786 IP (tos 0x10, ttl 64, id 20101, offset 0,
      flags [DF], proto 6, length: 57)
      10.0.0.1.1310 > 10.0.0.7.sunrpc:
      P [tcp sum ok] 6:11(5) ack 1 win 1460
      <nop,nop,timestamp 3849607577 656421>

15 23:17:43.685277 IP (tos 0x10, ttl 64, id 0, offset
      flags [DF], proto 6, length: 40)
      10.0.0.7.sunrpc > 10.0.0.1.1310:
      R [tcp sum ok] 2093796388:2093796388(0) win 0
```

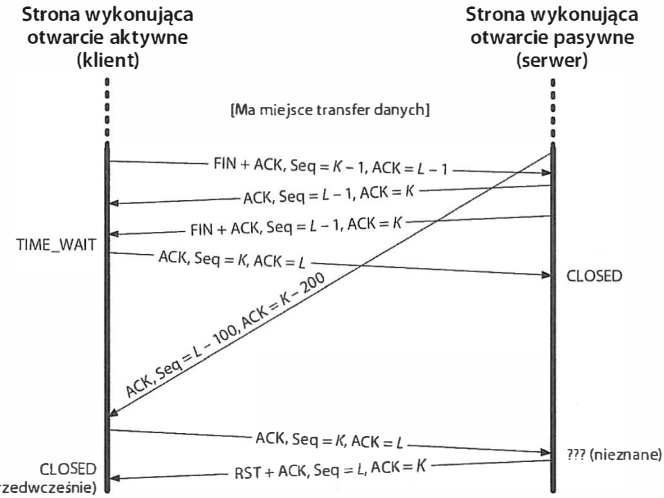
Segmenty od 1. do 3. są normalnym ustanowieniem połączenia. Segment 4. wysyła wiersz „foo" do serwera usługi sunrpc (wymagane 5 bajtów zawiera znaki powrotu karetki i nowego wiersza), a segment 5. to potwierdzenie.

W tym momencie odłączamy kabel Ethernetu od serwera (adres 10.0.0.7), restartujemy host i podłączamy kabel ponownie. To zajmuje ok. 90 s. Następnie wprowadzamy kolejny wiersz na wejściu klienta („bar"), a po naciśnięciu klawisza *Enter* wiersz zostaje wysłany do serwera (pierwszy segment TCP po komunikacji związanej z protokołem ARP na listingu 13.9). Serwer, który już nie ma żadnej wiedzy o istnieniu połączenia, odpowiada wysłaniem segmentu RST.

Zauważmy, że po restarcie host wysyła pakiet *gratuitous ARP* („dobrowolne ARP"; patrz rozdział 4.), aby ustalić, czy jego adres IPv4 jest już używany w segmencie sieci lokalnej, i dostarczyć go innym urządzeniom. Host również pyta o adres MAC dla adresu IPv4 10.0.0.1, ponieważ jest to domyślny router połączenia z Internetem.

13.6.4. TIME_WAIT Assassination (TWA)

Jak już poprzednio wspominaliśmy, stan TIME_WAIT został zamierzony, by umożliwić odrzucenie wszelkich zaległych datagramów pochodzących z zamkniętego połączenia. W czasie pozostawania w tym stanie odczekujący protokół TCP ma zwykle mało do roboty; jedynie podtrzymuje ten stan, dopóki nie otrzyma sygnału od zegara odmierzającego czas 2MSL. Jeśli jednak TCP podczas tego okresu otrzyma pewnego rodzaju segmenty, mówiąc dokładniej — segment RST, może utracić właściwą synchronizację. Nazywa się to TWA (*TIME-WAIT Assassination* — „zamach" na stan TIME-WAIT; patrz [RFC1337]). Rozważmy wymianę pakietów pokazaną na rysunku 13.10.



Rysunek 13.10. *Segment RST może „dokonać zamachu" (assassinate) na stan TIME_WAIT i wymusić przedwczesne zamknięcie połączenia. Istnieją różne metody przeciwstawienia się temu problemowi, łącznie z ignorowaniem segmentów RST w trakcie przebywania w stanie TIME_WAIT*

W przykładzie pokazanym na rysunku 13.10 serwer spełnił już swoją rolę w połączeniu i wyczyścił wszelkie dane o jego stanie. Klient zaś pozostaje w stanie TIME_WAIT. W momencie zakończenia wymiany segmentów FIN następny numer sekwencyjny ma wartość *K* dla klienta i wartość *L* dla serwera. Opóźniony segment został wysłany przez serwer do klienta przy użyciu numeru sekwencyjnego *L*–100 i zawiera numer potwierdzenia *K*–200. Kiedy klient odbiera ten segment, ustala, że wartości numeru sekwencyjnego i potwierdzenia ACK są „stare". W przypadku otrzymania takich starych segmentów TCP reaguje, wysyłając segment ACK z najbardziej aktualnymi wartościami numeru sekwencyjnego i ACK (odpowiednio: *K* i *L*). Jednakże serwer, gdy odbierze ten segment, nie ma jakiejkolwiek informacji o danym połączeniu i dlatego odpowiada wysłaniem segmentu RST. Nie jest to problemem dla serwera, ale powoduje, że klient przedwcześnie wykonuje przejście ze stanu TIME_WAIT do stanu CLOSED. Większość systemów, gdy znajduje się w stanie TIME_WAIT, unika tego problemu, po prostu nie reagując na segmenty RST.

13.7. Działanie serwera TCP

W rozdziale 1. stwierdziliśmy, że większość serwerów TCP działa w trybie równoległym. Kiedy nowe żądanie połączenia dociera do serwera, serwer akceptuje to połączenie i tworzy nowy proces lub wątek przeznaczony do obsługi nowego klienta. W zależności od systemu operacyjnego, dla uruchomienia nowego serwera mogą być przydzielane różne zasoby. Interesuje nas interakcja protokołu TCP z działającymi równolegle serwerami, a szczególnie chcielibyśmy poznać sposób, w jaki serwery TCP używają numerów portów i obsługują wiele równoległych klientów.

13.7.1. Numery portów TCP

Możemy dowiedzieć się, jak protokół TCP obsługuje numery portów, obserwując dowolny serwer TCP. Przypatrzymy się serwerowi bezpiecznej powłoki (sshd), korzystając z polecenia netstat na dwustosowym (*dual-stack*) hoście zdolnym do obsługi obu protokołów: IPv4 i IPv6. Aplikacja sshd implementuje ***protokół bezpiecznej powłoki*** (*Secure Shell Protocol* [RFC4254]), który oferuje szyfrowaną i uwierzytelnianą funkcjonalność zdalnego terminala. Poniższe dane wyjściowe dotyczą systemu bez aktywnych połączeń bezpiecznej powłoki. (Usunęliśmy wszystkie wiersze danych wyjściowych z wyjątkiem wiersza dotyczącego serwera).

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)
Proto Recv-Q Send-Q    Local Address    Foreign Address    State
tcp        0      0           :::22              :::*    LISTEN
```

Opcja -a sprawia, że raport obejmuje wszystkie punkty końcowe sieci, zarówno nasłuchujące, jak i niebędące w stanie nasłuchu. Flaga -n powoduje wyświetlanie adresów IP w formie rozdzielonych kropkami liczb dziesiętnych (lub heksadecymalnych), zamiast próby skorzystania z usługi DNS zamieniającej adres na nazwę, oraz wyświetlanie numerów portów w postaci liczbowej (np. 22) zamiast użycia nazw usług (np. ssh). Opcja -t wybiera tylko punkty końcowe protokołu TCP.

Adres lokalny (który w rzeczywistości oznacza lokalny punkt końcowy) jest wyświetlany jako :::22, co jest właściwym dla protokołu IPv6 sposobem zapisu adresu złożonego z samych zer, zwanego również adresem **wieloznacznym**, razem z numerem portu 22. To oznacza, że każde przychodzące żądanie połączenia (tzn. segment SYN) adresowane do portu 22 zostanie zaakceptowane przez każdy lokalny interfejs. Jeśli host byłby węzłem wieloadresowym (*multihomed*), a nasz przykładowy host ma tę własność, moglibyśmy określić pojedynczy adres IP jako lokalny adres IP (wybierając jeden z adresów IP hosta) i wtedy tylko połączenia odebrane przez ten interfejs byłyby akceptowane. (Zobaczymy przykład tego rozwiązania w dalszej części tego punktu). Port 22 jest dobrze znanym numerem portu zarezerwowanym dla protokołu bezpiecznej powłoki. Rejestr zarezerwowanych numerów portów jest utrzymywany przez organizację IANA (patrz [ITP]).

Adres obcy jest wyświetlany jako :::*, co oznacza wieloznaczność i adresu, i numeru portu (tzn. zapis ten oznacza dowolny punkt końcowy). W tym momencie obcy adres IP i obcy numer portu nie są jeszcze znane, ponieważ lokalny punkt końcowy jest w stanie LISTEN i oczekuje na przyjście połączenia. Teraz uruchamiamy klienta bezpiecznej powłoki na hoście 10.0.0.3, który łączy się z naszym serwerem. Oto istotne wiersze danych wyjściowych polecenia netstat (kolumny Recv-Q i Send-Q, które zawierają tylko wartości zerowe, zostały, dla większej przejrzystości, usunięte):

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)

Proto       Local Address       Foreign Address  State
tcp                 :::22                  :::*  LISTEN
tcp    ::ffff:10.0.0.1:22 ::ffff:10.0.0.3:16137  ESTABLISHED
```

Drugi wiersz zawierający numer portu 22 odnosi się do połączenia w stanie ESTABLISHED. Wszystkie cztery składniki definiujące lokalny i obcy punkt końcowy są dla tego połączenia wypełnione: lokalny adres IP i numer portu oraz obcy adres IP i numer portu. Lokalny adres IP odpowiada interfejsowi, do którego przyszło żądanie połączenia (interfejs Ethernetu, identyfikowany przez swój adres IPv6 mapowany na adres IPv4: ::ffff:10.0.0.1).

Lokalny punkt końcowy znajdujący się w stanie LISTEN jest pozostawiony w spokoju. Jest to ten punkt końcowy, którego używa działający w trybie równoległym serwer do przyjmowania przyszłych żądań połączenia. To moduł TCP w systemie operacyjnym tworzy nowy punkt końcowy w stanie ESTABLISHED, kiedy przychodzące żądanie połączenia zostaje odebrane i zaakceptowane. Zauważmy również, że numer portu dla połączenia w stanie ESTABLISHED nie zmienia się: ma wartość 22, taką samą jak w przypadku punktu końcowego w stanie LISTEN.

Teraz zainicjujemy jeszcze jedno żądanie klienta z tego samego systemu (10.0.0.3) adresowane do naszego serwera. Oto dane wyjściowe polecenia netstat:

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)
Proto       Local Address       Foreign Address   State
tcp                 :::22                  :::*   LISTEN
tcp    ::ffff:10.0.0.1:22 ::ffff:10.0.0.3:16140   ESTABLISHED
tcp    ::ffff:10.0.0.1:22 ::ffff:10.0.0.3:16137   ESTABLISHED
```

Mamy teraz dwa połączenia w stanie ESTABLISHED z tego samego hosta do tego samego serwera. Oba mają numer portu lokalnego na serwerze równy 22. Nie stanowi do dla TCP problemu, ponieważ numery portu obcego różnią się. Muszą być różne, ponieważ każdy z klientów bezpiecznej powłoki używa portu efemerycznego, a port efemeryczny jest z definicji takim portem, który nie jest aktualnie wykorzystywany na danym hoście (10.0.0.3) przez inne połączenie.

W przykładzie pokazujemy raz jeszcze, że TCP demultipleksuje przychodzące segmenty przy użyciu wszystkich czterech wartości, które współtworzą lokalny i obcy punkt końcowy, a mianowicie: docelowego adresu IP, docelowego numeru portu, źródłowego adresu IP oraz źródłowego numeru portu. Protokół TCP nie może określić, który proces ma otrzymać przychodzący segment, patrząc jedynie na numer portu docelowego. Poza tym, jedynym z trzech punktów końcowych używających portu 22, który może odbierać przychodzące żądania połączenia, jest ten w stanie LISTEN. Punkty końcowe w stanie ESTABLISHED nie mogą odbierać segmentów SYN, a punkt końcowy w stanie LISTEN nie może odbierać segmentów z danymi. Zapewnia to system operacyjny hosta. (Jeśliby tego nie robił, protokół TCP mógłby się zupełnie pogubić i nie działać, tak jak należy).

Następnie inicjujemy połączenie ze strony trzeciego klienta, z adresu IP 169.229.62.97, które wymaga użycia łącza DSL PPPoE od strony serwera 10.0.0.1, gdyż nie jest to połączenie w ramach tej samej sieci lokalnej Ethernet. (Na poniższym listingu dla większej przejrzystości usunięto kolumnę Proto, która zawiera zawsze nazwę tcp).

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)
Send-Q          Local Address            Foreign Address  State
     0                  :::22                         :::*  LISTEN
     0      ::ffff:10.0.0.1:22     ::ffff:10.0.0.3:16140  ESTABLISHED
     0      ::ffff:10.0.0.1:22     ::ffff:10.0.0.3:16137  ESTABLISHED
   928 ::ffff:67.125.227.195:22 ::ffff:169.229.62.97:1473  ESTABLISHED
```

Lokalny adres IP trzeciego połączenia w stanie ESTABLISHED odpowiada adresowi interfejsu łącza PPPoE w hoście z wieloma interfejsami sieciowymi (67.125.227.195). Zauważmy, że wartość w kolumnie Send-Q nie jest równa 0, ale wynosi 928 bajtów. Oznacza to, że host serwera wysłał w tym połączeniu 928 bajtów, dla których nie otrzymał jeszcze potwierdzenia.

13.7.2. Ograniczanie lokalnych adresów IP

Możemy sprawdzić, co się stanie, kiedy serwer nie stosuje wieloznacznych lokalnych adresów IP, ale ustala jeden konkretny adres lokalny. Jeśli uruchomimy nasz program sock jako serwer i podamy w wywołaniu jeden konkretny adres IP, adres ten stanie się lokalnym adresem nasłuchującego punktu końcowego, np.:

```
Linux% sock -s 10.0.0.1 8888
```

To ogranicza serwer do obsługiwania tylko tych połączeń, które przychodzą na lokalny adres IPv4 10.0.0.1. Odzwierciedla to wyjście polecenia netstat:

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)
Proto Recv-Q Send-Q  Local Address    Foreign Address     State
tcp        0      0  10.0.0.1:8888         0.0.0.0:*     LISTEN
```

Szczególnie interesujące w tym przykładzie jest to, że nasz program sock wiąże się tylko z lokalnym adresem IPv4 równym 10.0.0.1, więc nasze wyjście polecenia netstat wygląda znacząco inaczej. W poprzednim przykładzie oznaczenia adresu wieloznacznego i numeru portu odnosiły się do obu wersji adresowania IP. W tym jednak przypadku jesteśmy ograniczeni do konkretnego adresu, portu i rodziny adresów (tylko IPv4). Jeśli teraz połączymy się z tym serwerem z sieci lokalnej, z hosta 10.0.0.3, zadziała to doskonale:

```
Linux% netstat -a -n -t
Active Internet connections (servers and established)
Proto Recv-Q Send-Q   Local Address    Foreign Address   State
tcp        0      0   10.0.0.1:8888         0.0.0.0:*   LISTEN
tcp        0      0   10.0.0.1:8888     10.0.0.3:16153  ESTABLISHED
```

Jeśli natomiast spróbujemy połączyć się z naszym serwerem z hosta używającego adresu docelowego innego niż 10.0.0.1 (łącznie z adresem hosta lokalnego 127.0.0.1), żądanie połączenia nie zostanie zaakceptowane przez moduł TCP. Śledząc przesyłane pakiety za pomocą programu tcpdump, widzimy, że wysłanie segmentu SYN powoduje odpowiedź w postaci segmentu RST, co pokazujemy na listingu 13.9.

Listing 13.9. *Odrzucenie żądania połączenia używającego adresu IP hosta lokalnego wysłanego z hosta serwera*

```
1 22:29:19.905593 IP 127.0.0.1.1292 > 127.0.0.1.8888:
      S 591843787:591843787(0) win 32767
      <mss 16396,sackOK,timestamp 3587463952 0,nop,wscale 2>
2 22:29:19.906095 IP 127.0.0.1.8888 > 127.0.0.1.1292:
      R 0:0(0) ack 591843788 win 0
```

Aplikacja serwera nigdy nie zobaczy tego żądania połączenia — jego odrzucenie jest wykonywane przez moduł TCP systemu operacyjnego na podstawie adresu lokalnego określonego przez aplikację i adresu docelowego zawartego w przychodzącym segmencie SYN. Widzimy, że możliwość ograniczania lokalnych adresów IP może działać dość bezwzględnie.

13.7.3. Ograniczanie obcych punktów końcowych

W rozdziale 10. pisaliśmy, że serwer UDP może zwykle określić obcy adres IP i obcy numer portu, oprócz określenia lokalnego adresu IP i lokalnego numeru portu. Funkcje abstrakcyjnego interfejsu opisane w dokumencie [RFC0793] pozwalają serwerowi wykonującemu pasywne otwarcie zarówno na pełną specyfikację obcego punktu końcowego (aby oczekiwać na rozpoczęcie aktywnego otwarcia przez konkretnego klienta), jak i na nieokreślanie obcego punktu końcowego (by czekać na dowolnego klienta).

Niestety, zwykły interfejs gniazd Berkeley nie udostępnia sposobu, aby to zrealizować. Serwer musi pozostawić punkt końcowy klienta jako nieokreślony, poczekać na nadejście połączenia, a następnie sprawdzić adres IP i numer portu klienta. W tabeli 13.3 podsumowujemy trzy typy wiązań adresów, które może ustanowić serwer TCP.

Tabela 13.3. *Opcje wiązań adresów i numerów portu dostępne dla serwera TCP*

Adres lokalny	Obcy adres	Ograniczenie	Uwagi
local_IP.lport	*foraddr.foreign_port*	Jeden klient	Zwykle nieobsługiwana
local_IP.lport	*.*	Jeden lokalny punkt końcowy	Rzadko występująca (używana przez serwery DNS)
*.*local_port*	*.*	Jeden port lokalny	Najczęściej występująca; możliwa obsługa wielu rodzin adresów (IPv4/IPv6)

We wszystkich przypadkach *local_port* jest portem przypisanym do serwera, a *local_IP* musi być adresem IP transmisji pojedynczej używanym przez system lokalny. Uporządkowanie trzech wierszy w tabeli odpowiada porządkowi stosowanemu przez moduł TCP przy próbie ustalenia, który lokalny punkt końcowy jest odbiorcą przychodzącego żądania połączenia. W pierwszej kolejności sprawdzane jest najbardziej specyficzne wiązanie (opcja z pierwszego wiersza, jeśli jest obsługiwana), a na końcu najbardziej ogólne (ostatni wiersz z obydwoma adresami IP zapisanymi z użyciem symboli wieloznacznych). W systemach obsługujących IPv4 i IPv6 (czyli z podwójnym stosem) przestrzeń portów może być wspólna. Zasadniczo polega to na tym, że jeśli serwer dowiąże się do portu, używając adresowania IPv6, automatycznie rezerwuje sobie ten sam port dla protokołu IPv4.

13.7.4. Kolejka połączeń przychodzących

Serwer równoległy uruchamia nowy proces lub wątek do obsługi każdego klienta, tak żeby nasłuchujący serwer był zawsze gotowy do obsługi następnego przychodzącego żądania połączenia. Jest to podstawowa przyczyna używania serwerów równoległych. Ale wciąż istnieje możliwość, że nadejdzie wiele żądań połączeń w czasie, gdy nasłuchujący serwer tworzy nowy proces, lub w czasie, gdy system operacyjny jest zajęty, bo wykonuje inne procesy o wyższym priorytecie, lub gdy, co gorsza, serwer jest atakowany przez fałszywe żądania połączeń, do których całkowitego ustanowienia nigdy się nie dopuszcza. Jak protokół TCP obsługuje takie scenariusze?

Aby w pełni zbadać tę kwestię, musimy najpierw zrozumieć, że nowe połączenia mogą znajdować się w jednym z dwóch różnych stanów, zanim zostaną udostępnione aplikacji. Pierwszy przypadek dotyczy połączeń, których ustanowienie jeszcze nie zakończyło się, ale dla których został odebrany segment SYN (znajdują się one w stanie SYN_RCVD). Drugi przypadek obejmuje połączenia, które już zakończyły uzgadnianie trójetapowe i znajdują się w stanie ESTABLISHED, ale nie zostały jeszcze zaakceptowane przez aplikację. Wewnętrznie system operacyjny zwykle utrzymuje dwie różne kolejki, po jednej dla każdego z wyżej opisanych przypadków.

Aplikacja ma ograniczoną kontrolę nad kształtowaniem rozmiarów tych kolejek. Tradycyjnie przy użyciu interfejsu API gniazd Berkeley wyposażona była tylko w pośrednią kontrolę nad sumą rozmiarów tych dwóch kolejek. We współczesnych jądrach systemu Linux kontrola ta została przeniesiona na liczbę połączeń odpowiadających drugiemu przypadkowi (znajdujących się w stanie ESTABLISHED). Aplikacja może zatem ograniczać liczbę w pełni uformowanych połączeń oczekujących na obsługę z jej strony. W Linuksie wobec tego mają zastosowanie następujące reguły.

- Kiedy przychodzi żądanie połączenia (tzn. segment SYN), sprawdzany jest ogólnosystemowy parametr `net.ipv4.tcp_max_syn_backlog` (domyślna wartość wynosi 1000). Jeśli liczba połączeń znajdujących się w stanie SYN_RCVD miałaby przekroczyć ten próg, przychodzące połączenie jest odrzucane.
- Każdy nasłuchujący punkt końcowy posiada ustalonej długości kolejkę połączeń, które zostały całkowicie zaakceptowane przez TCP (tzn. uzgadnianie trójetapowe zostało przeprowadzone), ale nie zostały jeszcze zaakceptowane przez aplikację. Aplikacja określa limit dla tej kolejki, czyli tzw. *backlog* (termin angielski oznaczający zaległości). Wartość tego limitu musi być zawarta między wartością 0 a określonym dla systemu maksimum (o nazwie `net.core.somaxconn`) włącznie (domyślnie 128).

  Pamiętajmy, że opisywana wartość *backlog* określa tylko maksymalną liczbę zakolejkowanych połączeń odnoszących się do **jednego** nasłuchującego punktu końcowego, z których wszystkie zostały już zaakceptowane przez TCP i oczekują na zaakceptowanie przez aplikację.
- Jeżeli jest wolne miejsce w kolejce danego nasłuchującego punktu końcowego dla tego nowego połączenia, moduł TCP potwierdza przez ACK otrzymanie segmentu SYN i wykonuje otwarcie połączenia. Aplikacja serwera związana z nasłuchującym punktem końcowym nie widzi tego nowego połączenia, dopóki nie zostanie odebrany trzeci segment w ramach uzgadniania trójetapowego. Również klient może uważać, że serwer jest gotowy do odbierania danych, w momencie gdy aktywne otwarcie z jego strony kończy się powodzeniem, zanim jeszcze aplikacja zostanie powiadomiona o nowym połączeniu. Jeżeli to się dzieje, TCP serwera tylko kolejkuje przychodzące dane.
- Jeśli nie wystarcza miejsca w kolejce na nowe połączenie, TCP opóźnia swoją odpowiedź dla odebranego segmentu SYN, aby dać aplikacji szansę na odrobienie zaległości. Linux jest dosyć wyjątkowy w swoim zachowaniu — uparcie stara się nie ignorować przychodzących połączeń, o ile to jest możliwe. Jeżeli jest ustawiona zmienna sterująca systemu `net.ipv4.tcp_abort_on_overflow`, nowe połączenia przychodzące są resetowane przy użyciu segmentu RST.

Wysyłanie segmentu RST przy przepełnieniu nie jest na ogół zalecane i nie jest włączone domyślnie. Klient próbuje nawiązać kontakt z serwerem, a jeśli otrzymuje sygnał resetowania podczas wymiany segmentów SYN, może fałszywie wywnioskować, że nie ma dostępnego serwera (zamiast dojść do wniosku, że istnieje dostępny serwer, ale jest zajęty). Zajętość urządzenia jest formą „miękkiego" lub tymczasowego błędu, a nie błędu trwałego. Zazwyczaj, kiedy kolejka jest pełna, to zajęta jest albo aplikacja, albo system operacyjny, który uniemożliwia aplikacji obsługę przychodzących połączeń. Stan ten może się zmienić w krótkim czasie. Jeśli jednak serwer odpowie wysłaniem segmentu RST, aktywne otwarcie klienta zostanie przerwane (czyli stanie się to, co obserwowaliśmy, gdy serwer nie był uruchomiony). Jeśli resetowanie nie jest stosowane i nasłuchującemu serwerowi nie udaje się zaakceptować przynajmniej części połączeń już zaakceptowanych przez TCP, które zapełniły jego kolejkę, aż do osiągnięcia limitu, aktywne otwarcie klienta w końcu zostaje przeterminowane, zgodnie ze zwykłymi mechanizmami protokołu TCP. W przypadku Linuksa łączące się klienty są jedynie spowalniane przez znaczący okres czasu — nie zostaną ani przeterminowane, ani zresetowane.

Przy użyciu programu sock możemy zobaczyć, co się dzieje, kiedy kolejka połączeń przychodzących staje się pełna. Wywołujemy go z nową opcją (-O), która nakazuje programowi wstrzymanie działania (na określony czas) po utworzeniu nasłuchującego punktu końcowego, a przed przyjęciem jakichkolwiek żądań połączeń. Jeśli następnie w czasie tej przerwy uruchomimy wiele klientów, kolejka zaakceptowanych połączeń serwera powinna się zapełnić i będziemy mogli zobaczyć, co się dzieje; wykorzystamy do tego program tcpdump.

```
Linux% sock -s -v -ql -O30000 6666
```

Opcja -ql ustawia wartość limitu zaległości (*backlog*) dla nasłuchującego punktu końcowego na 1. Opcja -O30000 powoduje, że program zasypia na 30 000 s. (właściwie dość długi czas, ok. 8 godzin), zanim przyjmie jakiekolwiek połączenia od klientów. Jeśli teraz próbujemy raz za razem łączyć się z tym serwerem, pierwsze cztery połączenia są wykonane niezwłocznie. Po czym kolejne dwa połączenia są realizowane co 9 s. Inne systemy operacyjne różnią się znacznie pod względem tej obsługi. Przykładowo w systemach Solaris 8 i FreeBSD 4.7 dwa połączenia są obsługiwane natychmiast, a trzecie zostaje przeterminowane; wszystkie kolejne połączenia zostają również przeterminowane.

Na listingu 13.10 pokazujemy wygenerowane programem tcpdump dane obrazujące klienta łączącego się z serwerem FreeBSD wykonującym program sock z wyżej podanymi argumentami. (Oznaczyliśmy numery portu klienta pogrubioną czcionką od momentu, w którym połączenie TCP jest ustanowione — kiedy kończy się uzgadnianie trójetapowe).

Listing 13.10. *Serwer FreeBSD akceptuje dwa połączenia bez zwłoki. Kolejne połączenia nie otrzymują odpowiedzi i ostatecznie zostają przeterminowane przez klienta*

```
1 21:28:47.399872 IP (tos 0x0, ttl 64, id 46646, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2461 > 169.229.62.97.6666:
      S [tcp sum ok] 2998137201:2998137201(0) win 5808
      <mss 1452,sackOK,timestamp 4102309703 0,nop,wscale 2>

2 21:28:47.413770 IP (tos 0x0, ttl 47, id 6876, offset 0,
      flags [DF], proto 6, length: 60)
      169.229.62.97.6666 > 63.203.76.212.2461:
      S [tcp sum ok] 5583769:5583769(0) ack 2998137202 win 1460
      <mss 1412,nop,wscale 0,nop,nop,timestamp 219082980 4102309703>

3 21:28:47.414058 IP (tos 0x0, ttl 64, id 46648, offset 0,
      flags [DF], proto 6, length: 52)
      63.203.76.212.2461 > 169.229.62.97.6666:
      . [tcp sum ok] 1:1(0) ack 1 win 1452
      <nop,nop,timestamp 4102309717 219082980>

4 21:28:47.423673 IP (tos 0x0, ttl 64, id 19651, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2462 > 169.229.62.97.6666:
      S [tcp sum ok] 2996964252:2996964252(0) win 5808
      <mss 1452,sackOK,timestamp 4102309727 0,nop,wscale 2>

5 21:28:47.436897 IP (tos 0x0, ttl 47, id 26581, offset 0,
      flags [DF], proto 6, length: 60)
```

```
      169.229.62.97.6666 > 63.203.76.212.2462:
      S [tcp sum ok] 3761536245:3761536245(0) ack 2996964253 win 1460
      <mss 1412,nop,wscale 0,nop,nop,timestamp 219082983 4102309727>

6 21:28:47.437186 IP (tos 0x0, ttl 64, id 19653, offset 0,
      flags [DF], proto 6, length: 52)
      63.203.76.212.2462 > 169.229.62.97.6666:
      . [tcp sum ok] 1:1(0) ack 1 win 1452
      <nop,nop,timestamp 4102309741 219082983>

7 21:28:47.446198 IP (tos 0x0, ttl 64, id 24292, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102309749 0,nop,wscale 2>

8 21:28:50.445771 IP (tos 0x0, ttl 64, id 24294, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102312750 0,nop,wscale 2>

9 21:28:56.444900 IP (tos 0x0, ttl 64, id 24296, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102318750 0,nop,wscale 2>

10 21:29:08.443031 IP (tos 0x0, ttl 64, id 24298, offset 0,
      flags [DF], proto 6, length: 60) 6
      3.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102330750 0,nop,wscale 2>

11 21:29:32.439406 IP (tos 0x0, ttl 64, id 24300, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102354750 0,nop,wscale 2>

12 21:30:20.432118 IP (tos 0x0, ttl 64, id 24302, offset 0,
      flags [DF], proto 6, length: 60)
      63.203.76.212.2463 > 169.229.62.97.6666:
      S [tcp sum ok] 2991331729:2991331729(0) win 5808
      <mss 1452,sackOK,timestamp 4102402750 0,nop,wscale 2>
```

Pierwsze żądanie połączenia od klienta z portu 2461 zostaje zaakceptowane przez TCP (segmenty od 1. do 3.). Drugie żądanie połączenia wysłane przez klienta z portu numer 2462 jest także zaakceptowane przez TCP (segmenty od 4. do 6.). Aplikacja serwera jest wciąż w stanie uśpienia i nie zaakceptowała jeszcze żadnego połączenia. Dotychczas wszystkie działania zostały wykonane przez moduł TCP systemu operacyjnego. Poza tym, dwa klienty wykonały z powodzeniem swoje aktywne otwarcia, ponieważ uzgadnianie trójetapowe zostało w ich przypadku zrealizowane.

Próbujemy uruchomić trzecie połączenie, dla którego żądanie SYN pojawia się jako segment numer 7. (port 2463), ale TCP po stronie serwera ignoruje segmenty SYN, ponieważ kolejka dla tego nasłuchującego punktu końcowego jest pełna. Klient retransmituje żądanie SYN w segmentach od 8. do 12. przy użyciu procedury binarnego odczekiwania wykładniczego. W systemach FreeBSD i Solaris TCP ignoruje przychodzące segmenty SYN, gdy kolejka jest pełna.

Przypomnijmy sobie, że TCP akceptuje przychodzące żądanie połączenia (tzn. segment SYN), jeśli jest wolne miejsce w kolejce nasłuchującego serwera, nie dając aplikacji możliwości sprawdzenia, skąd ono pochodzi (czyli sprawdzenia źródłowego adresu IP i numeru portu źródłowego). To nie wynika z wymagań protokołu TCP; jest to tylko powszechnie stosowana technika implementacyjna (tzn. sposób, w jaki funkcjonują gniazda Berkeley). Jeśli byłby używany alternatywny dla gniazd Berkeley interfejs API (np. TLI/XTI), aplikacja mogłaby otrzymać sposób na to, by się dowiedzieć o przychodzącym żądaniu połączenia i następnie zadecydować, czy zaakceptować to połączenie, czy też nie. Jeśli nawet, teoretycznie, interfejs TLI oferował tę możliwość, nigdy się w pełni nie przyjął, więc praktycznie pozostaje nam tylko interfejs TCP dostarczany przez gniazda Berkeley.

Więc w przypadku gniazd Berkeley i protokołu TCP musimy być świadomi, że kiedy aplikacja zostaje powiadomiona, że właśnie przyszło połączenie, uzgadnianie trójetapowe **już się zakończyło**. Ten sposób działania oznacza również, że serwer TCP nie posiada sposobu, by sprawić, aby aktywne otwarcie klienta zakończyło się niepowodzeniem. Kiedy nowe połączenie z klientem jest przekazywane do aplikacji serwera, uzgadnianie trójetapowe protokołu TCP już się zakończyło, a otwarcie aktywne wykonywane przez klienta osiągnęło sukces. Jeżeli potem serwer sprawdza adres IP oraz numer portu i podejmuje decyzję, że nie chce obsługiwać tego klienta, wszystko, co może zrobić, to zamknąć połączenie (powodując wysłanie segmentu FIN) lub je zresetować (powodując wysłanie segmentu RST). W każdym przypadku klient, sądząc po pomyślnym wykonaniu aktywnego otwarcia, że wszystko jest w porządku, mógł już wysłać swoje żądanie do serwera. Mogą zostać zaimplementowane inne protokoły warstwy transportowej (w rozumieniu modelu OSI), które umożliwią aplikacji oddzielenie nadejścia połączenia od jego akceptacji, ale TCP tego nie zapewnia.

13.8. Ataki związane z zarządzaniem połączeniem TCP

Przepełnienie bufora pakietami typu SYN (*SYN flood*) jest ukierunkowanym na TCP atakiem typu DoS, podczas którego złośliwe klienty generują serię żądań nawiązania połączenia TCP (czyli segmentów SYN) i wysyłają je do serwera często z fałszywymi (np. wybranymi losowo) źródłowymi adresami IP. Serwer przydziela pewną ilość zasobów każdemu częściowemu połączeniu. Ponieważ te połączenia nie są nigdy do końca ustanawiane, serwer może zacząć odmawiać świadczenia usługi przyszłym uprawnionym połączeniom, bo jego pamięć wyczerpała się (przechowuje stan licznych częściowo otwartych połączeń).

Atak ten jest dość trudny do odparcia, ponieważ nie zawsze łatwo odróżnić właściwe próby nawiązania połączenia od ataków typu *SYN flood*. Jeden z mechanizmów wymyślonych w celu poradzenia sobie z tym problemem nazywa się ***SYN cookies*** (patrz

[RFC4987]). Główne spostrzeżenie będące u podstaw wykorzystania *SYN cookies* jest takie, że większość informacji o połączeniu, które byłyby zapamiętane w momencie nadejścia segmentu SYN, można by zakodować w polu *Numer sekwencyjny* przekazywanym w segmencie SYN+ACK. Komputer docelowy korzystający z *SYN cookies* nie potrzebuje przydzielać żadnej pamięci dla przychodzącego żądania połączenia — przydziela on rzeczywistą pamięć dopiero wtedy, kiedy sam segment SYN+ACK zostanie potwierdzony (i zwrócony początkowy numer sekwencyjny). Wówczas wszystkie niezbędne parametry połączenia mogą zostać odtworzone, a połączenie przeniesione do stanu ESTABLISHED.

Tworzenie *SYN cookies* wiąże się ze staranną procedurą wyboru numeru ISN przez protokół TCP serwerów. Przede wszystkim serwer musi zakodować wszystkie istotne dane o stanie połączenia w polu *Numer sekwencyjny* w wysyłanym przez siebie segmencie SYN+ACK, którego wartość zostanie zwrócona w polu *Numer ACK* przez właściwego klienta. Istnieje kilka sposobów realizacji tego zadania, lecz opiszemy pokrótce technikę przyjętą przez system Linux.

Serwer odbierający przychodzące żądanie SYN sprawia, że jego numerowi ISN (który zostanie dostarczony do klienta w segmencie SYN+ACK) zostaje przypisana wartość utworzona w następujący sposób: 5 najwyższych bitów zawiera wartość (t mod 32), gdzie t oznacza 32-bitowy licznik, który zwiększa swą zawartość o 1 co 64 s, następne 3 bity zawierają zakodowaną wartość parametru MSS serwera (jedną z ośmiu możliwych wartości), a pozostałe 24 bity to wartość wybranego przez serwer skrótu kryptograficznego utworzonego na podstawie 4-członowego identyfikatora połączenia i wartości t. (Szczegółowe wyjaśnienie skrótów kryptograficznych można znaleźć w rozdziale 18.).

Kiedy używane są *SYN cookies*, serwer zawsze odpowiada, wysyłając segment SYN+ACK (tak jak przy każdym typowym ustanowieniu połączenia TCP), i jest w stanie odtworzyć swoją kolejkę przychodzących żądań SYN, kiedy otrzyma prawidłowe potwierdzenie ACK, gdzie wartość t nadal generuje ten sam wynik funkcji skrótu kryptograficznego. Istnieją co najmniej dwa problemy związane z tym podejściem. Po pierwsze, procedura ta nie pozwala na używanie segmentów dowolnego rozmiaru z powodu przyjętego sposobu kodowania wartości MSS. Po drugie, co jest o wiele mniej poważnym problemem, cykle ustanawiania połączeń, które są bardzo długie (dłuższe niż 64 s), nie działają poprawnie ze względu na możliwość przepełnienia licznika. Z tych powodów funkcja ta nie jest włączona domyślnie.

Inny typ ataku ukierunkowany na obniżenie jakości usługi TCP jest związany z procedurą PMTUD. W tym przypadku napastnik fabrykuje komunikat PTB protokołu ICMP zawierający bardzo małą wartość MTU (np. 68 bajtów). To zmusza zaatakowany protokół TCP do podjęcia próby wpasowania swoich danych w bardzo małe pakiety, co zmniejsza znacznie jego wydajność. Problem ten może być rozwiązywany na wiele sposobów. Najbardziej brutalną metodą byłoby po prostu wyłączenie PMTUD dla hosta. Inną opcją byłoby wyłączenie PMTUD tylko w przypadkach, gdy odebrany zostanie komunikat PTB protokołu ICMP z wartością MTU następnego przeskoku poniżej 576 bajtów. Jeszcze inna opcja, zaimplementowana przez system Linux i opisana krótko już wcześniej, polega na obstawaniu, że minimalny rozmiar pakietu (dla dużych pakietów używanych przez TCP) jest zawsze ustalany na pewną stałą wartość, a większe pakiety nie mają po prostu włączonego bitu DF w nagłówku IPv4. To podejście jest podobne, choć być może nieco bardziej atrakcyjne niż całkowite wyłączenie PMTUD.

Inny typ ataku polega na dezorganizacji istniejącego połączenia TCP i ewentualnym jego przejęciu (*hijacking* — porwanie). Te formy ataku zwykle zawierają pierwszy krok polegający na „desynchronizacji" dwóch punktów końcowych połączenia TCP, tak aby przesyłając sobie nawzajem komunikaty, używały niewłaściwych numerów sekwencyjnych. Są to konkretne przykłady **ataków z użyciem numerów sekwencyjnych** (patrz [RFC1948]). Mogą być one przeprowadzone na co najmniej dwa sposoby: przez spowodowanie nieprawidłowych przejść między stanami podczas ustanawiania połączenia (podobnie jak TWA; patrz punkt 13.6.4) i przez generowanie dodatkowych danych, gdy połączenie jest w stanie ESTABLISHED. Kiedy już dwa punkty końcowe nie mogą dłużej komunikować się (ale sądzą, że mają otwarte połączenie między sobą), napastnik może wprowadzić do połączenia nową komunikację, która będzie uważana (przynajmniej przez TCP) za prawidłową.

Zespół ataków zwanych ogólnie **atakami z podszywaniem się** (*spoofing attacks*) opiera się na wykorzystaniu segmentów TCP, które zostały specjalnie zaprojektowane przez napastnika, by zdezorganizować lub zmienić funkcjonowanie istniejącego połączenia TCP. Różne rodzaje tych ataków oraz techniki łagodzenia ich skutków są analizowane w dokumencie [RFC4953]. Napastnik może wygenerować fałszywy segment resetujący i wysłać go do istniejącego punktu końcowego TCP. Jeśli tylko 4-członowy identyfikator połączenia i suma kontrolna są poprawne, a numer sekwencyjny mieści się w zakresie, przesłanie żądania resetowania zwykle skutkuje przerwaniem połączenia w każdym z punktów końcowych. Jest to rosnący problem, ponieważ, w miarę jak sieci stają się coraz szybsze, coraz większy zakres numerów sekwencyjnych jest uważany za „mieszczący się w oknie", aby utrzymać wysoką wydajność (patrz rozdział 15.). Inne rodzaje segmentów (segmenty SYN, nawet segmenty ACK) mogą również być podrabiane, (i w połączeniu z atakami zalewania) wywołując mnóstwo problemów. Techniki łagodzenia skutków zawierają uwierzytelnianie każdego segmentu (np. przy użyciu opcji TCP-AO), wymaganie, aby segmenty z żądaniem resetowania posiadały jedną określoną wartość numeru sekwencyjnego, a nie tylko mieszczącą się w określonym zakresie, wymaganie konkretnych wartości w opcji *Znaczniki czasu* i używanie innych form *cookies*, w których wartości skądinąd niekrytycznych danych są celowo uzależnione od dokładniejszej wiedzy o połączeniu lub od sekretnej wartości.

Istnieją ataki z podszywaniem się, które nie należą do obszaru objętego protokołem TCP, jednak mogą mieć negatywny wpływ na działanie TCP. Przykładowo protokół ICMP może zostać wykorzystany do modyfikacji zachowania procedury PMTUD. Może być także wykorzystany do zasygnalizowania, że port lub host są niedostępne, a to często powoduje zakończenie połączenia TCP. Wiele z tych ataków jest opisanych w dokumencie [RFC5927], który także proponuje szereg sposobów poprawienia odporności na podrobione komunikaty ICMP. Proponowane rozwiązania obejmują walidację nie tylko samego komunikatu ICMP, ale również tak dużej części zawartego w nim segmentu TCP, jak to tylko możliwe. Przykładowo zawarty w komunikacie segment powinien posiadać poprawny 4-członowy identyfikator i numer sekwencyjny.

13.9. Podsumowanie

Zanim dwa procesy będą mogły wymieniać dane przy użyciu protokołu TCP, muszą ustanowić między sobą połączenie. Kiedy wymiana danych zostanie zrealizowana, uczestniczące w niej procesy kończą połączenie. W tym rozdziale przyjrzeliśmy się szczegółowo temu, jak połączenia są ustanawiane przy użyciu procedury uzgadniania trójetapowego i jak są kończone przy wykorzystaniu czterech segmentów. Dowiedzieliśmy się również, jak TCP może przeprowadzić operacje jednoczesnego otwarcia oraz jednoczesnego zamknięcia i w jaki sposób są obsługiwane różne opcje, takie jak *Selektywne ACK*, *Znaczniki czasu*, MSS, TCP-AO i UTO.

Korzystaliśmy z programów tcpdump i Wireshark, aby pokazać zachowanie protokołu TCP i sposób wykorzystania pól nagłówka TCP. Dowiedzieliśmy się również, jak ustanowienie połączenia TCP może ulec przeterminowaniu, w jaki sposób są wysyłane i interpretowane żądania zresetowania połączenia, co się dzieje w przypadku połączenia częściowo otwartego i jak TCP umożliwia częściowe zamknięcie połączenia. Protokół TCP ogranicza zarówno liczbę podejmowanych prób nawiązania połączenia przy wykonywaniu otwarcia aktywnego, jak i liczbę prób połączenia obsługiwanych po wykonaniu otwarcia pasywnego.

Fundamentalne znaczenie dla zrozumienia działania protokołu TCP ma jego diagram stanów. Prześledziliśmy kroki związane z ustanawianiem i kończeniem połączenia i zachodzące przejścia między stanami. Przyjrzeliśmy się również, jak sposób ustanawiania połączenia przez TCP wpływa na projektowanie równoległych serwerów TCP.

Połączenie TCP jest jednoznacznie definiowane przez czwórkę wartości: lokalny adres IP, lokalny numer portu, obcy adres IP i obcy numer portu. Zawsze, kiedy połączenie jest kończone, jedna ze stron musi utrzymywać wiedzę o tym połączeniu; zobaczyliśmy, jak to zadanie obsługuje stan TIME_WAIT. Reguła polega na tym, że strona, która wykonuje zamknięcie aktywne, wchodzi w ten stan na czas równy podwójnej wartości parametru MSL dla danej implementacji, co pomaga w zapobieganiu przetwarzania przez TCP segmentów pochodzących ze starszej instancji tego samego połączenia. Użycie opcji *Znaczniki czasu* może skrócić czas odczekiwania w przypadku, gdy nowe połączenia próbują użyć tego samego 4-członowego identyfikatora, niesie także inne korzyści, bo pomaga w wykrywaniu zdublowanych numerów sekwencyjnych i wykonywaniu lepszych pomiarów czasu RTT.

Protokół TCP może być wrażliwy i na wyczerpanie się zasobów, i na ataki z podszywaniem się, ale opracowano szereg metod, by przeciwstawić się takim problemom. W dodatku na działanie TCP mogą negatywnie wpływać problemy związane z innymi protokołami, takimi jak ICMP. W przypadku protokołu ICMP jest możliwa dodatkowa ochrona poprzez dokładną analizę oryginalnych datagramów zwracanych przez komunikaty ICMP. Na koniec, TCP może być używany w połączeniu z innymi protokołami, które zapewniają bezpieczeństwo na poziomie innych warstw (np. IPsec i TLS/SSL, opisane w rozdziale 18.), co jest obecnie standardową praktyką.

13.10. Bibliografia

[CERTISN] *http://www.cert.org/advisories/CA-2001-09.html*

[ITP] *http://www.iana.org/assignments/service-names-port-numbers*

[LS10] Luckie M., Stasiewicz B., *Measuring Path MTU Discovery Behavior*, Proc. ACM IMC, listopad 2010.

[RFC0793] Postel J., *Transmission Control Protocol*, Internet RFC 0793/STD 0007, wrzesień 1981.

[RFC0854] Postel J., Reynolds J.K., *Telnet Protocol Specification*, Internet RFC 0854/STD 0008, maj 1983.

[RFC0879] Postel J., *The TCP Maximum Segment Size and Related Topics*, Internet RFC 0879, listopad 1983.

[RFC1025] Postel J., *TCP and IP Bake Off*, Internet RFC 1025, wrzesień 1987.

[RFC1122] Braden R., ed., *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1191] Mogul J.C., Deering S.E., *Path MTU Discovery*, Internet RFC 1191, listopad 1990.

[RFC1323] Jacobson V., Braden R., Borman D., *TCP Extensions for High Performance*, Internet RFC 1323, maj 1992.

[RFC1337] Braden R., *TIME-WAIT Assassination Hazards in TCP*, Internet RFC 1337 (informational), maj 1992.

[RFC1948] Bellovin S., *Defending against Sequence Number Attacks*, Internet RFC 1948 (informational), maj1996.

[RFC1981] McCann J., Deering S., Mogul J., *Path MTU Discovery for IP Version 6*, Internet RFC 1981, sierpień 1996.

[RFC2018] Mathis M., Mahdavi J., Floyd S., Romanow A., *TCP Selective Acknowledgment Options*, Internet RFC 2018, październik 1996.

[RFC2385] Heffernan A., *Protection of BGP Sessions via the TCP MD5 Signature Option*, Internet RFC 2385 (obsolete), sierpień 1998.

[RFC2675] Borman D., Deering S., Hinden R., *IPv6 Jumbograms*, Internet RFC 2675, sierpień 1999.

[RFC2883] Floyd S., Mahdavi J., Mathis M., Podolsky M., *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, Internet RFC 2883, lipiec 2000.

[RFC2923] Lahey K., *TCP Problems with Path MTU Discovery*, Internet RFC 2923 (informational), wrzesień 2000.

[RFC4254] Ylonen T., Lonvick C., ed., *The Secure Shell (SSH) Connection Protocol*, Internet RFC 4254, styczeń 2006.

[RFC4727] Fenner B., *Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers*, Internet RFC 4727, listopad 2006.

[RFC4821] Mathis M., Heffner J., *Packetization Layer Path MTU Discovery*, Internet RFC 4821, marzec 2007.

[RFC4953] Touch J., *Defending TCP against Spoofing Attacks*, Internet RFC 4953 (informational), lipiec 2007.

[RFC4987] Eddy W., *TCP SYN Flooding Attacks and Common Mitigations*, Internet RFC 4987 (informational), sierpień 2007.

[RFC5482] Eggert L., Gont F., *TCP User Timeout Option*, Internet RFC 5482, marzec 2009.

[RFC5925] Touch J., Mankin A., Bonica R., *The TCP Authentication Option*, Internet RFC 5925, czerwiec 2010.

[RFC5926] Lebovitz G., Rescorla E., *Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)*, Internet RFC 5926, czerwiec 2010.

[RFC5927] Gont F., *ICMP Attacks against TCP*, Internet RFC 5927 (informational), lipiec 2010.

[RFC5961] Ramaiah A., Stewart R., Dalal M., *Improving TCP's Robustness to Blind In-Window Attacks*, Internet RFC 5961, sierpień 2010.

[RFC6056] Larsen M., Gont F., *Recommendations for Transport-Protocol Port Randomization*, Internet RFC 6056/BCP 0156, styczeń 2011.

[RFC6146] Bagnulo M., Matthews P., van Beijnum I., *Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers*, Internet RFC 6146, kwiecień 2011.

[RFC6191] Gont F., *Reducing the TIME-WAIT State Using TCP Timestamps*, Internet RFC 6191/BCP 0159, kwiecień 2011.

[RFC6298] Paxson V., Allman M., Chu J., Sargent M., *Computing TCP'sRetransmission Timer*, Internet RFC 6298, czerwiec 2011.

[S96] Schneier B., *Applied Cryptography*, John Wiley & Sons, 1996.

[TPARAMS] *http://www.iana.org/assignments/tcp-parameters/tcp-arameters.xml*

Rozdział 14. Przeterminowanie i retransmisja w TCP

14.1. Wprowadzenie

Efektywność i wydajność to zagadnienia, których analizie nie poświęciliśmy dotychczas zbyt wiele miejsca. Byliśmy przede wszystkim zainteresowani poprawnością działania. W tym i dwóch następnych rozdziałach będziemy koncentrować się nie tylko na podstawowych zadaniach protokołu TCP, ale również na tym, jak sprawnie on je wykonuje. Protokół TCP zapewnia niezawodną usługę dostarczania danych między dwoma aplikacjami, wykorzystując niżej położoną warstwę sieciową (IP), która może gubić lub powielać pakiety oraz zmieniać ich kolejność. Aby zapewnić wolną od błędów wymianę danych, TCP ponownie wysyła dane, które uważa za utracone. Przy decydowaniu, które dane należy przesłać ponownie, TCP polega na nieustannym przepływie potwierdzeń od odbiorcy do nadawcy. Kiedy dochodzi do utraty segmentów z danymi lub potwierdzeń w trakcie transmisji, TCP inicjuje **retransmisję** danych, które nie zostały potwierdzone. Protokół TCP zawiera dwa oddzielne mechanizmy do wykonywania retransmisji — jeden oparty na czasie, drugi wykorzystujący strukturę potwierdzeń. To drugie podejście jest zwykle o wiele bardziej efektywne niż pierwsze.

W momencie wysyłania danych TCP ustawia licznik czasu, a jeśli dane nie zostaną potwierdzone, zanim licznik czasu skończy odliczanie, dochodzi do tzw. przeterminowania (*timeout*), czyli do retransmisji na podstawie licznika czasu (*timer-based retransmission*). Przeterminowanie następuje po upływie okresu czasu nazwanego **czasem oczekiwania na retransmisję** (**RTO**; *Retransmission Timeout*). TCP wyposażono także w inny sposób inicjowania retransmisji zwany **szybką retransmisją** (*fast retransmission* lub *fast retransmit*), w przypadku której nie stosuje się zwykle żadnego opóźnienia. Szybka retransmisja jest oparta na wywnioskowaniu o utracie danych po odnotowaniu, że numery kumulatywnych potwierdzeń w odbieranych w ciągu czasu segmentach ACK nie zwiększają się lub że odebrane segmenty ACK zawierające selektywne potwierdzenia (opcje SACK) wskazują na obecność u odbiorcy segmentów dostarczonych poza kolejnością. Ujmując to ogólnie: kiedy nadawca uważa, że odbiorcy może brakować pewnych danych, musi wybrać między wysłaniem nowych (dotąd niewysyłanych) danych a retransmisją. W tym rozdziale przyjrzymy się dokładnie, jak TCP ustala, że segment został utracony i co należy przesłać w odpowiedzi. Problem, jak **dużo** należy przesłać, od-

kładamy do rozdziału 16., gdzie omawiamy procedury kontroli przeciążenia, które są zwykle uruchamiane, gdy istnieje podejrzenie utraty pakietu. W bieżącym rozdziale zbadamy, jak czas RTO jest ustalany w oparciu o pomiary czasu „podróży w obie strony", czyli RTT, jak funkcjonuje mechanika retransmisji z wykorzystaniem licznika czasu (*timer-based retransmission*) i jak działa mechanizm szybkiej retransmisji. Przyglądniemy się również temu, jak wykorzystywane są opcje SACK, aby pomóc nadawcy w ustaleniu, których danych brakuje u odbiorcy, jaki wpływ na zachowanie protokołu TCP ma zmiana kolejności i powielanie pakietów IP oraz w jaki sposób TCP może zmienić rozmiar pakietu przy retransmisji. Dokonamy także krótkiego przeglądu niektórych ataków, jakie mogą zostać przeprowadzone w celu sprowokowania bardziej agresywnego albo bardziej pasywnego zachowania protokołu TCP.

14.2. Prosty przykład przeterminowania i retransmisji

Już poznaliśmy kilka przykładów przeterminowania i retransmisji. (1) W przykładzie dotyczącym komunikatu *Destination Unreachable* (miejsce przeznaczenia nieosiągalne) z kodem błędu *Port Unreachable* (port nieosiągalny) w rozdziale 8. widzieliśmy, jak klient protokołu TFTP, korzystając z protokołu UDP, stosował prostą (i słabą) strategię retransmisji po przeterminowaniu: założył, że okres 5 s stanowi odpowiedni limit czasu i przeprowadzał retransmisję co 5 s. (2) Wykonując próbę połączenia z nieistniejącym hostem w rozdziale 13., widzieliśmy, że protokół TCP, kiedy próbował ustanowić połączenie, retransmitował swój segment SYN przy użyciu coraz dłuższych opóźnień między kolejnymi retransmisjami. (3) W rozdziale 3. zobaczyliśmy, co się dzieje, kiedy Ethernet napotyka kolizję. Wszystkie te mechanizmy są inicjowane wyzerowaniem się licznika czasu.

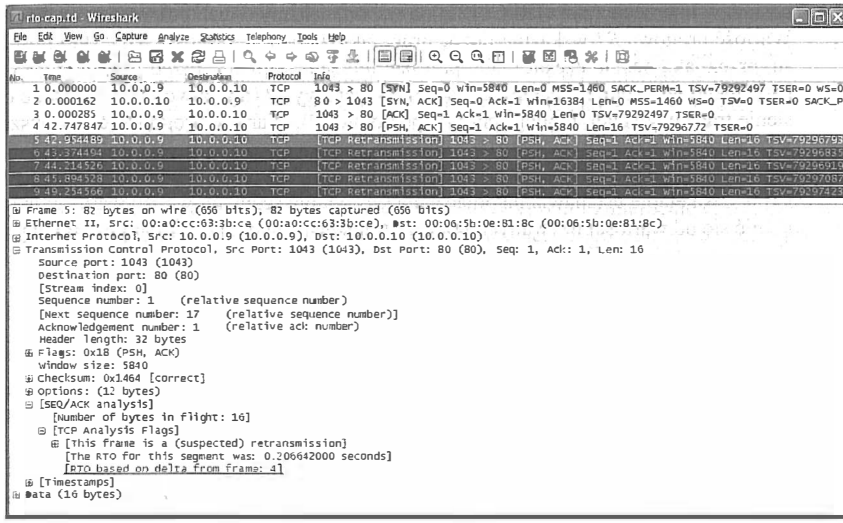
Najpierw przyjrzymy się używanej przez TCP strategii opartej na zastosowaniu licznika czasu. Ustanowimy połączenie, prześlemy nieco danych, aby sprawdzić, że wszystko jest w porządku, odłączymy jeden koniec połączenia, prześlemy jeszcze trochę danych i zobaczymy, co zrobi TCP. W tym przypadku użyjemy programu Wireshark, aby zaobserwować, jak przebiega połączenie (patrz rysunek 14.1).

Segmenty 1., 2. i 3. odpowiadają zwykłej procedurze ustanawiania połączenia poprzez uzgadnianie trójetapowe, stosowanej przez protokół TCP. Kiedy serwer WWW kończy ustanawianie połączenia, nie generuje już żadnego ruchu w sieci, oczekując na żądanie strony WWW. Przed przesłaniem tego żądania odłączamy host serwera od sieci. Na wejściu klienta wprowadzamy następujące polecenia:

```
Linux% telnet 10.0.0.10 80
Trying 10.0.0.10...
Connected to 10.0.0.10.
Escape character is '^]'.
GET / HTTP/1.0
Connection closed by foreign host.
```

Żądanie klienta nie może być dostarczone do serwera, więc pozostaje w kolejce TCP u klienta przez dość długi czas. W ciągu tego okresu polecenie netstat w systemie klienta pokazuje, że kolejka nie jest pusta:

```
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q  Local Address       Foreign Address State
tcp        0     18  10.0.0.9:1043       10.0.0.10:www   ESTABLISHED
```



Rysunek 14.1. *Prosty przykład używanego przez protokół TCP mechanizmu retransmisji po przeterminowaniu. Pierwsza retransmisja ma miejsce w czasie 42,954 s, po niej następują kolejne retransmisje w czasach 43,374 s, 44,215 s, 45,895 s i 49,255 s. Odstępy czasowe między kolejnymi retransmisjami wynoszą odpowiednio 206 ms, 420 ms, 841 ms, 1,68 s i 3,36 ms. Czasy te pokazują podwajanie limitu czasu między kolejnymi retransmisjami tego samego segmentu*

Widzimy tu, że w kolejce danych do wysłania znajduje się 18 bajtów, oczekujących na dostarczenie do serwera WWW. Te 18 bajtów stanowią znaki wprowadzonego poprzednio (i widocznego na listingu) polecenia GET oraz dwie pary znaków powrotu karetki i nowego wiersza. Szczegóły pozostałych danych wyświetlonych poleceniem netstat, w tym informacje o adresach i stanie połączenia, zostaną opisane w następnych akapitach.

Segment 4. stanowi pierwszą próbę wysłania żądania strony WWW, w czasie 42,748 s. Następna próba ma miejsce w czasie 42,954 s, tzn. 0,206 s później. Potem klient podejmuje kolejną próbę w czasie 43,374, a więc 0,420 s później. Następne ponowienia próby przesłania danych (retransmisje) występują w czasach 44,215 s, 45,895 s i 49,255 s. Różnice między tymi czasami wynoszą odpowiednio 0,841 s, 1,680 s i 3,360 s.

To podwajanie czasu między kolejnymi retransmisjami nazywa się **binarnym odczekiwaniem wykładniczym** (*binary exponential backoff*), a spotkaliśmy się z nim w rozdziale 13., analizując nieudaną próbę ustanowienia połączenia TCP. Później zagadnienie to zbadamy bardziej szczegółowo. Jeśli zmierzymy czas, jaki upłynął od początkowego żądania do momentu, w którym połączenie zostało ostatecznie przerwane, otrzymamy w sumie ok. 15,5 minuty. Po upływie tego czasu na ekranie klienta zostaje wyświetlony następujący komunikat błędu:

```
Connection closed by foreign host.
```

Pod względem logicznym TCP ma do dyspozycji dwie wartości progowe służące do określenia, jak długo ma podtrzymywać próby ponownego wysyłania tego samego segmentu. Te dwa progi są opisane w dokumencie RFC specyfikującym wymagania dla

hostów w Internecie (patrz [RFC1122]), a wspomnieliśmy o nich pokrótce w rozdziale 13. Wartość progowa *R1* wskazuje liczbę prób, jakie podejmie protokół TCP (lub ilość czasu, przez jaką będzie czekać), aby przesłać ten sam segment, przed przekazaniem „negatywnego wskazania" (*negative advice*) do warstwy IP (np. powodując ponowne określenie trasy używanej w połączeniu na poziomie IP). Wartość progowa *R2* (większa niż *R1*) dyktuje moment, w którym TCP powinno zrezygnować i przerwać połączenie. Sugeruje się, aby minimalne wartości tych progów wynosiły odpowiednio trzy retransmisje i 100 s. Na etapie ustanawiania połączenia (wysyłania segmentów SYN) wartości te mogą różnić się od wartości używanych podczas transmisji segmentów z danymi, np. wymaga się, aby wartość *R2* dla segmentów SYN wynosiła co najmniej 3 minuty.

W Linuksie wartości *R1* i *R2* dla zwykłych segmentów z danymi są dostępne do zmiany dla aplikacji, mogą też być zmienione przez wykorzystanie zmiennych konfiguracyjnych systemu, odpowiednio: `net.ipv4.tcp_retries1` i `net.ipv4.tcp_retries2`. W tym przypadku miarą obu wartości jest liczba retransmisji, a nie jednostki czasu. Wartość domyślna parametru `net.ipv4.tcp_retries2` wynosi 15, co odpowiada z grubsza 13 – 30 minutom, w zależności od czasu RTO połączenia. Wartość domyślna parametru `net.ipv4.tcp_retries1` wynosi 3. W przypadku segmentów SYN parametry `net.ipv4.tcp_syn_retries` i `net.ipv4.tcp_synack_retries` ograniczają liczbę retransmisji segmentów SYN; ich wartość domyślna wynosi 5 (w przybliżeniu 180 s). System Windows również utrzymuje pewną liczbę zmiennych, które mają wpływ na całościowe zachowanie protokołu TCP, w tym na wartości *R1* i *R2*. Wszystkie są dostępne poprzez modyfikację wartości następujących kluczy rejestru (patrz [WINREG]):

```
HKLM\System\CurrentControlSet\Services\Tcpip\Parameters
HKLM\System\CurrentControlSet\Services\Tcpip6\Parameters
```

Na bezzwłoczne zainteresowanie zasługuje wartość kryjąca się pod nazwą `TcpMaxData` `↪Retransmissions`. Odpowiada ona wartości `tcp_retries2` w Linuksie i ma wartość domyślną 5. Nawet w prostym przykładzie retransmisji, z którym dotychczas mieliśmy do czynienia, wymaga się, aby protokół TCP przypisał pewien limit czasu do zegara retransmisji, który podyktuje, jak długo należy oczekiwać na potwierdzenie ACK dla wysłanych danych. Gdyby protokół TCP był wykorzystywany w jednym statycznym środowisku, możliwe byłoby określenie jednej konkretnej poprawnej wartości dla limitu czasu. Ponieważ TCP musi funkcjonować w bardzo różniących się środowiskach, które ponadto zmieniają się w czasie, protokół musi ustalać wartość limitu czasu w oparciu o bieżącą sytuację. Jeśli np. doszłoby do awarii jakiegoś łącza w sieci i ruch sieciowy musiałby zostać przekierowany, czas RTT uległby zmianie (możliwe, że w poważnym stopniu). Inaczej mówiąc, TCP musi **dynamicznie** określać czas RTO. Rozważymy ten problem jako następny.

14.3. Ustalanie czasu oczekiwania na retransmisję (RTO)

Podstawowe znaczenie dla procedur obsługujących przeterminowanie i retransmisje ma sposób ustalania czasu RTO w oparciu o pomiar czasów RTT występujących w danym połączeniu. Jeśli TCP wykona retransmisję segmentu przed upływem czasu RTT, może niepotrzebnie wprowadzić do sieci zdublowane dane. Na odwrót, jeśli TCP opóźni wy-

syłanie o czas dużo dłuższy niż pojedynczy czas RTT, całkowite wykorzystanie sieci (i przepustowość pojedynczego połączenia) spadnie, gdy nastąpi utrata danych. Określanie czasu RTT jest skomplikowane, bo może się on zmieniać w następstwie zmian tras i stopnia wykorzystania sieci. TCP musi śledzić te zmiany i modyfikować swój limit czasu w stosowny sposób, aby utrzymywać dobrą wydajność.

Ponieważ TCP wysyła potwierdzenia, kiedy odbiera dane, możliwe jest wysłanie pojedynczego bajta z określonym numerem sekwencyjnym i zmierzenie czasu, który upłynie do momentu otrzymania potwierdzenia obejmującego ten numer sekwencyjny. Każdy taki pomiar jest nazywany **próbką RTT**. Wyzwaniem dla TCP jest wykonanie dobrego oszacowania zakresu wartości RTT na podstawie zbioru próbek, które zmieniają się w czasie. Drugi krok stanowi określenie czasu RTO na podstawie tych wartości. Właściwe wykonanie powyższego zadania jest bardzo ważne dla wydajności protokołu TCP.

Czas RTT jest szacowany dla każdego połączenia TCP oddzielnie, a jeden licznik czasu oczekiwania na retransmisję jest ustawiony, kiedy tylko jakieś dane konsumujące numer sekwencyjny (w tym segmenty `SYN` i `FIN`) są w trakcie przesyłania przez sieć. Właściwy sposób ustawiania tego licznika czasu jest od lat przedmiotem badań, a od czasu do czasu wprowadzane są ulepszenia. W tym podrozdziale poznamy niektóre spośród ważniejszych kamieni milowych w ewolucji metody stosowanej do obliczenia czasu RTO. Rozpoczniemy od pierwszej („klasycznej") metody, której szczegóły można znaleźć w dokumencie [RFC0793].

14.3.1. Metoda klasyczna

Oryginalna specyfikacja protokołu TCP ([RFC0793]) wymagała od TCP aktualizacji tzw. „wygładzonego" estymatora RTT (**SRTT**; *Smoothed RTT estimator*) przy użyciu następującego wzoru:

$$SRTT \leftarrow \alpha(SRTT) + (1-\alpha)RTT_S$$

Zgodnie z powyższym wzorem, estymator *SRTT* jest aktualizowany zarówno na podstawie swej dotychczasowej wartości, jak i wartości nowej próbki (*sample*), czyli RTT_S. Stała α jest współczynnikiem wygładzającym, określającym wagę składników wzoru, z zalecaną wartością między 0,8 a 0,9. Wartość *SRTT* jest aktualizowana za każdym razem, gdy wykonywany jest nowy pomiar. Przy zachowaniu oryginalnej zalecanej wartości dla współczynnika α jest oczywiste, że każde nowe oszacowanie zależy w zakresie od 80% do 90% od poprzedniego oszacowania, a nowy pomiar ma wagę od 10% do 20%. Ten typ średniej jest również znany jako **wykładniczo ważona średnia ruchoma** (**EWMA**; *Exponentially Weighted Moving Average*) lub **filtr dolnoprzepustowy** (*low-pass filter*). Jest on wygodny ze względów implementacyjnych, ponieważ wymaga zapamiętania tylko jednej poprzedniej wartości *SRTT* do prowadzenia bieżącego oszacowania.

Przy danym estymatorze *SRTT*, który zmienia się wraz ze zmianami czasów RTT, dokument [RFC0793] zalecił, aby wyznaczać czas RTO zgodnie z następującym wzorem:

$$RTO = \min(ubound, \max(lbound, (SRTT)\beta))$$

gdzie β oznacza współczynnik wariancji opóźnienia o zalecanej wartości od 1,3 do 2,0, *ubound* (*upper bound*) wyznacza górny limit (sugerowana wartość, np. 1 minuta), a *lbound* (*lower bound*) stanowi dolne ograniczenie (sugerowana wartość, np. 1 s) dla wartości RTO. Nazwiemy tę procedurę przypisania wartości RTO **metodą klasyczną**. Na ogół skutkuje ona ustawieniem czasu RTO albo na 1 s, albo na podwojoną wartość *SRTT*. Była odpowiednia dla stosunkowo stabilnych rozkładów czasów RTT. Kiedy jednak protokół TCP był wykorzystywany w sieciach charakteryzujących się dużą zmiennością czasów RTT (np. w przypadku pakietowych sieci radiowych), metoda klasyczna nie spisywała się zbyt dobrze.

14.3.2. Metoda standardowa

W referacie [J88] Jacobson wyszczególnił problemy związane z metodą klasyczną, idąc jeszcze dalej — zasadniczo chodziło mu o to, że licznik czasu zdefiniowany przez dokument [RFC0793] nie może nadążyć za dużymi fluktuacjami czasu RTT (i szczególnie wywołuje niepotrzebne retransmisje w sytuacji, gdy rzeczywisty czas RTT jest dużo większy od przewidywanego). Niepotrzebne retransmisje przyczyniają się do dodatkowego zwiększenia obciążenia sieci, gdy sieć i tak jest mocno obciążona, na co wskazują rosnące czasy RTT próbek.

Aby zaradzić temu problemowi, metoda używana do wyznaczania czasu RTO została poprawiona, uwzględniono w niej większą zmienność czasów RTT. Zostało to osiągnięte przez prowadzenie szacowania zmienności występującej w pomiarach RTT oprócz szacowania samej średniej wynikającej z pomiarów. Ustalanie czasu RTO zarówno na podstawie średniej, jak i estymatora zmienności pozwala na uzyskanie lepszej reakcji stosowanych limitów czasu na duże fluktuacje czasów podróży w obie strony niż podczas prostego obliczania RTO jako ustalonej wielokrotności średniej czasów RTT.

Rysunki 5. i 6. w pracy [J88] przedstawiają porównanie wartości RTO obliczonych metodą opisaną w dokumencie [RFC0793] dla pewnych rzeczywistych czasów RTT, z obliczeniami RTO, które wkrótce zaprezentujemy, a w których wzięto pod uwagę zmienność czasów podróży w obie strony. Jeżeli potraktujemy pomiary czasu RTT wykonywane przez TCP jako próbki procesu statystycznego, oszacowanie zarówno średniej, jak i wariancji (lub odchylenia standardowego) pomaga lepiej przewidywać możliwe przyszłe wartości, które proces może przyjąć. Dobra prognoza dotycząca zakresu możliwych wartości czasu RTT pomaga protokołowi TCP określić wartość RTO, która nie będzie dla większości przypadków ani za duża, ani za mała.

Jak to przedstawia Jacobson, **średnie odchylenie** stanowi dobrą aproksymację odchylenia standardowego, ale jest łatwiejsze do obliczenia i można to zrobić szybciej. Obliczenie odchylenia standardowego wymaga przeprowadzenia matematycznej operacji wyciągnięcia pierwiastka kwadratowego z wariancji, co jest zbyt kosztowne czasowo dla szybkiej implementacji TCP. (To nie wyczerpuje całości sprawy. Aby poznać fascynującą historię tzw. „debaty", sięgnij do źródła [G04].) A zatem musimy wykonywać oszacowania nie tylko średniej, ale także średniego odchylenia. Prowadzi to do następujących równań, które należy stosować do każdej wartości pomiaru czasu RTT oznaczonej literą M (*measurement*; wcześniej wartość tę oznaczaliśmy RTT_S):

$$srtt \leftarrow (1-g)(srtt) + (g)M$$
$$rttvar \leftarrow (1-h)(rttvar) + (h)(|M - srtt|)$$
$$\text{RTO} = srtt + 4(rttvar)$$

W powyższych wzorach wartość *srtt* w istocie zastępuje wcześniejszą wartość *SRTT*, a wartość *rttvar*, która staje się wartością EWMA średniego odchylenia, jest używana zamiast mnożnika β do ustalenia czasu RTO. Ten zestaw równań można zapisać w postaci, która wymaga mniejszej liczby operacji, gdy jest implementowana na konwencjonalnym komputerze:

$$Err = M - srtt$$
$$srtt \leftarrow srtt + g(Err)$$
$$rttvar \leftarrow rttvar + h(|Err| - rttvar)$$
$$\text{RTO} = srtt + 4(rttvar)$$

Zgodnie z zaleceniem, *srtt* jest średnią EWMA czasów RTT, a *rttvar* jest średnią EWMA błędu bezwzględnego, czyli $|Err|$. Wartość zmiennej Err stanowi różnicę między zmierzoną wartością M i aktualną wartością estymatora RTT wynoszącą *srtt*. Obie wartości, *srtt* i *rttvar*, są używane do obliczenia wartości RTO, która zmienia się w czasie. Współczynnik przyrostu g jest wagą nadaną nowej próbce czasu RTT o wartości M w średniej *srtt* i ma przypisaną wartość $^1/_8$. Współczynnik przyrostu h, który jest wagą przypisaną nowej próbce średniego odchylenia (dokładnie, bezwzględnej wartości różnicy między wartością M dla nowej próbki a bieżącą wartością średniej *srtt*) w oszacowaniu odchylenia reprezentowanym przez wartość *rttvar*, ma nadaną wartość ¼. Przyjęcie większej wartości współczynnika przyrostu dla odchylenia skutkuje szybszym wzrostem czasu RTO, kiedy zmieni się wartość czasu RTT. Wartości g i h zostały wybrane jako (ujemne) potęgi liczby 2, co pozwala na zaimplementowanie całego procesu obliczeń na komputerze używającym stałoprzecinkowej arytmetyki liczb całkowitych, stosującym operacje przesunięcia i dodawania zamiast operacji mnożenia i dzielenia.

Źródło [J88] zawierało wyrażenie $2 \cdot rttvar$ w obliczeniu wartości RTO, ale po dalszych badaniach wartość ta została zmieniona na $4 \cdot rttvar$ w pracy [J90] i w tej postaci została użyta w implementacji BSD Net/1, a ostatecznie w standardzie [RFC6298].

Porównując metodę klasyczną z metodą zaproponowaną przez Jacobsona, widzimy, że obliczanie średniej wartości RTT jest w obu przypadkach podobne (α równa się 1 minus współczynnik przyrostu g), ale używana jest inna wartość współczynnika przyrostu. Poza tym, obliczenie czasu RTO zastosowane przez Jacobsona zależy i od wygładzonego czasu RTT, i od wygładzonego odchylenia, podczas gdy w metodzie klasycznej używa się prostego mnożnika zastosowanego do wygładzonego czasu RTT. Metoda Jacobsona stanowi podstawę sposobu obliczania czasu RTO używanego do dzisiaj przez wiele implementacji protokołu TCP, a z powodu przyjęcia jej jako podstawy dokumentu [RFC6298] będziemy nazywać ją **metodą standardową**, chociaż dokument [RFC6298] zawiera jej niewielkie udoskonalenia, które teraz omówimy.

14.3.2.1. Ziarnistość zegara i ograniczenia czasu RTO

Protokół TCP utrzymuje stale chodzący „zegar", który jest wykorzystywany przy wykonywaniu pomiarów czasów RTT. Podobnie jak w przypadku numerów sekwencyjnych, rzeczywiste połączenia TCP nie uruchamiają swoich zegarów w momencie zero,

a zegar nie posiada nieskończonej precyzji. Właściwie zegar protokołu TCP jest reprezentowany przez wartość zmiennej, która jest aktualizowana wraz ze wzrostem wskazania zegara systemowego, ale niekoniecznie w stosunku jeden-do-jeden. Długość czasu pojedynczego „tyknięcia" zegara TCP nazywa się jego **ziarnistością** (*granularity*). Tradycyjnie wartość ta była stosunkowo duża (ok. 500 ms), ale nowsze implementacje korzystają z zegarów o drobniejszej ziarnistości (np. 1 ms w przypadku Linuksa).

Ziarnistość może mieć wpływ na szczegóły wykonywania pomiarów RTT, a także na sposób ustalenia czasu RTO. W dokumencie [RFC6298] ziarnistość jest wykorzystywana do udoskonalenia procedury aktualizacji czasu RTO. Dodatkowo na czas RTO zostało nałożone dolne ograniczenie. Zastosowane równanie wygląda następująco:

$$\mathrm{RTO} = \max\left(srtt + \max\left(G, 4\left(rttvar\right)\right), 1000\right)$$

gdzie G oznacza ziarnistość licznika czasu, a 1000 ms stanowi dolną granicę całkowitego czasu RTO (zgodnie z zaleceniem zasady (2.4) zawartej w dokumencie [RFC6298]). W konsekwencji czas RTO zawsze wynosi co najmniej 1 s. Opcjonalna górna granica jest również dozwolona, pod warunkiem że jej wartość wynosi co najmniej 60 s.

14.3.2.2. Wartości początkowe

Dowiedzieliśmy się, jak estymatory są aktualizowane wraz z upływem czasu, ale musimy również wiedzieć, jak ustawić ich wartości początkowe. Przed wymianą pierwszych segmentów SYN TCP nie potrafi dobrze określić, jakiej wartości ma użyć do ustawienia początkowego czasu RTO. Protokół także „nie wie", jakie wartości początkowe wybrać dla swoich estymatorów, o ile system nie dostarczy odpowiednich wskazówek (niektóre systemy buforują te informacje w tablicy przekazywania — *forwarding table*; patrz podrozdział 14.9). Zgodnie z dokumentem [RFC6298] początkowe ustawienie czasu RTO powinno wynosić 1 s, chociaż wartość 3 s jest używana w przypadku limitu czasu oczekiwania na potwierdzenie początkowego segmentu SYN. Kiedy wykonany zostanie pierwszy pomiar czasu RTT, którego wartość wynosi M, estymatory zostaną zainicjowane w sposób następujący:

$$srtt \leftarrow M$$
$$rttvar \leftarrow M/2$$

Znamy teraz wystarczającą ilość szczegółów, by rozumieć, jak estymatory są inicjowane i utrzymywane. Odnośne procedury zależą od uzyskiwania próbek czasu RTT, co mogłoby się wydawać prostą sprawą. Zobaczymy teraz, dlaczego nie zawsze tak jest.

14.3.2.3. Niejednoznaczność wynikająca z retransmisji i algorytm Karna

Może wystąpić problem z pomiarem próbki RTT, w sytuacji gdy pakiet jest retransmitowany. Powiedzmy, że pakiet jest transmitowany, dochodzi do przeterminowania, w następstwie czego pakiet jest retransmitowany i zostaje odebrane potwierdzenie dla tego pakietu. Czy potwierdzenie ACK dotyczy pierwszej, czy drugiej transmisji? Jest to przykład **problemu niejednoznaczności wynikającej z retransmisji** (*retransmission ambiguity problem*). Problem ten występuje, jeśli nie jest używana opcja *Znaczniki czasu*, bo wtedy odebrany segment ACK zawiera jedynie numer potwierdzenia bez żadnej wskazówki, który egzemplarz (tj. pierwszy czy drugi) numeru sekwencyjnego jest potwierdzany numerem ACK.

Referat [KP87] ustala, że kiedy wystąpi przeterminowanie i retransmisja, nie możemy zaktualizować estymatorów RTT po ostatecznym przyjściu potwierdzenia dla retransmitowanych danych. Zasada ta stanowi „pierwszą część" algorytmu Karna. Eliminuje ona problem niejednoznaczności potwierdzenia przez usunięcie tej niejednoznaczności z procedury obliczania szacunkowego czasu RTT. Zasada ta stanowi wymaganie zawarte w dokumencie [RFC6298].

Jeśli jednak po prostu ignorowalibyśmy całkowicie retransmitowane segmenty, ustalając wartość RTO, tracilibyśmy możliwość uwzględnienia pewnych pożytecznych informacji dostarczanych przez sieć (np. tego, że sieć okazuje pewnego rodzaju niezdolność do szybkiego dostarczania pakietów). W takich przypadkach korzystne byłoby zredukowanie obciążenia sieci przez zmniejszenie częstości retransmisji, przynajmniej dopóki pakiety przestaną być gubione. To rozumowanie stanowi podstawę stosowania odczekiwania wykładniczego, które zostało przedstawione na rysunku 14.1.

TCP stosuje **współczynnik odczekiwania** (*backoff factor*) do wartości RTO, który podwaja się za każdym kolejnym wyzerowaniem licznika czasu oczekiwania na retransmisję. Podwajanie jest kontynuowane, dopóki nie zostanie odebrane potwierdzenie dla segmentu, który nie był retransmitowany. W tym momencie współczynnik odczekiwania otrzymuje z powrotem wartość 1 (tzn. następuje anulowanie binarnego odczekiwania wykładniczego), a licznik czasu oczekiwania na retransmisję wraca do swej normalnej wartości. Podwajanie współczynnika odczekiwania przy kolejnych retransmisjach stanowi „drugą część" algorytmu Karna. Zauważmy, że kiedy protokół TCP obsługuje zdarzenie przeterminowania, uruchamia również procedury kontroli przeciążenia, które zmieniają jego szybkość transmisji. (Kontrola przeciążenia jest szczegółowo omawiana w rozdziale 16.). Algorytm Karna składa się więc rzeczywiście z dwóch części. Świadczy o tym niżej zacytowany fragment referatu z 1987 roku (patrz [KP87]:

> *Kiedy przychodzi potwierdzenie dla pakietu, który został wysłany więcej niż jeden raz (tzn. był przynajmniej raz retransmitowany), należy zignorować jakikolwiek pomiar oparty na tym pakiecie, unikając w ten sposób problemu niejednoznaczności wynikającej z retransmisji. Dodatkowo, uwzględniająca współczynnik odczekiwania wartość RTO dla tego pakietu jest zachowywana dla następnego pakietu. Dopiero wtedy, gdy zostanie on (lub kolejny pakiet) potwierdzony bez retransmisji w międzyczasie, wartość RTO zostanie przeliczona ponownie na podstawie SRTT.*

Algorytm ten już od pewnego czasu (od publikacji dokumentu [RFC1122]) jest procedurą wymaganą do zastosowania w implementacji protokołu TCP. Istnieje jednak wyjątek, gdy używana jest opcja TCP *Znaczniki czasu* (patrz rozdział 13.). W tym przypadku można uniknąć problemu niejednoznaczności potwierdzenia, a więc pierwsza część algorytmu Karna nie ma tu zastosowania.

14.3.2.4. Pomiar RTT (RTTM) z opcją znaczników czasu

Opcja *Znaczniki czasu protokołu* TCP (TSOPT), oprócz dostarczenia podstawy do algorytmu PAWS, który poznaliśmy w rozdziale 13., może być także wykorzystana do pomiaru czasu RTT (**RTTM**; *Round-Trip Time Measurement*; patrz [RFC1323]). Podstawowy format opcji TSOPT został opisany w rozdziale 13. Pozwala ona nadawcy na umieszczenie w segmencie TCP 32-bitowej liczby, która jest zwracana w odpowiednim potwierdzeniu.

Wartość znacznika czasu (TSV) jest przekazywana w opcji TSOPT początkowego segmentu SYN i jest zwracana w części TSER opcji TSOPT w segmencie SYN+ACK, co dostarcza sposobu ustalenia wartości początkowych estymatorów srtt, rttvar i czasu RTO. Ponieważ początkowy segment SYN „liczy się" jako element danych (tzn. jest retransmitowany, jeśli zostanie zgubiony, oraz konsumuje numer sekwencyjny), jest mierzony jego czas RTT. Opcje TSOPT są przekazywane także w innych segmentach, więc czas RTT dla połączenia może być szacowany na bieżąco. Wydaje się to dość proste, ale okazuje się bardziej złożone, ponieważ TCP nie zawsze dostarcza potwierdzenia ACK dla każdego odebranego segmentu. Przykładowo TCP dostarcza potwierdzenia ACK dla co drugiego segmentu (patrz rozdział 15.), kiedy przesyłane są wielkie ilości danych. W dodatku, kiedy dane są tracone, jest zmieniana ich kolejność lub są skutecznie retransmitowane, kumulatywny mechanizm potwierdzeń ACK stosowany przez TCP oznacza, że niekoniecznie istnieje ustalona odpowiedniość między segmentem a jego potwierdzeniem ACK. Aby sprostać tym wyzwaniom, implementacje TCP, które używają opcji TSOPT (obecnie większość — łącznie z systemami Linux i Windows), stosują następujący algorytm pobierania próbek RTT:

- Wysyłający protokół TCP umieszcza 32-bitową wartość znacznika czasu w części TSV opcji TSOPT w każdym wysyłanym segmencie TCP. Pole to zawiera wartość „zegara" TCP nadawcy w momencie wysłania segmentu.
- Odbierający protokół TCP przechowuje odebraną wartość TSV, aby przesłać ją w następnym potwierdzeniu ACK, jakie wygeneruje (w zmiennej, która zwykle nazywa się TsRecent), oraz numer ACK przekazany w ostatnim wysłanym przez siebie potwierdzeniu ACK (w zmiennej o nazwie LastACK). Przypomnijmy sobie, że numery ACK reprezentują następne w kolejności numery sekwencyjne, których odbiorca (czyli nadawca potwierdzenia ACK) spodziewa się od nadawcy.
- Kiedy przychodzi nowy segment, to, pod warunkiem że jego numer sekwencyjny zgadza się z wartością pamiętaną w zmiennej LastACK (tzn. jest to oczekiwany następny segment), zawartość pola TSV segmentu zostaje zapamiętana w zmiennej TsRecent.
- Zawsze gdy odbiorca wysyła potwierdzenie ACK, przekazuje w nim opcję TSOPT, umieszczając w polu TSER tej opcji wartość znacznika czasu zawartą w zmiennej TsRecent.
- Nadawca po otrzymaniu potwierdzenia ACK, które skutkuje przesunięciem jego okna do przodu (czyli w prawo), odejmuje wartość TSER od aktualnej zawartości swojego zegara TCP i używa otrzymanej różnicy jako wartości próbki do aktualizacji swoich estymatorów czasu RTT.

Znaczniki czasu są domyślnie włączone w systemach FreeBSD i Linux oraz, w reakcji na systemy używające tej opcji, w późniejszych wersjach systemu Windows. W Linuksie ustawienie zmiennej konfiguracyjnej systemu net.ipv4.tcp_timestamps rozstrzyga, czy znaczniki czasu są używane (wartość 1), czy nie (wartość 0). W systemie Windows ich użycie jest sterowane przez parametr Tcp1323Opts we wspomnianym wcześniej obszarze rejestru. Jeśli ma on wartość 0, znaczniki czasu są wyłączone. Jeśli jego wartość wynosi 2, znaczniki czasu są włączone. Klucz ten nie ma wartości domyślnej (domyślnie nie ma go w rejestrze). Domyślne zachowanie protokołu polega na wykorzystaniu znaczników czasu, jeśli druga strona użyje ich przy inicjowaniu połączenia.

14.3.3. Metoda systemu Linux

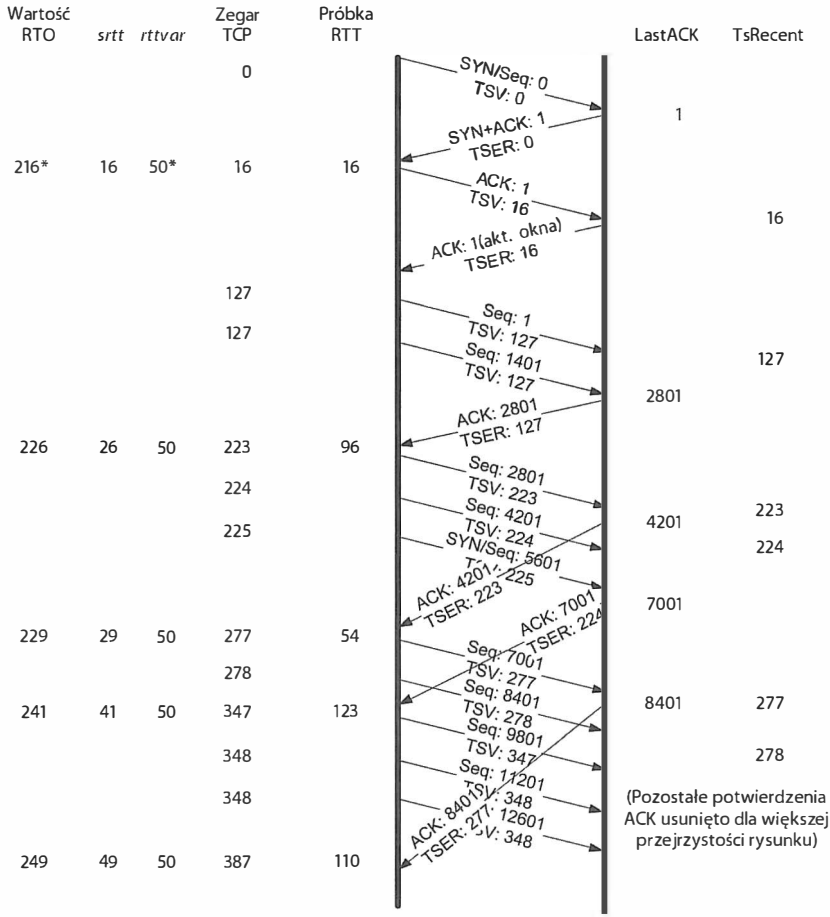
Stosowana w systemie Linux procedura szacowania czasu RTT działa nieco inaczej niż metoda standardowa. Korzysta ona z ziarnistości zegara równej 1 ms, a więc drobniejszej od stosowanych przez wiele innych implementacji, oraz z opcji TSOPT. Połączenie częstych pomiarów czasu RTT i drobnej ziarnistości zegara przyczynia się do dokładniejszego oszacowania czasu RTT, ale również tworzy tendencję do minimalizowania wartości rttvar w przebiegu czasu [LS00]. Dzieje się tak, ponieważ występuje tendencja do wzajemnego znoszenia się próbek średniego odchylenia, kiedy nagromadzi się zbyt duża ich liczba. Jest to jeden z powodów takiego sposobu ustawiania czasu RTO, który różni się nieco od metody standardowej. Drugi powód wiąże się ze sposobem, w jaki metoda standardowa zwiększa wartość rttvar, kiedy próbka RTT ma znacznie **mniejszą** wartość niż aktualne oszacowanie czasu RTT reprezentowane przez srtt.

Aby lepiej zrozumieć tę drugą kwestię, przypomnijmy sobie, że czas RTO zwykle otrzymuje wartość srtt+4(rttvar). W konsekwencji dowolna duża zmiana wartości rttvar powoduje zwiększenie czasu RTO, niezależnie od tego, czy ostatnia próbka RTT jest większa, czy też mniejsza od srtt. Jest to sprzeczne z intuicją — jeśli rzeczywisty czas RTT zmniejszył się znacznie, nie chcemy, aby konsekwencją tego było zwiększenie czasu RTO. Linux radzi sobie z tym problemem, ograniczając wpływ znacznych spadków wartości próbek RTT na wartość rttvar. Teraz przyjrzymy się szczegółom procedury używanej przez system Linux do ustalania wartości RTO; procedura ta stara się rozwiązać oba wyżej przedstawione problemy.

Linux utrzymuje zmienne srtt i rttvar, podobnie jak metoda standardowa, ale używa także nowych zmiennych o nazwach mdev i mdev_max. Wartość zmiennej mdev jest obliczana jako bieżące oszacowanie średniego odchylenia przy użyciu opisanego wcześniej standardowego algorytmu służącego do obliczenia wartości rttvar. Zmienna mdev_max przechowuje maksymalną wartość mdev, która wystąpiła w ciągu ostatnio zmierzonego czasu RTT, i nigdy nie może zawierać wartości mniejszej niż 50 ms. Ponadto zmienna rttvar jest regularnie aktualizowana, by mieć pewność, że nie zawiera wartości mniejszej niż mdev_max. W konsekwencji wartość czasu RTO nigdy nie spada poniżej 200 ms.

Minimalna wartość RTO może zostać zmieniona. Stała konfiguracyjna jądra systemu, TCP_RTO_MIN, może zostać zmieniona przed rekompilacją i instalacją jądra. Niektóre wersje Linuksa umożliwiają jej zmianę przy użyciu polecenia ip route. Kiedy protokół TCP jest używany w sieciach centrów danych, gdzie czasy RTT mogą wynosić kilka mikrosekund, minimalne RTO równe 200 ms może doprowadzić do poważnej degradacji wydajności ze względu na wolną obsługę utraty pakietów w lokalnych przełącznikach przez protokół TCP. Jest to tzw. (w angielskiej nomenklaturze sieciowej) *incast problem.* Istnieją różne rozwiązania tego problemu, w tym modyfikacja ziarnistości licznika czasu protokołu TCP oraz zmiana wartości minimalnego RTO do rzędu mikrosekund [VO09]. Używanie tak małych wartości minimalnych czasu RTO w globalnym Internecie nie jest zalecane.

Linux uaktualnia zmienną rttvar, przypisując jej wartość zmiennej mdev_max, ilekroć zwiększy się maksimum (przechowywane w tej ostatniej zmiennej). System zawsze ustawia wartość RTO jako sumę srtt i 4(rttvar), równocześnie zapewniając, że nigdy nie przekroczy ona wartości TCP_RTO_MAX, która domyślnie wynosi 120 s. Więcej szczegółów można znaleźć w pracy [SK02]. Szczegóły działania całości tej procedury możemy zobaczyć na rysunku 14.1, na którym pokazujemy również, jak funkcjonuje opcja znaczników czasu.



Rysunek 14.2. *Opcja znaczników czasu protokołu TCP przesyła kopię bieżącej wartości zegara TCP nadawcy. Potwierdzenia ACK zwracają tę wartość do nadawcy, który używa różnicy (aktualne wskazanie zegara minus zwrócony znacznik czasu) do aktualizacji swych oszacowań srtt i rttvar. Ze względu na przejrzystość diagramu został przedstawiony tylko jeden zestaw znaczników czasu. W tym systemie Linux wartość rttvar została ograniczona do przynajmniej 50 jednostek (milisekund), a dolna granica czasu RTO wynosi 200 ms*

Na rysunku 14.2 widzimy połączenie TCP używające opcji znaczników czasu od momentu rozpoczęcia. Nadawcą jest Linux 2.6, a odbiorcą jest system FreeBSD 5.4. Numery sekwencyjne i wartości znaczników czasu zostały pokazane jako wartości względne dla zwiększenia przejrzystości. W dodatku zostały przedstawione tylko znaczniki czasu nadawcy. Rysunek nie odzwierciedla dokładnie skali czasu, aby wartości liczbowe były łatwiejsze do odczytania. W oparciu o początkowy pomiar czasu RTT wykonany w naszym przykładzie Linux wykonuje następujące aktualizacje:

- srtt = 16 ms
- mdev = (16/2) ms = 8 ms
- rttvar = mdev_max = max(mdev, TCP_RTO_MIN) = max(8, 50) = 50 ms
- RTO = srtt+4(rttvar) = 16 + 4(50) = 216 ms

Po początkowej wymianie segmentów SYN nadawca dostarcza potwierdzenie ACK dla segmentu SYN otrzymanego od odbiorcy, a odbiorca odpowiada przesłaniem aktualizacji okna. Ponieważ żaden z tych pakietów nie zawiera danych (ani nie ma ustawionych bitów SYN lub FIN, które są traktowane jako dane), czasy ich przesyłania nie są mierzone. Wobec tego, kiedy aktualizacja okna dotrze do nadawcy, nie jest wykonywane żadne uaktualnienie estymatora RTT. Segmenty, które nie zawierają danych, nie są dostarczane w sposób niezawodny przez TCP, co oznacza, że nie są retransmitowane w przypadku utraty. Te rodzaje segmentów nie wymagają ustawienia licznika czasu oczekiwania na retransmisję, ponieważ nigdy nie są retransmitowane.

Warto zauważyć, że same opcje TCP także nie są retransmitowane ani dostarczane w niezawodny sposób. Tylko wtedy, gdy opcje zostaną specjalnie umieszczone w segmentach danych (w tym w segmentach SYN i FIN), będą retransmitowane w przypadku utraty i nawet wtedy będzie to tylko efekt uboczny.

Kiedy aplikacja wykonuje swoją pierwszą operację zapisu, wysyłający protokół TCP transmituje dwa segmenty, każdy z wartością TSV równą 127. Wartości te są identyczne dla tych dwóch segmentów, ponieważ czas, który upłynął między pierwszą a drugą transmisją, był mniejszy od 1 ms (czyli ziarnistości zegara TCP nadawcy). Nie jest to niezwykłe, że zegar nie zwiększa swojego wskazania lub zwiększa go o niewielką wartość, gdy nadawca wysyła wiele segmentów, jeden po drugim, tak jak w naszym przykładzie.

Zmienna LastACK u odbiorcy przechowuje numer ACK ostatnio przesłany przez odbiorcę. W naszym przykładzie pierwsza wartość zmiennej LastACK wynosi 1, ponieważ ostatnie potwierdzenie ACK zostało przesłane w pakiecie SYN+ACK podczas ustanawiania połączenia. Kiedy przychodzi pierwszy „pełnowymiarowy" segment, jego numer sekwencyjny jest zgodny z wartością zmiennej LastACK, więc zostanie zaktualizowana zmienna TsRecent, która otrzyma wartość 127 pobraną z pola TSV przychodzącego segmentu. Przyjście drugiego segmentu nie powoduje aktualizacji zmiennej TsRecent, ponieważ zawartość pola *Numer sekwencyjny* nie jest zgodna z wartością znajdującą się w zmiennej LastACK. Potwierdzenie ACK wysłane w odpowiedzi na odebrane segmenty zawiera wartość zmiennej TsRecent w polu TSER, a jego transmisja powoduje także aktualizację zmiennej LastACK przez odbiorcę, który umieszcza w niej numer ACK równy 2801.

Kiedy ostatnie potwierdzenie ACK dociera do nadawcy, TCP może wykonać swój drugi pomiar czasu RTT. Pobiera aktualną wartość zegara i odejmuje od niej wartość pola TSER z otrzymanego pakietu — tworzy w ten sposób wartość pomiaru m: m = 223–127 = 96. Uwzględniając ten pomiar, protokół TCP Linuksa aktualizuje zmienne połączenia w następujący sposób:

- mdev = mdev $(^3/_4)$+|m–srtt|$(^1/_4)$ = 8$(^3/_4)$+|80|$(^1/_4)$ = 26 ms
- mdev_max = max(mdev_max, mdev) = max(50, 26) = 50 ms
- srtt = srtt$(^7/_8)$+m$(^1/_8)$ = 16$(^7/_8)$+96$(^1/_8)$ = 14+12 = 26 ms

- rttvar = mdev_max = 50 ms
- RTO = srtt+4(rttvar) = 26+4(50) = 226 ms

Jak już przedtem wspominaliśmy, protokół TCP systemu Linux zawiera kilka specjalnych modyfikacji w stosunku do klasycznego algorytmu szacowania czasu RTT, które zasługują na omówienie. Kiedy były opracowywane klasyczne algorytmy, typowa ziarnistość zegara TCP wynosiła 500 ms, a użycie opcji znaczników czasu nie było szeroko rozpowszechnione. Typowe było pobieranie jednej próbki czasu RTT na okno danych i odpowiednie aktualizowanie estymatorów. Jest to wciąż stosowane, w przypadku gdy znaczniki czasu nie są dostępne lub nie są włączone.

Jeżeli pobierana jest tylko jedna próbka na okno danych, składnik rttvar zmienia się stosunkowo wolno. W przypadku zastosowania znaczników czasu i pomiarów czasu dla poszczególnych pakietów na podstawie zawartych w nich znaczników czasu możliwe jest wykonywanie znacznie większej ilości pomiarów. Ponieważ zazwyczaj czasy RTT mało się różnią w kolejnych pakietach tego samego okna danych, wykonywanie tak wielu pomiarów w krótkim okresie czasu (szczególnie wtedy, kiedy okno jest duże) może prowadzić do małych wartości oszacowania średniego odchylenia (bliskich zeru, na mocy prawa wielkich liczb; patrz [F68]). Aby zaradzić temu problemowi, Linux utrzymuje zmienną mdev jako bieżącą wartość oszacowania średniego odchylenia, ale ustala wartość RTO na podstawie zmiennej rttvar zawierającej maksymalną wartość mdev, jaka wystąpiła w czasie transmisji pojedynczego okna danych, nie mniejszą jednak od minimum wynoszącego 50 ms. Wartość rttvar może zmniejszyć się tylko raz, przy przejściu od jednego okna do następnego.

Podejście standardowe przypisuje składnikowi rttvar dużą wagę (współczynnik 4) i w rezultacie wartość RTO ma tendencję do wzrostu nawet wtedy, gdy czas RTT spada. W przypadku zegara o grubej ziarnistości (np. 500 ms) efekt ten może być stosunkowo niewielki, ponieważ jest tak mało wartości, które może przyjąć czas RTO. Jednakże przy zegarze o drobniejszej ziarnistości, przyjmującej jak w systemie Linux wartość 1 ms, może to budzić niepokój. Aby zaradzić temu problemowi, Linux specjalnie obsługuje przypadek zmniejszającego się czasu RTT, nadając mniejszą wagę nowej próbce, jeśli ma ona wartość mniejszą niż „dolny koniec" oszacowania zakresu czasu RTT (srtt–mdev). Całość relacji wygląda następująco:

$$\text{if}\left(m < (srtt - mdev)\right)$$
$$mdev = (31/32)\cdot mdev + (1/32)\cdot|srtt - m|$$
$$\text{else}$$
$$mdev = (3/4)\cdot mdev + (1/4)\cdot|srtt - m|$$

Wyrażenie warunkowe sprawdza, czy wartość nowej próbki czasu RTT jest niższa od dolnej granicy zakresu oczekiwanych wyników pomiarów. Jeśli tak się dzieje, nowa próbka wskazuje, że połączenie może doświadczać znaczącego zmniejszania się czasów RTT. Aby w takich przypadkach uniknąć zwiększenia wartości mdev (a w konsekwencji, wartości rttvar i RTO), nowa próbka średniego odchylenia |srtt–m| otrzymuje wagę zmniejszoną 8-krotnie w stosunku do swej normalnej wartości. W sumie skutkuje to uniknięciem problemu zwiększającego się czasu RTO w przypadkach, gdy czas RTT się zmniejsza. Dogłębną analizę tych kwestii można znaleźć w źródłach [LS00] i [SK02].

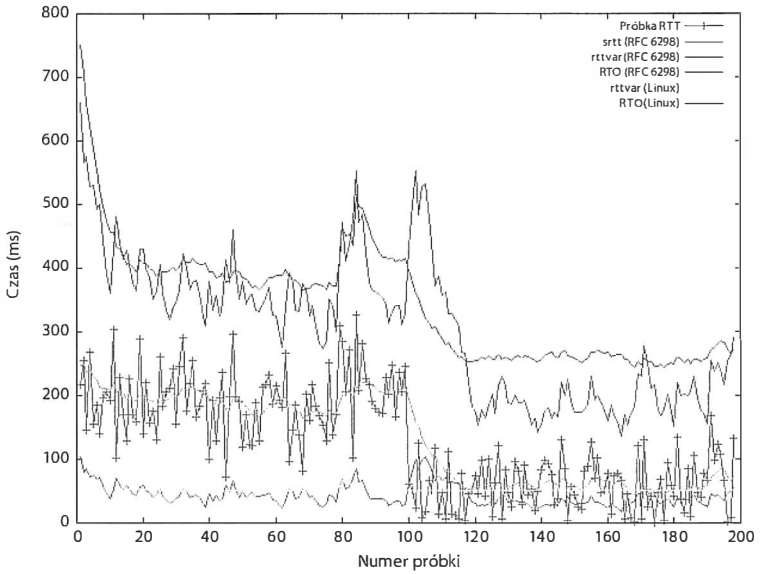
W opracowaniu [RKS07] autorzy dokonali oceny algorytmów szacowania czasu RTT używanych przez protokół TCP w różnych systemach operacyjnych na podstawie analizy 2,8 miliona połączeń TCP. Doszli do wniosku, że estymator systemu Linux jest najbardziej efektywny wśród zbadanych rozwiązań, głównie z powodu swej stosunkowo szybkiej konwergencji, ale również dlatego, że może być najbardziej skutecznie dostrojony przez zredukowanie wpływu wariancji RTT na ustalanie czasu RTO. Wróćmy teraz do rysunku 14.2. Kiedy potwierdzenie z numerem `ACK 7001` jest tworzone u odbiorcy, widzimy, że jego pole `TSER` zawiera kopię wartości `TSV` pochodzącej nie z segmentu, który przyszedł ostatni, ale z najstarszego segmentu, który nie został jeszcze potwierdzony. Po zwróceniu do nadawcy to `ACK` powoduje pomiar próbki RTT od momentu wysłania pierwszego z dwóch segmentów, a nie od momentu wysłania ostatniego segmentu. Właśnie tak działa algorytm z użyciem znaczników czasu przy opóźnionych lub w inny sposób nieregularnych potwierdzeniach `ACK`. Kiedy próbka RTT jest mierzona od momentu wysłania najstarszego pakietu, odzwierciedla czas, przez jaki nadawca powinien oczekiwać na spodziewane potwierdzenie `ACK`, a nie rzeczywisty czas RTT sieci. Jest to ważne, ponieważ nadawca musi opierać swój czas RTO na spodziewanej szybkości otrzymywania potwierdzeń `ACK` od odbiorcy, która może być mniejsza niż szybkość przesyłania pakietów.

14.3.4. Działanie estymatorów RTT

Jak widzieliśmy, pokaźna ilość innowacji i inżynierii została zainwestowana w rozwiązanie kwestii ustalania czasu RTO i szacowania czasu RTT w protokole TCP. Na rysunku 14.3 pokazujemy, jak działają bardziej popularne estymatory w oparciu o zastosowanie algorytmu standardowego i algorytmu Linuksa do zbioru sztucznych danych. 1-sekundowe minimum dla czasu RTO zalecane przez dokument [RFC6298] dla metody standardowej zostało usunięte dla lepszego jej zilustrowania. Większość implementacji TCP funkcjonujących w rzeczywistym świecie i tak narusza tę dyrektywę (patrz [RKS07]).

Diagram pokazuje wykres szeregów czasowych 200 sztucznie wygenerowanych wartości pobranych z rozkładów prawdopodobieństwa Gaussa, $N(200, 50)$ i $N(50,50)$. Pierwszy rozkład został wykorzystany do pobrania pierwszych 100 próbek, a drugi do pobrania następnych 100 próbek. Wszystkie ujemne wartości próbek zostały zamienione na dodatnie przez zmianę znaku (dotyczy to tylko drugiego rozkładu). Każdy znak plus (+) oznacza pojedynczą wartość próbki. Widoczny jest znaczny spadek wartości próbek po próbce numer 100 i łatwo zauważyć, że metoda Linuksa obniża wartość RTO niemal natychmiast po tej próbce, podczas gdy podejście standardowe wymaga do tego jeszcze 20 próbek.

Jeśli teraz skupimy uwagę na wykresie wartości `rttvar`, widzimy, że pozostaje ona stosunkowo stała. Wynika to z równego 50 ms minimum nałożonego na wartość `mdev_max` (a w konsekwencji na wartość `rttvar`). W rezultacie wartość RTO w systemie Linux nigdy nie spada poniżej 200 ms i to pozwala uniknąć wszystkich niepotrzebnych retransmisji (chociaż, z drugiej strony, licznik czasu może zwlekać z sygnalizacją przeterminowania, co prowadzi do zmniejszonej wydajności w przypadku utraty pakietów). Podejście standardowe wpada w potencjalne problemy przy próbkach numer 78 i 191, gdzie może dojść do **zbędnej retransmisji** (*spurious retransmission*). Przeanalizujemy ten problem później.



Rysunek 14.3. *Linuksowe i standardowe algorytmy szacowania czasu RTT i wyznaczania czasu RTO zastosowane do sztucznych (pseudolosowych) próbek. Pierwsze 100 próbek pobrano z rozkładu N(200,50), a drugie 100 z rozkładu N(50,50) z zamianą ujemnych wartości na dodatnie. Linux unika wzrostu wartości RTO, kiedy średnia zmniejsza się po próbce numer 100. W systemie Linux efektywna wartość minimalnego czasu RTO wynosi 200 ms, więc po próbce numer 120 metoda standardowa uwzględnia mniejszy margines błędu. Linuksowi udaje się uniknąć ustalenia czasu RTO na zbyt niskim poziomie we wszystkich przypadkach objętych tym przykładem. Podejście standardowe napotyka na potencjalne problemy przy próbkach numer 78 i 191*

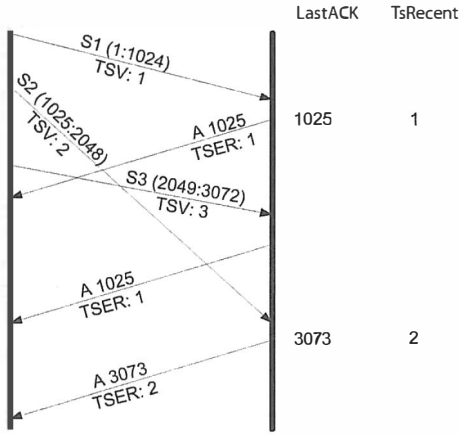
14.3.5. Odporność procedury RTTM na utratę i zmianę kolejności pakietów

Pokazano, że procedura z użyciem opcji TSOPT działa we właściwy sposób, kiedy nie są gubione pakiety, niezależnie od tego, czy odbiorca opóźnia wysyłanie niektórych potwierdzeń ACK, czy nie. Algorytm także funkcjonuje poprawnie w następujących przypadkach.

- **Segmenty poza kolejnością**: Kiedy odbiorca otrzymuje segment poza kolejnością, zazwyczaj z powodu utraty poprzedniego segmentu, przewiduje się natychmiastowe wygenerowanie potwierdzenia ACK, aby wspomóc działanie algorytmu szybkiej retransmisji (patrz podrozdział 14.5). To potwierdzenie ACK zawiera w swoim polu TSER wartość pola TSV ostatniego segmentu otrzymanego przez odbiorcę we właściwej kolejności (tzn. ostatniego segmentu przesuwającego do przodu okno danych, który **nie jest** na ogół tym segmentem, który przyszedł poza kolejnością). To zwykle powoduje wzrost wartości próbek czasu RTT u nadawcy, prowadząc do analogicznego wzrostu czasu RTO nadawcy. Kiedy pakiety docierają do odbiorcy w zmienionej kolejności, efekt ten jest korzystny, ponieważ daje nadawcy nieco więcej czasu, przed ewentualnym zainicjowaniem retransmisji, na rozpoznanie, że została tylko zmieniona kolejność pakietów i nie doszło do ich utraty.

- **Skuteczne retransmisje**: Kiedy odbiorca otrzymuje segment, który wypełnia lukę w buforze odbiorczym (np. z powodu pomyślnego zakończenia retransmisji), okno na ogół przeskakuje do przodu. W tym przypadku wartość przekazywana w polu TSER odpowiedniego potwierdzenia ACK pochodzi z segmentu, który przyszedł jako ostatni. Jest to pożyteczne, ponieważ użycie wartości TSV ze starszego segmentu, którego wiek mógłby być większy niż wartość czasu RTO, doprowadziłoby do dużego, niepożądanego odchylenia w oszacowaniu czasu RTT nadawcy.

Przykład przedstawiony na rysunku 14.4 jest ilustracją tych przypadków. Załóżmy, że trzy segmenty, z których każdy zawiera po 1024 bajty, zostały odebrane w następującej kolejności: segment numer 1 z bajtami 1 – 1024, segment numer 3 z bajtami 2049 – 3072, a następnie segment numer 2 z bajtami 1025 – 2048.



Rysunek 14.4. *Kiedy segmenty docierają do odbiorcy w zmienionej kolejności, zwrócony znacznik czasu pochodzi z ostatniego segmentu, który spowodował przesunięcie okna danych odbiorcy do przodu (a nie znacznik czasu o największej wartości spośród tych, które dotarły do odbiorcy). Zawyżone oszacowania czasu RTT w okresach występowania zmian w kolejności dostarczanych pakietów wpływają na czas RTO nadawcy, redukując jego agresywność*

Odesłane potwierdzenia ACK na rysunku 14.4 to: ACK z numerem 1025 i znacznikiem czasu z segmentu numer 1 (normalne potwierdzenie ACK dla danych, które były oczekiwane), ACK z numerem 1025 i znacznikiem czasu z segmentu 1 (zduplikowane potwierdzenie ACK wysłane w odpowiedzi na segment, który dotarł poza kolejnością, ale mieści się w oknie odbiorczym), wreszcie ACK z numerem 3073 i znacznikiem czasu z segmentu numer 2 (a nie z segmentu numer 3). Daje to efekt zawyżonego oszacowania czasu RTT w sytuacji, gdy segmenty docierają w zmienionej kolejności (lub są gubione). Wyższe oszacowanie czasu RTT prowadzi do ustalenia większego czasu RTO, zmniejszając agresywność nadawcy w inicjowaniu retransmisji. Jest to szczególnie pożądane, gdy zachodzi zmiana kolejności pakietów, ponieważ agresywne retransmisje prawdopodobnie okażą się zbędne.

Tak więc zobaczyliśmy, że opcja *Znaczniki czasu* umożliwia nadawcy wykonywanie oszacowań czasu RTT nawet wtedy, gdy występują opóźnienia w dostarczaniu pakietów, zdarzenia utraty pakietów lub zmiany kolejności ich dostarczania. Nadawca może

mierzyć czas RTT, używając w opcjach takich wartości, jakich sobie życzy, ale wykorzystywane jednostki muszą przynajmniej być proporcjonalne do rzeczywistego czasu i posiadać rozsądną ziarnistość (*granularity*), aby były kompatybilne z numerami sekwencyjnymi TCP i prawdopodobnymi szybkościami łączy (więcej szczegółów na ten temat można znaleźć w [RFC1323]). Aby zegar TCP był pożyteczny dla nadawcy, musi „tyknąć" przynajmniej raz dla każdego możliwego czasu RTT. Z drugiej strony, nie powinien zmieniać swej zawartości szybciej niż raz na 59 ns. Jeśli tak by się działo, 32-bitowe pole TSV przechowujące wartość zegara TCP mogłoby się przepełnić w ciągu maksymalnego czasu, przez jaki warstwa IP pozwala istnieć w sieci pojedynczemu pakietowi (255 s; patrz [ID1323b]). Zakładając, że wszystko to jest prawidłowe, można teraz wykorzystać wartość czasu RTO do uruchamiania retransmisji.

14.4. Retransmisje na podstawie licznika czasu

Kiedy nadawczy protokół TCP ustali swój czas RTO na podstawie pomiarów zmieniających się w czasie wartości efektywnego czasu RTT, za każdym razem, kiedy wysyła segment, zapewnia odpowiednie ustawienie licznika czasu retransmisji. W trakcie ustawiania licznika czasu retransmisji zapisywany jest numer sekwencyjny tzw. synchronizowanego segmentu (*timed segment*) i jeśli potwierdzenie ACK będzie odebrane w przewidzianym czasie, licznik czasu retransmisji zostanie anulowany. Gdy nadawca wysyła pakiet zawierający dane po raz kolejny, ustawiany jest nowy licznik czasu retransmisji, stary licznik jest anulowany oraz zapisywany jest nowy numer sekwencyjny. Wobec tego nadawczy protokół TCP ustawicznie ustawia i anuluje jeden licznik czasu retransmisji dla połączenia; jeżeli nie dochodzi nigdy do utraty danych, żaden licznik czasu retransmisji nie sygnalizuje przeterminowania.

Powyższa obserwacja stanowiła pewnego rodzaju niespodziankę dla projektantów systemów operacyjnych hostów. W typowym systemie operacyjnym liczniki czasu są używane do sygnalizacji szerokiego wachlarza zdarzeń, a implementacja mechanizmu licznika czasu jest tak dostrojona, aby w efektywny sposób ustawiać liczniki i obsługiwać zdarzenie zakończenia odliczania (co wiąże się z wywołaniem funkcji systemowej). Jednak w protokole TCP głównym wymogiem jest efektywne ustawianie i resetowanie, czyli anulowanie liczników czasu; jeżeli zatem TCP dobrze funkcjonuje, liczniki czasu nigdy nie sygnalizują przeterminowania.

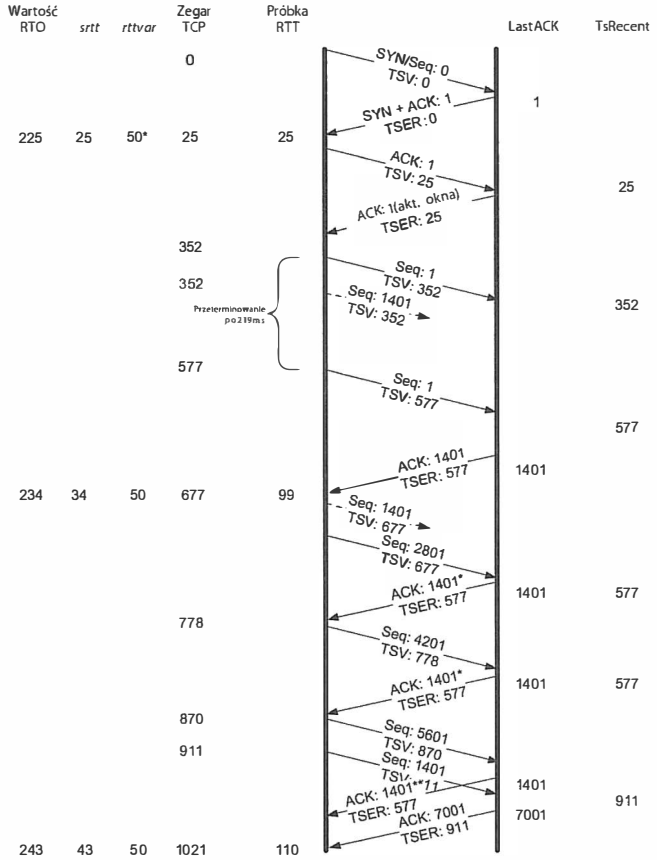
Kiedy TCP nie otrzymuje potwierdzenia ACK dla segmentu objętego w połączeniu pomiarem czasu w okresie czasu wyznaczonym przez wartość RTO, wykonuje retransmisję na podstawie licznika czasu. Widzieliśmy już tę sytuację na rysunku 14.1. Protokół TCP traktuje retransmisję na podstawie licznika czasu jako dosyć poważne zdarzenie; kiedy do niego dochodzi, reaguje bardzo ostrożnie, szybko zmniejszając tempo wysyłania danych do sieci. Robi to na dwa sposoby. Pierwszy z nich polega na zmniejszeniu rozmiaru nadawczego okna danych w oparciu o procedury kontroli przeciążenia (patrz rozdział 16.). Drugi sposób polega na ciągłym zwiększaniu wynikającego z procedury odczekiwania mnożnika stosowanego do czasu RTO za każdym razem, gdy retransmitowany segment jest retransmitowany ponownie. Jest on zaimplementowany w „drugiej części" wspomnianego wcześniej algorytmu Karna. Konkretnie mówiąc, wartość czasu RTO jest (przez pewien czas) mnożona przez liczbę γ, aby utworzyć limit czasu uwzględniający odczekiwanie, kiedy dochodzi do wielu retransmisji tego samego segmentu:

$$RTO = \gamma \cdot RTO$$

W zwykłych okolicznościach γ ma wartość 1. Przy kolejnych retransmisjach wartość γ jest podwajana: 2, 4, 8 itd. Istnieje zazwyczaj maksymalny współczynnik odczekiwania, którego wartość γ nie może przekroczyć (Linux zapewnia, że wykorzystywany czas RTO nigdy nie przekracza wartości TCP_RTO_MAX, która domyślnie wynosi 120 s). Kiedy tylko zostanie odebrane akceptowalne potwierdzenie ACK, wartość γ jest z powrotem ustawiana na 1.

14.4.1. Przykład

Możemy przyjrzeć się działaniu licznika czasu retransmisji, tworząc połączenie podobne do tego, jakie widzieliśmy na rysunkach 14.1 i 14.2, ale w którym celowo spowodowaliśmy dwukrotną utratę segmentu z numerem sekwencyjnym 1401 (patrz rysunek 14.5).



Rysunek 14.5. *Dwukrotnie wymuszono utratę segmentu 1401. To powoduje retransmisję na podstawie licznika czasu u nadawcy. Wartości zmiennych srtt, rttvar i czasu RTO są aktualizowane tylko przez powracające potwierdzenie ACK, które powoduje przesunięcie okna danych nadawcy do przodu. Potwierdzenia ACK oznaczone gwiazdką (*) zawierają informacje SACK*

W tym przykładzie przepuszczamy wysyłane segmenty przez specjalną funkcję, która może wymuszać ich utratę określoną liczbę razy na podstawie numerów sekwencyjnych nadanych przez TCP. Dokłada to nieco dodatkowego opóźnienia do czasu RTT w porównaniu z rysunkiem 14.2. Połączenie rozpoczyna się, tak jak poprzednio, z wyjątkiem tego, że kiedy zostaje wysłana para segmentów z numerami sekwencyjnymi 1 i 1401, drugi pakiet zostaje usunięty. Przypuszczalnie pierwszy z tych segmentów dociera do odbiorcy, ale odbiorca opóźnia wysyłanie potwierdzeń ACK i nie odpowiada natychmiast. Brak odpowiedzi w ciągu 219 ms sprawia, że licznik czasu retransmisji u nadawcy sygnalizuje przeterminowanie, co powoduje ponowne wysłanie pakietu z numerem sekwencyjnym 1 (tym razem z wartością 577 w polu TSV). Jego przyjście generuje potwierdzenie ACK od odbiorcy, które wraca do nadawcy. Ponieważ to ACK potwierdza dane i przesuwa do przodu okno nadawcy, przekazana w nim wartość TSER zostaje użyta do aktualizacji zmiennej srtt i czasu RTO przez przypisanie im odpowiednio wartości 34 i 234.

Następne trzy potwierdzenia ACK są generowane w odpowiedzi na pakiety, które przychodzą do odbiorcy. Potwierdzenia ACK oznaczone gwiazdką (*) są wszystkie zdublowanymi potwierdzeniami i zawierają informacje SACK. Znaczenie zduplikowanych potwierdzeń ACK i opcji SACK omówimy w podrozdziałach 14.5 i 14.6. Na razie, ponieważ te potwierdzenia nie przesuwają okna nadawcy do przodu, ich wartości TSER nie są wykorzystywane.

Wraz z ostatnią retransmisją i dotarciem segmentu 1401 (w czasie 911 wg zegara TCP) do odbiorcy okres naprawy kończy się i odbiorca odpowiada numerem ACK 7001, co wskazuje, że wszystkie dane zostały odebrane.

Licznik czasu retransmisji dostarcza czegoś w rodzaju „ostatniej deski ratunku" połączeniu TCP, które przestało regularnie przesyłać dane przez sieć. W większości przypadków nie jest potrzebne (ani pożądane) wywoływanie retransmisji przez liczniki czasu, ponieważ czas RTO jest ustalany na wartość większą od typowego czasu RTT (z współczynnikiem bezpieczeństwa wynoszącym 2 lub więcej); tak więc retransmisja na podstawie licznika czasu często prowadzi do niepełnego wykorzystania pojemności sieci. Na szczęście, TCP ma jeszcze inną metodę wykrywania i naprawiania zgubionych pakietów, która jest prawie zawsze bardziej wydajna niż retransmisje na podstawie licznika czasu. Jest ona nazywana szybką retransmisją (*fast retransmit*), ponieważ jej uruchomienie nie wymaga sygnalizacji przeterminowania przez licznik czasu retransmisji.

14.5. Szybka retransmisja

Szybka retransmisja (patrz [RFC5681]) jest procedurą protokołu TCP, która może wywołać retransmisję pakietu na podstawie informacji zwrotnej od odbiorcy, nie wymagając sygnalizacji przeterminowania przez licznik czasu. W rezultacie utrata pakietów może być szybciej i efektywniej naprawiona przy użyciu szybkiej retransmisji niż retransmisji na podstawie licznika czasu. Typowy protokół TCP zawiera implementację zarówno szybkiej retransmisji, jak i retransmisji na podstawie licznika czasu. Zanim opiszemy szybką retransmisję bardziej szczegółowo, jest ważne, abyśmy sobie uświadomili, że wymaga się od protokołu TCP wygenerowania natychmiastowego potwierdzenia (tzw. zduplikowanego ACK), kiedy zostanie odebrany segment poza kolejnością, i że z utraty

segmentu wynika przychodzenie następnych danych poza kolejnością. Kiedy tak się dzieje, u odbiorcy powstają luki. Wtedy zadaniem nadawcy staje się zapełnienie luk, które pojawiły się u odbiorcy, tak szybko i efektywnie, jak to tylko możliwe.

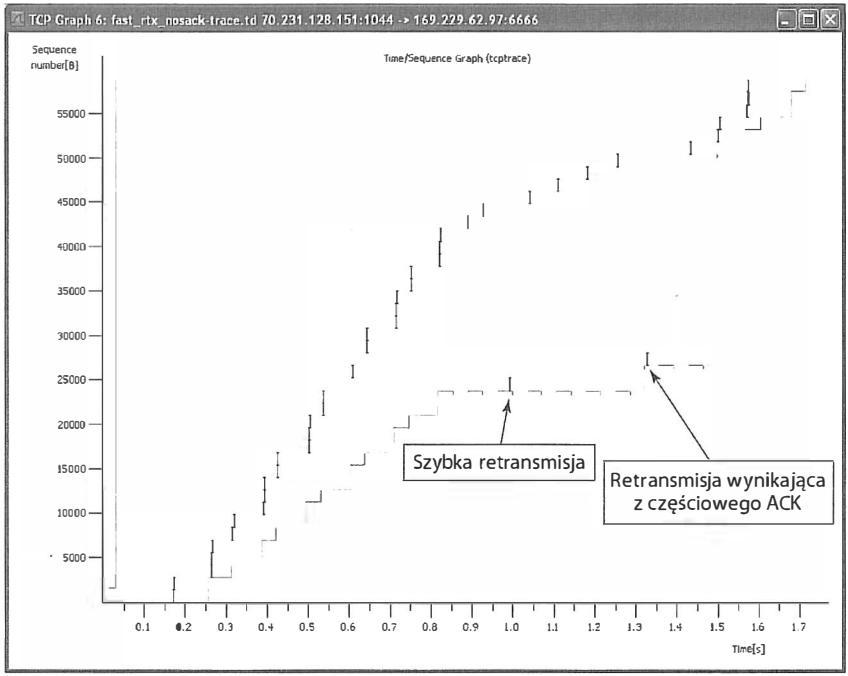
Zduplikowane potwierdzenia ACK wysyłane bezpośrednio po przyjściu danych poza kolejnością nie są opóźniane. Ich celem jest powiadomienie nadawcy, że został odebrany segment poza kolejnością i wskazanie, jaki numer sekwencyjny jest oczekiwany (tzn. gdzie znajduje się luka). Jeżeli używana jest opcja SACK, te zduplikowane potwierdzenia ACK zwykle zawierają także bloki SACK, które mogą dostarczyć informacji o więcej niż jednej luce.

Zduplikowane potwierdzenie ACK (zawierające bloki SACK lub nie) przychodzące do nadawcy jest potencjalnym wskaźnikiem, że wysłany wcześniej pakiet został utracony. Jak pokaże bardziej szczegółowa analiza, którą przeprowadzimy w podrozdziale 14.8, zduplikowane potwierdzenia ACK mogą się również pojawić, kiedy nastąpi w sieci zmiana kolejności pakietów — jeśli odbiorca otrzymuje pakiet z numerem sekwencyjnym przekraczającym następną oczekiwaną wartość, to albo oczekiwany pakiet został utracony, albo jedynie opóźniony. Ponieważ na ogół nie wiemy, który przypadek miał miejsce, protokół TCP czeka na niewielką liczbę zduplikowanych potwierdzeń ACK (nazywaną **progiem ilości zduplikowanych potwierdzeń ACK**, *duplicate ACK threshold* lub *dupthresh*), które należy odebrać przed dojściem do wniosku, że pakiet został utracony, i zainicjowaniem szybkiej retransmisji. Tradycyjnie próg dupthresh był stałą (o wartości 3), ale niektóre niestandardowe implementacje (w tym system Linux) zmieniają wartość tego progu na podstawie aktualnie zmierzonego poziomu zmian kolejności pakietów (patrz podrozdział 14.8).

Protokół TCP nadawcy, który odnotuje przynajmniej progową ilość zduplikowanych potwierdzeń ACK, retransmituje jeden lub więcej pakietów wyglądających na utracone bez czekania na sygnalizację przeterminowania przez licznik czasu. Może również wysłać nowe dane, które jeszcze nie zostały wysłane. To jest istota algorytmu szybkiej retransmisji. Zakłada się, że utrata pakietów wywnioskowana z obecności zduplikowanych potwierdzeń ACK jest związana z przeciążeniem sieci, a procedury kontroli przeciążenia (omawiane w rozdziale 16.) są uruchamiane równolegle z szybką retransmisją. Bez opcji SACK jest zwykle retransmitowany najwyżej jeden segment, dopóki nie zostanie odebrane potwierdzenie ACK. W przypadku użycia opcji SACK potwierdzenia ACK zawierają dodatkową informację umożliwiającą nadawcy wypełnienie więcej niż jednej luki w danych otrzymanych przez odbiorcę w ciągu jednego czasu RTT. Użycie opcji SACK z szybką retransmisją zbadamy po przedstawieniu przykładu podstawowego algorytmu szybkiej retransmisji.

14.5.1. Przykład

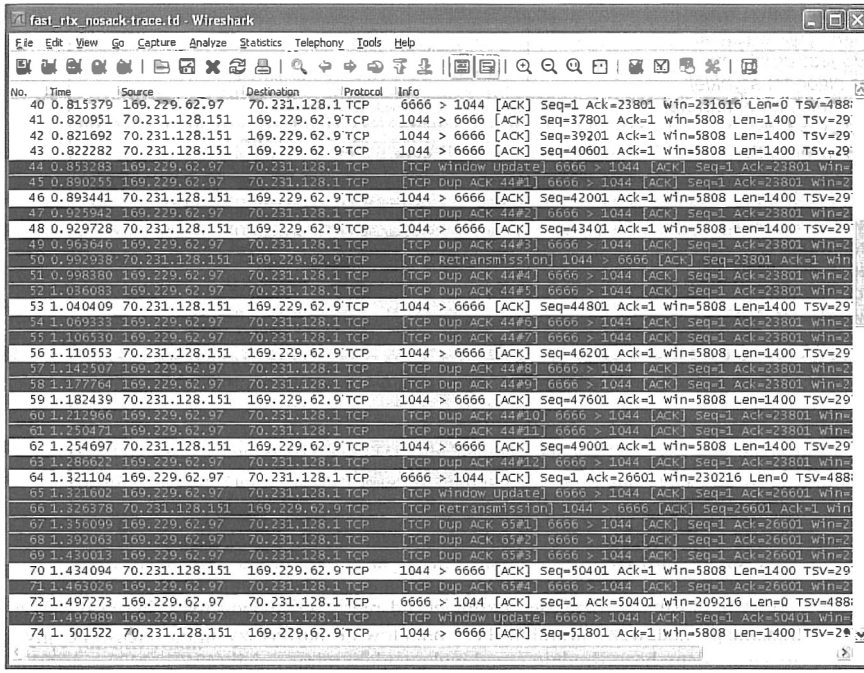
W poniższym przykładzie utworzymy połączenie TCP podobne do przedstawionego na rysunku 14.4, jednak tym razem spowodujemy utratę segmentów 23801 i 26601, a opcja SACK będzie wyłączona. Zobaczymy, jak TCP korzysta z podstawowego algorytmu szybkiej retransmisji, aby naprawić te luki. Nadawcą jest system Linux 2.6, a odbiorcą — system FreeBSD 5.4. Na wykresie z rysunku 14.6, przedstawiającym zrzut ekranu programu Wireshark po wybraniu z menu polecenia *Statistics/TCP Stream Graph/Time-Sequence Graph (tcptrace)*, można zobaczyć szybką retransmisję w działaniu.



Rysunek 14.6. *Na tym wykresie numery sekwencyjne są przedstawione na osi Y, a czas na osi X. Segmenty wychodzące są wyświetlone w postaci odcinków narysowanych ciemniejszą linią, a przychodzące numery ACK są pokazane jako odcinki w jaśniejszym odcieniu szarości. Szybka retransmisja została wywołana przez przyjście trzeciego zduplikowanego potwierdzenia ACK w czasie 0,993 s. Połączenie nie używa opcji SACK, więc jest w stanie naprawić co najwyżej jedną lukę na pojedynczy RTT. Dodatkowe zduplikowane potwierdzenia ACK, przychodzące po trzecim potwierdzeniu, skłaniają nadawcę do wysłania nowych segmentów (nie są to retransmisje). Tzw. „częściowe ACK" przychodzące w czasie 1,32 s powoduje następną retransmisję*

Na osi Y wykresu zaznaczone są względne numery sekwencyjne wysyłanych danych, a na osi X czas, który upłynął od momentu początkowego. Czarne, pionowe elementy w kształcie litery I pokazują zakres numerów sekwencyjnych zawartych w transmitowanym segmencie. Niebieskie linie wyświetlane przez program Wireshark (dolna jasnoszara linia na rysunku 14.6) wskazują numery ACK w zwracanych pakietach. Mniej więcej w czasie 1,0 retransmitowany jest numer sekwencyjny 23801 w wyniku algorytmu szybkiej retransmisji (pierwsza transmisja nie jest widoczna, ponieważ została odrzucona przez proces działający u nadawcy poniżej warstwy protokołu TCP). Retransmisja jest wywołana przybyciem trzeciego zduplikowanego potwierdzenia ACK, co ilustrują powtarzające się (na tym samym poziomie) odcinki dolnej linii. Retransmisje są również widoczne na ekranie programu Wireshark przedstawiającym podstawową analizę (patrz rysunek 14.7).

Pierwszy wiersz rysunku 14.7 (oznaczony numerem 40) wskazuje moment, kiedy potwierdzenie z numerem ACK 23801 zostało odebrane po raz pierwszy. Program podświetla (w kolorze czerwonym wyglądającym na rysunku 14.7 jak czarny) inne „interesujące" pakiety TCP. Takie pakiety różnią się od tego, czego można by oczekiwać w przypadku

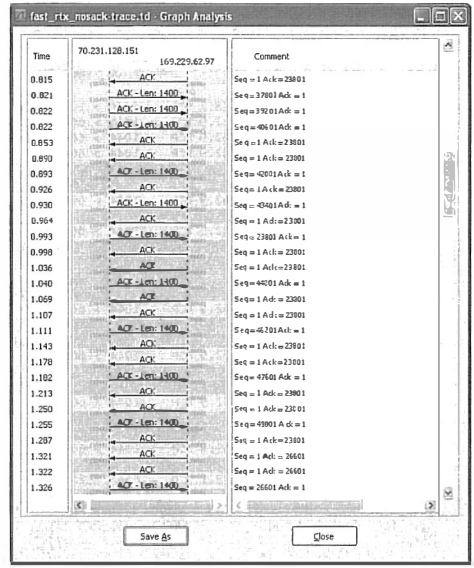


Rysunek 14.7. *Wymiana pakietów w połączeniu TCP pokazująca względne numery sekwencyjne. Pakiety 50. i 66. są retransmisjami. Pakiet 50. jest retransmitowany w wyniku działania algorytmu szybkiej retransmisji uruchomionego przyjściem trzech zduplikowanych potwierdzeń ACK. Nie jest potrzebny żaden licznik czasu retransmisji, więc proces odtwarzania przebiega stosunkowo szybko*

transferu TCP wolnego od utraty segmentów i innych anomalii. Aktualizacja okna w czasie 0,853 jest potwierdzeniem ACK ze zduplikowanym numerem sekwencyjnym (ponieważ nie są przekazywane żadne dane), ale zawiera zmianę okna sterowania przepływem protokołu TCP. Okno zmienia swój rozmiar z 231 616 bajtów na 233 016 bajtów. Dlatego też potwierdzenie to nie jest liczone w procedurze sprawdzającej, czy został osiągnięty próg trzech zduplikowanych potwierdzeń ACK wymagany do zainicjowania szybkiej retransmisji. Aktualizacje okna jedynie dostarczają kopię propozycji okna. Przyjrzymy się im bardziej szczegółowo w rozdziale 15.

Pakiety przychodzące w momentach czasu 0,890 s, 0,926 s i 0,964 s są wszystkie zduplikowanymi potwierdzeniami ACK dla numeru sekwencyjnego 23801. Przybycie trzeciego z tych zduplikowanych potwierdzeń uruchamia szybką retransmisję segmentu 23801 w czasie 0,993. Można to również zobaczyć, używając funkcji *Statistics/Flow Graph* programu Wireshark (patrz rysunek 14.8).

Na powyższym rysunku pokazujemy w nieco inny sposób tę samą szybką retransmisję w czasie 0,993. Widzimy także, że w czasie 1,326 ma miejsce druga retransmisja. Ta druga retransmisja ma miejsce z powodu przybycia potwierdzenia ACK w czasie 1,322.



Rysunek 14.8. *Retransmisja w czasie 0,993 jest uruchomiona przez algorytm szybkiej retransmisji po odebraniu zduplikowanych potwierdzeń ACK w czasach 0,890, 0,926 i 0,964. Potwierdzenie ACK w czasie 0,853 nie jest uważane za zduplikowane potwierdzenie ACK, ponieważ zawiera aktualizację okna*

Druga retransmisja różni się cokolwiek od pierwszej. Kiedy ma miejsce pierwsza retransmisja, nadawczy protokół TCP odnotowuje najwyższy numer sekwencyjny, jaki został wysłany bezpośrednio przed wykonaniem retransmisji (43401+1400 = 44801). Nazywamy go **punktem odtwarzania** (*recovery point*). Uważa się, że protokół TCP znajduje się w stanie odzyskiwania straty po retransmisji, dopóki nie otrzyma potwierdzenia ACK zgodnego z numerem sekwencyjnym punktu odtwarzania lub przekraczającego ten numer. W tym przykładzie potwierdzenia ACK w czasach 1,322 i 1,321 nie dotyczą numeru 44801, ale numeru 26601. Numer ten jest większy od dotychczasowej najwyższej wartości numeru ACK (23801), ale nie jest wystarczająco duży, by osiągnąć lub przekroczyć punkt odtwarzania (44801). Z tego powodu ten rodzaj ACK został nazwany **częściowym potwierdzeniem ACK** (*partial ACK*). Kiedy przychodzą częściowe potwierdzenia ACK, nadawczy protokół TCP natychmiast wysyła segment, którego brak się ujawnił (26601 w tym przypadku) i dalej działa w ten sam sposób, dopóki nie zostanie osiągnięty lub przekroczony punkt odtwarzania przez przychodzące potwierdzenie ACK. Jeśli pozwalają na to procedury kontroli przeciążenia (patrz rozdział 16.), może transmitować nowe, jeszcze niewysłane dane.

Przykład ten ilustruje działanie protokołu TCP, który nie używa opcji SACK, kiedy korzysta z szybkiej retransmisji i kiedy wykonuje dodatkowe retransmisje w czasie procesu odtwarzania w oparciu o nadawczy algorytm *NewReno* (patrz [RFC3782]). Ponieważ nie są używane opcje SACK, nadawca może dowiedzieć się o co najwyżej jednej luce u odbiorcy w pojedynczym cyklu transmisji (*round-trip time*); dopiero wzrost numeru ACK w zwracanych pakietach, który może wystąpić dopiero wtedy, gdy retransmisja wypełniająca lukę z najniższym numerem sekwencyjnym zostanie odebrana i potwierdzona przez ACK, może wskazać kolejną lukę.

Dokładny sposób działania w trakcie procesu odtwarzania różni się w zależności od typu i konfiguracji strony nadawczej i odbiorczej protokołu TCP. Ten przykład ilustruje nadawcę nieużywającego opcji SACK, który korzysta z algorytmu *NewReno*, co stanowi dość często występujący układ. W algorytmie *NewReno* częściowe potwierdzenia ACK utrzymują nadawcę w stanie odtwarzania, jak to opisano wcześniej. W starszych wariantach protokołu TCP (zwykły Reno, *plain Reno*) pojęcie częściowego potwierdzenia nie istnieje, a każde akceptowalne potwierdzenie ACK wyprowadza TCP ze stanu odtwarzania. Ten sposób postępowania może sprawić protokołowi TCP pewne problemy związane z wydajnością, które są omawiane szczegółowo w rozdziale 16. Algorytm *NewReno* i opcja SACK, którą omówimy w następnej kolejności, są czasem nazywane technikami „zaawansowanego odzyskiwania strat" w celu ich odróżnienia od starszych metod.

14.6. Retransmisja z potwierdzeniami selektywnymi

Korzystając ze standaryzacji opcji *Selektywne potwierdzenia* zawartej w dokumencie [RFC2018], odbiorczy protokół TCP obsługujący opcję SACK może opisać otrzymane dane, posiadające numery sekwencyjne przekraczające kumulatywną wartość pola *Numer ACK* przesyłanego w głównej części nagłówka TCP. Jak wspomnieliśmy wcześniej, przerwy między aktualnym numerem ACK a innymi danymi mieszczącymi się w oknie, buforowanymi przez odbiorcę, są nazywane lukami. Dane lokowane przez swe numery sekwencyjne za lukami są nazywane danymi poza kolejnością, ponieważ nie stanowią pod względem numerów sekwencyjnych ciągłej kontynuacji innych danych, które odbiorca już otrzymał.

Zadaniem nadawczego TCP jest wypełnianie luk u odbiorcy przez retransmitowanie wszystkich danych, których brakuje odbiorcy, z zachowaniem jednak takiej wydajności, jaka tylko jest możliwa, przez unikanie wysyłania danych, które odbiorca już posiada. W wielu przypadkach właściwie działający nadawca używający opcji SACK potrafi wypełniać te luki szybciej i z mniejszą ilością niepotrzebnych retransmisji niż porównywalny nadawca nieużywający opcji SACK, ponieważ nie musi czekać przez cały okres RTT, aby dowiedzieć się o dodatkowych lukach. Kiedy jest używana opcja SACK, potwierdzenie ACK może być wzbogacone o maksymalnie trzy lub cztery bloki SACK, które zawierają informację o danych poza kolejnością znajdujących się u odbiorcy. Każdy blok SACK zawiera dwie 32-bitowe wartości reprezentujące pierwszy i ostatni numer sekwencyjny (zwiększony o 1) ciągłego bloku danych poza kolejnością przechowywanych przez odbiorcę.

Opcja SACK, która definiuje n bloków, ma (w bajtach) długość $8n+2$, więc 40 bajtów dostępnych (w nagłówku) do przechowywania opcji TCP wyznacza maksimum, czyli cztery bloki. Oczekuje się, że opcja SACK będzie często używana w połączeniu z opcją TSOPT, która zajmuje dodatkowe 10 bajtów (plus 2 bajty dopełnienia), co oznacza, że w typowym potwierdzeniu ACK opcja SACK może zawierać tylko trzy bloki.

Przez specyfikację trzech różnych bloków można zgłosić nadawcy aż trzy luki jednocześnie. Jeśli nie narusza to ograniczeń wynikających z kontroli przeciążenia (patrz rozdział 16.), wszystkie trzy luki mogą zostać wypełnione w ciągu jednego czasu RTT, jeśli nadawca obsługuje opcję SACK. Pakiet ACK zawierający jeden lub więcej bloków SACK jest czasem nazywany po prostu „pakietem SACK".

14.6.1. Zachowanie odbiorcy obsługującego opcję SACK

Zdolny do obsługi opcji SACK odbiorca może generować segmenty SACK, jeśli otrzymał opcję SACK-Permitted w czasie ustanawiania połączenia TCP (patrz rozdział 13.). Mówiąc ogólnie, odbiorca generuje segmenty SACK, ilekroć w jego buforze znajdują się jakiekolwiek dane poza kolejnością. Może się to zdarzyć albo na skutek utraty danych w trakcie transmisji, albo w wyniku zmiany kolejności ich dostarczania, która sprawiła, że nowsze dane dotarły do odbiorcy przed starszymi. Teraz przyjrzymy się pierwszemu przypadkowi, a drugi omówimy później.

Odbiorca umieszcza w pierwszym bloku SACK zakres numerów sekwencyjnych zawartych w segmencie, który został **odebrany jako ostatni**. Ponieważ ilość miejsca w opcji SACK jest ograniczona, najlepiej upewnić się, o ile to możliwe, że najświeższe informacje są zawsze dostarczane do nadawczego TCP. Pozostałe bloki SACK są umieszczane w takiej kolejności, w jakiej występowały jako pierwsze bloki w poprzednich opcjach SACK. Znaczy to, że są wpisywane przez powielenie ostatnio wysłanych bloków SACK (w innych segmentach), które nie są podzbiorami innego bloku, jaki ma być umieszczony w aktualnie tworzonej opcji.

Celem umieszczania więcej niż jednego bloku SACK w opcji SACK i powtarzania tych bloków w wielu kolejnych segmentach SACK jest zapewnienie pewnej redundancji, na wypadek gdyby segmenty SACK zostały zgubione. Dokument [RFC2018] wykazuje, że gdyby segmenty SACK nigdy nie były tracone, wystarczyłby tylko jeden blok SACK w każdej opcji SACK dla zapewnienia pełnej funkcjonalności tej opcji. Niestety, pakiety SACK i zwykłe potwierdzenia ACK są czasem gubione i nie są retransmitowane przez TCP, chyba że zawierają dane (albo bit sterujący SYN lub FIN jest włączony).

14.6.2. Zachowanie nadawcy obsługującego opcję SACK

Chociaż dla pełnego wykorzystania opcji SACK jest niezbędne, aby obsługujący tę opcję odbiorca generował właściwe informacje SACK, nie wystarczy to jednak, aby połączenie TCP mogło z tej opcji skorzystać. Musi funkcjonować nadawca zdolny do obsługi opcji SACK, który prawidłowo potraktuje informacje zawarte w blokach SACK i wykona **selektywną retransmisję** (*selective retransmission*), wysyłając tylko te segmenty, których brakuje u odbiorcy; proces ten jest również nazywany **selektywnym powtórzeniem** (*selective repeat*). Nadawca obsługujący opcję SACK śledzi nie tylko kumulatywne informacje przekazywane przez odbierane potwierdzenia ACK (podobnie jak każdy nadawczy protokół TCP), ale także wszystkie informacje SACK, które otrzymuje. Wykorzystuje informacje SACK, które otrzymuje w potwierdzeniach ACK generowanych przez odbiorcę, aby uniknąć retransmitowania danych, o których odbiorca raportuje, że je posiada. Jednym ze sposobów realizacji tego celu jest utrzymywanie dla każdego segmentu znajdującego się w buforze retransmisji specjalnego wskaźnika SACKed, który jest ustawiany, ilekroć odpowiedni zakres numerów sekwencyjnych przychodzi w opcji SACK.

Kiedy obsługujący opcję SACK nadawca ma okazję wykonania retransmisji, zwykle dlatego, że otrzymał pakiet SACK lub odnotował kilka zduplikowanych potwierdzeń ACK, staje przed wyborem, czy wysyłać nowe dane, czy retransmitować stare. Informacje SACK zawierają zakresy numerów sekwencyjnych danych znajdujących się u odbiorcy, więc nadawca może wywnioskować, które segmenty prawdopodobnie wymagają re-

transmisji, aby wypełnić luki u odbiorcy. Najprostszym podejściem jest w pierwszej kolejności wypełnienie luk u odbiorcy, a w dalszym kroku wysyłanie nowych danych (patrz [RFC3517]), jeśli pozwolą na to procedury kontroli przeciążenia. Jest to najbardziej popularne podejście.

Istnieje jeden wyjątek odnoszący się do tego zachowania. W dokumencie [RFC2018], zawierającym aktualną specyfikację opcji SACK, bloki SACK są traktowane jako **konsultatywne** (*advisory*). Oznacza to, że odbiorca mógłby wysłać nadawcy opcję SACK wskazującą, że pewne numery sekwencyjne zostały odebrane z sukcesem, a potem zmienić swoje zdanie i wycofać się z poprzedniej deklaracji (*renege*). Z tego powodu nadawca obsługujący opcję SACK nie może ze swojego bufora retransmisji usunąć danych, dla których tylko otrzymał potwierdzenie SACK; wolno mu zwolnić blok danych dopiero wtedy, kiedy zwykły numer ACK protokołu TCP otrzymany od odbiorcy przekroczy najwyższą wartość numeru sekwencyjnego tych danych. Ta zasada[1] dotyczy również tego, co powinien zrobić protokół TCP, kiedy licznik czasu retransmisji zgłosi przeterminowanie. Kiedy nadawczy protokół TCP inicjuje retransmisję na podstawie licznika czasu, wszelkie informacje dotyczące danych poza kolejnością znajdujących się u odbiorcy pochodzące z opcji SACK powinny zostać zapomniane. Jeśli dane poza kolejnością pozostają u odbiorcy, potwierdzenie ACK dla retransmitowanego segmentu zawiera dodatkowe bloki SACK, które teraz mogą być wykorzystane przez nadawcę. Na szczęście, wycofywanie wcześniejszej informacji SACK (*reneging*) zdarza się rzadko i jest odradzane.

14.6.3. Przykład

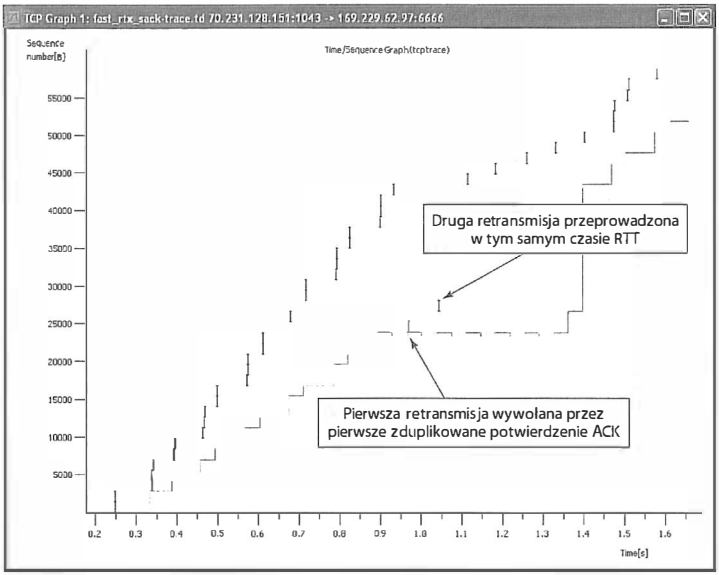
Aby zrozumieć, w jaki sposób użycie opcji SACK zmienia zachowanie nadawcy i odbiorcy, powtarzamy wcześniejszy eksperyment z szybką retransmisją przy tej samej konfiguracji (wymuszenie utraty numerów sekwencyjnych 23601 i 28801), ale tym razem nadawca i odbiorca używają opcji SACK. Aby uzyskać natychmiastowy pogląd na to, co się dzieje, znowu używamy funkcji programu Wireshark wyświetlającej wykres numerów sekwencyjnych TCP (`tcptrace`; patrz rysunek 14.9).

Rysunek 14.9 jest podobny do rysunku 14.6, ale obsługujący opcję SACK nadawca nie musiał odczekać czasu RTT, by retransmitować utracony segment 28801 po retransmisji segmentu 23601. Jest to efekt informacji SACK zawartych w przychodzących potwierdzeniach ACK. Przypatrzymy się im później szczegółowo, ale najpierw sprawdzimy negocjowanie opcji `SACK-Permitted` wykonywane w czasie konfigurowania połączenia. Można je zobaczyć na rysunku 14.10.

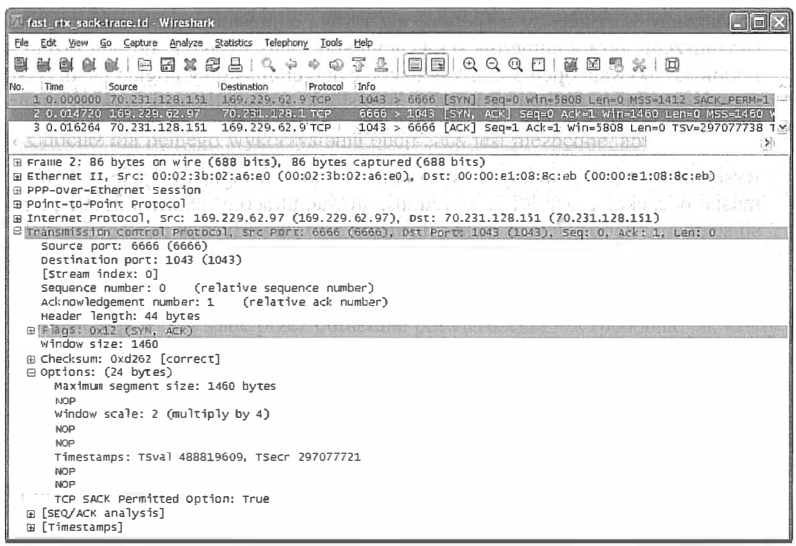
Jak można się spodziewać, odbiorca pokazuje swą zdolność do używania opcji SACK za pomocą opcji `SACK-Permitted`. Pakiet SYN od nadawcy, czyli pierwszy pakiet zapisu śledzenia, zawiera identyczną opcję. Te opcje są obecne tylko przy konfigurowaniu połączenia i dlatego występują tylko w segmentach z ustawionym bitem *SYN*.

Kiedy już połączenie ma pozwolenie na używanie opcji SACK, utrata pakietu powoduje na ogół rozpoczęcie tworzenia opcji SACK przez odbiorcę. Przykładowo program Wireshark pokazuje zawartość opcji SACK po wybraniu pierwszego segmentu zawierającego tę opcję (patrz rysunek 14.11).

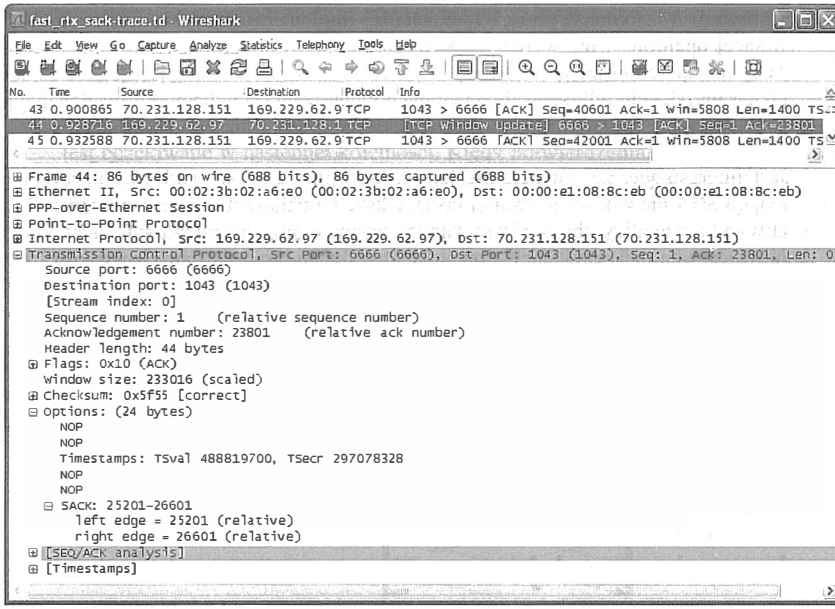
[1] Zasada konsultatywnego charakteru potwierdzeń SACK — *przyp. tłum.*



Rysunek 14.9. *Szybka retransmisja została wywołana przez przyjście pierwszego zduplikowanego potwierdzenia ACK zawierającego informację SACK. Przyjście następnego pakietu ACK pozwala nadawcy dowiedzieć się o drugim brakującym segmencie i retransmitować go w trakcie tego samego czasu RTT*



Rysunek 14.10. *Opcja SACK-Permitted jest wymieniana w segmentach SYN, by pokazać zdolność do generowania i przetwarzania informacji SACK. Większość współczesnych implementacji protokołu TCP obsługuje opcje MSS, Znaczniki czasu, Skalowanie okna i SACK-Permitted w czasie ustanawiania połączenia*



Rysunek 14.11. *Pierwsze potwierdzenie ACK zawierające informację SACK wskazuje na blok danych poza kolejnością z numerami sekwencyjnymi w zakresie od 25201 do 26601*

Na rysunku 14.11 pokazujemy ciąg zdarzeń po otrzymaniu pierwszej informacji SACK. Program Wireshark przedstawia informację SACK, wskazując lewą i prawą granicę zakresu SACK. W tym przypadku widzimy, że ACK z numerem potwierdzenia 23801 zawiera blok SACK [25201, 26601], pokazujący lukę u odbiorcy. Odbiorcy brakuje zakresu numerów sekwencyjnych [23801, 25200], który odpowiada pojedynczemu 1400-bajtowemu pakietowi rozpoczynającemu się od numeru sekwencyjnego 23801. Zauważmy, że pokazywany pakiet SACK jest również aktualizacją okna, więc nie jest liczony jako zduplikowane potwierdzenie ACK z omówionych wcześniej powodów i nie wywołuje szybkiej retransmisji.

Segment SACK przychodzący w czasie 0,967 zawiera dwa bloki SACK: [28001, 29401] i [25201, 26601]. Przypomnijmy sobie, że pierwsze bloki SACK z poprzednich segmentów SACK są powtarzane na dalszych pozycjach w kolejnych segmentach SACK ze względu na zapewnienie odporności na gubienie potwierdzeń ACK. Ten segment SACK jest zduplikowanym potwierdzeniem ACK dla numeru sekwencyjnego 23801 i sugeruje, że odbiorca wymaga teraz dostarczenia dwóch pełnowymiarowych segmentów rozpoczynających się od numerów sekwencyjnych 23801 i 26601. Nadawca reaguje natychmiast, inicjując szybką retransmisję, ale z powodu procedur kontroli przeciążenia (patrz rozdział 16.) przeprowadza tylko jedną retransmisję, dla segmentu 23801. Po przyjściu dwóch dodatkowych potwierdzeń ACK nadawca otrzymuje pozwolenie na wykonanie swojej drugiej retransmisji, dla segmentu 26601.

Nadawczy protokół TCP obsługujący opcję SACK wykorzystuje pojęcie punktu odtwarzania wprowadzonego w algorytmie *NewReno*. W tym przykładzie najwyższy numer sekwencyjny wysłany przed retransmisją jest równy 43400, czyli jest niższy niż w przykładzie

algorytmu *NewReno* z rysunku 14.5. Dla tej implementacji szybkiej retransmisji, używającej opcji SACK, nie wymaga się trzech zduplikowanych potwierdzeń ACK; protokół TCP inicjuje retransmisję wcześniej. Wyjście ze stanu odtwarzania jest jednak zasadniczo takie samo. Kiedy tylko zostaje odebrane potwierdzenie ACK dla numeru sekwencyjnego 43401 w czasie 1,3958, proces odtwarzania się kończy.

Jest interesujące, że potencjalnie lepsza kontrola ze strony nadawcy korzystającego z opcji SACK nie zawsze prowadzi do zwiększenia całkowitej przepustowości. Sugestia ta wynika z analizy obu porównywanych przykładów. Nadawca używający algorytmu *NewReno* (ale niekorzystający z opcji SACK) wykonuje transfer danych wielkości 131 074 bajtów w czasie 3,592 s. Natomiast nadawca używający opcji SACK wykonuje taki sam transfer w ciągu 3,674 s. Te dwa pomiary nie są jednak całkowicie porównywalne, ponieważ nie przebiegały w tych samych warunkach panujących w sieci (nie była to symulacja, ale test na żywo), chociaż warunki były w dużej mierze podobne. Korzyści z używania opcji SACK są wyraźniejsze, kiedy czas RTT jest duży i ilość gubionych pakietów poważna. W takich warunkach korzyści wynikające z możliwości wypełniania więcej niż jednej luki w pojedynczym czasie RTT prawdopodobnie będą bardziej znaczące.

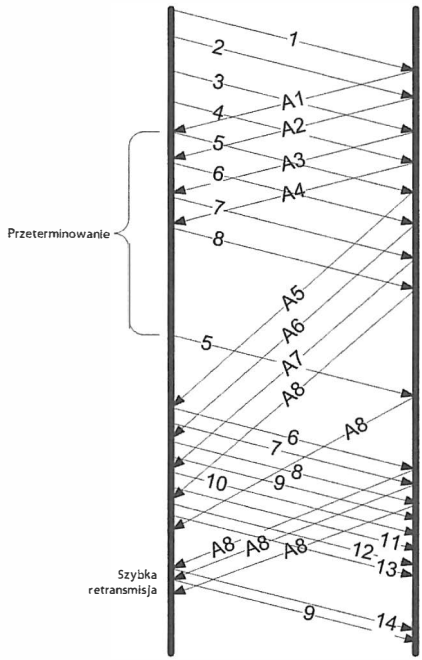
14.7. Fałszywe przeterminowania i zbędne retransmisje

W wielu przypadkach TCP może zainicjować retransmisję nawet wówczas, gdy żadne dane nie zostały zgubione. Takie niepożądane retransmisje, nazywane **zbędnymi retransmisjami** (*spurious retransmissions*), są powodowane przez **fałszywe przeterminowania** (*spurious timeouts*; przeterminowania, które są zgłaszane przedwcześnie) i inne przyczyny, takie jak zmiana kolejności pakietów, powielanie pakietów i gubienie potwierdzeń ACK. Fałszywe przeterminowania mogą się zdarzyć, gdy rzeczywisty czas RTT znacznie wzrósł w ostatnim okresie i przekroczył wartość RTO. Sytuacja ta występuje częściej w środowiskach, w których protokoły niższych warstw mają bardzo zmienną wydajność (np. w sieciach bezprzewodowych) i została przedstawiona jako problem w referacie [KP87]. W tym miejscu skupimy się głównie na zbędnych retransmisjach powodowanych przez fałszywe przeterminowania. Analizę skutków zmiany kolejności i powielania pakietów odkładamy do następnego podrozdziału.

Zaproponowano szereg podejść do kwestii fałszywych przeterminowań. Metody te na ogół zawierają algorytm **wykrywania** (*detection*) i algorytm **odpowiedzi** (*response*). Algorytm wykrywania stara się ustalić, czy przeterminowanie było fałszywe, a retransmisja na podstawie licznika czasu zbędna. Algorytm odpowiedzi jest uruchamiany wtedy, gdy przeterminowanie zostało uznane za fałszywe lub retransmisja za błędną. Jego celem jest wycofanie lub zminimalizowanie działań, które są normalnie wykonywane, kiedy licznik czasu retransmisji sygnalizuje przeterminowanie. W tym rozdziale analizujemy jedynie działania związane z retransmisją segmentów. Algorytmy odpowiedzi zwykle obejmują również zmiany w obszarze kontroli przeciążenia, a te ich aspekty są omawiane w rozdziale 16.

Na rysunku 14.12 przedstawiamy bardzo uproszczoną wymianę pakietów i pokazujemy, co się dzieje z podstawową wersją protokołu TCP, kiedy wystąpi zbędna retransmisja z powodu nagłego wzrostu opóźnienia w ścieżce potwierdzenia ACK po wysłaniu seg-

mentu numer 8. Po wystąpieniu retransmisji segmentu numer 5 z powodu przeterminowania wciąż w drodze znajdują się potwierdzenia ACK dla pierwotnych transmisji segmentów, od segmentu numer 5 do segmentu numer 8. W naszym sposobie przedstawienia numery sekwencyjne i numery ACK, dla uproszczenia, odnoszą się do pakietów zamiast do bajtów, przy czym potwierdzenia ACK wskazują, co już przyszło, a nie to, co jest oczekiwane w następnej kolejności. Kiedy potwierdzenia przychodzą do nadawcy, TCP zaczyna retransmitować kolejne segmenty, które już zostały odebrane, rozpoczynając od następnego segmentu po segmencie potwierdzonym przez ACK. To sprawia, że TCP funkcjonuje według niepożądanego wzorca zachowań właściwego dla protokołu *go-back-N*, co z kolei powoduje generowanie i odsyłanie do nadawcy kolekcji zduplikowanych potwierdzeń ACK, które mogą dodatkowo uruchomić szybką retransmisję. Opracowano kilka technik w celu zminimalizowania tych problemów. Teraz przyjrzymy się niektórym z bardziej popularnych metod.



Rysunek 14.12. *Nagły wzrost opóźnienia występuje po transmisji pakietu numer 8, powodując fałszywe przeterminowanie i retransmisję pakietu numer 5. Po retransmisji przychodzi potwierdzenie dla pierwszego egzemplarza pakietu numer 5. Retransmisja pakietu numer 5 tworzy zduplikowany pakiet u odbiorcy, po czym obserwujemy niepożądane zachowanie nadawcy typu „go-back-N", na skutek którego są retransmitowane pakiety o numerach 6, 7 i 8, mimo że znajdują się już u odbiorcy*

14.7.1. Rozszerzenie Duplicate SACK (DSACK)

W protokole TCP nieobsługującym opcji SACK potwierdzenie ACK może wskazywać nadawcy jedynie ostatni segment w ciągłej sekwencji odebranych danych. Z opcją SACK może ono również sygnalizować obecność innych (odebranych poza kolejnością) seg-

mentów u odbiorcy. Podstawowy mechanizm SACK, który wcześniej analizowaliśmy, nie mówi, co się dzieje, kiedy odbiorca otrzymuje zdublowane segmenty danych. Takie segmenty mogą być wynikiem zbędnych retransmisji, powielania pakietów w sieci lub innych przyczyn.

Duplicate SACK (zduplikowane potwierdzenie selektywne; patrz [RFC2883]), w skrócie **DSACK** lub **D-SACK**, jest regułą stosowaną przez odbiorcę obsługującego opcję SACK i interoperacyjną z konwencjonalnymi nadawcami obsługującymi opcję SACK, na mocy której pierwszy blok SACK wskazuje numery sekwencyjne zdublowanego segmentu, jaki dotarł do odbiorcy. Głównym celem informacji DSACK jest ustalenie, która retransmisja była niepotrzebna, i poznanie dodatkowych faktów dotyczących sieci. Dzięki niej nadawca ma przynajmniej możliwość wywnioskowania, czy mają miejsce zmiana kolejności pakietów, utrata potwierdzeń ACK, replikacja pakietów, czy też zbędne retransmisje.

Implementacja reguły DSACK jest kompatybilna z konwencjonalną opcją SACK w tym sensie, że nie są wymagane żadne odrębne negocjacje, by zrobić z niej użytek. Żeby działała, jak należy, potrzebna jest zmiana w zawartości opcji SACK wysyłanych przez odbiorcę i odpowiadająca jej zmiana w logice nadawcy. Gdy protokół TCP nieobsługujący informacji DSACK dzieli połączenie z protokołem TCP obsługującym DSACK, będą one współpracować, ale bez jakichkolwiek korzyści, które daje informacja DSACK.

Zmiana dotycząca odbiorcy obsługującego opcję SACK polega na możliwości dołączenia bloku SACK nawet wtedy, gdy obejmuje on numery sekwencyjne **mniejsze** od kumulatywnej wartości pola *Numer ACK* lub jej równe. Nie było to pierwotnym zamierzeniem przy tworzeniu opcji SACK, ale jej potencjał dobrze pasuje do tego nowego celu. (Możliwość ta znajduje równie dobre zastosowanie, gdy zakres numerów sekwencyjnych w informacji DSACK znajduje się powyżej kumulatywnej zawartości pola *Numer ACK*; tak się dzieje w przypadku zduplikowanych segmentów odebranych poza kolejnością). Informacja DSACK jest umieszczana tylko w pojedynczym potwierdzeniu ACK, które nazywa się wtedy potwierdzeniem DSACK. Informacja DSACK nie jest powtarzana w wielu kolejnych potwierdzeniach SACK, w przeciwieństwie do konwencjonalnych informacji SACK. W konsekwencji informacje DSACK są mniej odporne na utratę potwierdzeń ACK niż regularne informacje SACK.

Co dokładnie nadawca, który otrzymał informację DSACK, powinien z nią zrobić, nie jest określone przez dokument [RFC2883]. W dokumencie [RFC3708] został przedstawiony eksperymentalny algorytm wykrywania zbędnych retransmisji przy wykorzystaniu informacji DSACK, ale brakuje w nim algorytmu odpowiedzi. Jedyną opcją wymienioną w tym dokumencie jest użycie algorytmu odpowiedzi *Eifel*, który zbadamy w punkcie 14.7.4 po przedstawieniu kilku algorytmów wykrywania.

14.7.2. Algorytm wykrywania Eifel

Na początku tego rozdziału analizowaliśmy problem niejednoznaczności retransmisji. Eksperymentalny dokument *Eifel Detection Algorithm* ([RFC3522]) radzi sobie z tym problemem, korzystając z opcji TSOPT protokołu TCP w celu wykrywania zbędnych retransmisji. Po wystąpieniu przeterminowania powodującego retransmisję algorytm *Eifel* czeka na kolejne akceptowalne potwierdzenie ACK. Jeżeli potwierdzenie to wskazuje pierwszy egzemplarz retransmitowanego pakietu (zwany **oryginalną transmisją**, *original transmit*) jako swoją przyczynę, retransmisja jest uważana za zbędną.

Algorytm wykrywania *Eifel* potrafi wykryć zbędne działania wcześniej niż podejście wykorzystujące tylko informacje DSACK, ponieważ działa w oparciu o potwierdzenia ACK generowane w wyniku przychodzenia pakietów do odbiorcy, **zanim** zostaje zainicjowany proces odtwarzania. Informacje DSACK, na odwrót, mogą zostać wysłane dopiero po przybyciu do odbiorcy zdublowanego pakietu, a wynikające z nich działania podjęte dopiero po ich dostarczeniu do nadawcy. Wczesne wykrywanie zbędnych retransmisji może przynieść korzyści, ponieważ pozwala nadawcy na uniknięcie większości zachowań według wspomnianego wcześniej wzorca *go-back-N*.

Mechanika algorytmu wykrywania *Eifel* jest prosta. Wymaga użycia opcji TSOPT protokołu TCP. W momencie wykonywania retransmisji (na podstawie licznika czasu lub w ramach procedury szybkiej retransmisji) zapamiętywana jest wartość TSV. Kiedy zostaje odebrane pierwsze akceptowalne potwierdzenie ACK obejmujące numer sekwencyjny retransmitowanego segmentu, sprawdzana jest wartość pola TSER przychodzącego potwierdzenia. Jeśli jest ona mniejsza od zapamiętanej wartości, potwierdzenie dotyczy oryginalnej transmisji pakietu, a nie jego retransmisji, z czego wynika, że retransmisja musiała być zbędna. Rozwiązanie to jest także dość odporne na utratę potwierdzeń ACK. Jeśli jakieś potwierdzenie zostanie zgubione, wszystkie kolejne ACK będą zawierać w polu TSER wartości mniejsze od zapamiętanej wartości pola TSV retransmitowanego segmentu. Zatem retransmisja może zostać uznana za zbędną w wyniku przyjścia dowolnego spośród potwierdzeń ACK dotyczących danych z tego samego okna nadawczego, tak więc utrata pojedynczego potwierdzenia ACK najprawdopodobniej nie sprawi problemu.

Algorytm wykrywania *Eifel* może zostać użyty w połączeniu z wykorzystaniem informacji DSACK. Może to przynieść korzyść w sytuacji, w której utracone zostaną potwierdzenia ACK dla całego okna, a oryginalna transmisja i retransmisja dotrą do odbiorcy. W tym konkretnym przypadku przychodząca retransmisja powoduje wygenerowanie informacji DSACK. Algorytm wykrywania *Eifel* mógłby standardowo dojść do wniosku, że retransmisja była zbędna. Uważa się jednak, że w sytuacji, gdy traconych jest tak wiele potwierdzeń ACK, pożyteczne jest utrzymanie protokołu TCP w przekonaniu, że retransmisja **nie była zbędna** (np. by spowodować spowolnienie nadawania — w konsekwencji działania procedur kontroli przeciążenia, które omawiamy w rozdziale 16.). Tak więc przychodzące informacje DSACK sprawiają, że algorytm wykrywania *Eifel* uznaje, iż odpowiednia retransmisja **nie jest zbędna**.

14.7.3. Odtwarzanie Forward-RTO (F-RTO)

Odtwarzanie Forward-RTO (*Forward-RTO Recovery*, **F-RTO**; patrz [RFC5682]) jest standardowym algorytmem wykrywania zbędnych retransmisji. Algorytm ten nie wymaga stosowania żadnych opcji TCP, jeśli więc został zaimplementowany u nadawcy, może zostać skutecznie wykorzystany nawet w przypadku odbiorcy, który nie obsługuje opcji TSOPT protokołu TCP. Próbuje on wykrywać tylko zbędne retransmisje wywołane przeterminowaniem licznika czasu retransmisji; jego działanie nie obejmuje innych przyczyn zbędnych retransmisji lub dublowania transmisji, o których przedtem wspominaliśmy.

Algorytm F-RTO wprowadza modyfikację do działania, jakie protokół TCP podejmuje po retransmisji na podstawie licznika czasu. Retransmisje te są wykonywane dla najmniejszego numeru sekwencyjnego, który dotąd nie otrzymał potwierdzenia ACK. W zwykłym

trybie protokół TCP wysyła po kolei dalsze, sąsiednie pakiety w miarę przychodzenia następnych potwierdzeń ACK. Jest to działanie zgodne z wcześniej opisanym wzorcem *go-back-N*.

F-RTO modyfikuje zwyczajne zachowanie protokołu TCP, powodując wysłanie przez TCP nowych (dotąd niewysłanych) danych po retransmisji wynikającej z przeterminowania, kiedy przychodzi pierwsze potwierdzenie ACK. Następnie algorytm sprawdza drugie przychodzące potwierdzenie ACK. Jeśli którekolwiek z pierwszych dwóch potwierdzeń ACK, które przyszły po wysłaniu retransmisji, jest zduplikowanym potwierdzeniem ACK, retransmisja zostaje uznana za uzasadnioną. Jeżeli obydwa potwierdzenia są akceptowalnymi ACK, powodującymi przesunięcie okna nadawcy do przodu, retransmisja jest uznana za zbędną. Podejście to jest dość intuicyjne. To, że transmisja nowych danych skutkuje przyjściem akceptowalnych potwierdzeń ACK, oznacza, że przyjście nowych danych przesuwa do przodu okno odbiorcy. Jeśli zaś takie dane generują tylko zduplikowane potwierdzenia ACK, musi istnieć u odbiorcy jedna lub więcej luk. W każdym z tych przypadków otrzymanie nowych danych przez odbiorcę nie wpływa negatywnie na całkowitą szybkość transferu danych (przy założeniu, że odbiorca ma dość miejsca na buforowanie danych).

14.7.4. Algorytm odpowiedzi Eifel

Algorytm odpowiedzi *Eifel* (patrz [RFC4015]) jest standardowym zbiorem operacji, które powinny zostać wykonane przez protokół TCP, kiedy retransmisja zostanie już uznana za zbędną. Ponieważ algorytm odpowiedzi *Eifel* jest logicznie oddzielony od algorytmu wykrywania *Eifel*, może być używany razem z dowolnym omawianym tu protokołem wykrywania. Według pierwotnego zamierzenia algorytm odpowiedzi *Eifel* miał obsługiwać zarówno zbędne retransmisje wygenerowane na podstawie licznika czasu, jak i wynikające z mechanizmu szybkiej retransmisji, ale jego obecna specyfika obejmuje tylko retransmisje na podstawie licznika czasu.

Chociaż algorytm odpowiedzi *Eifel* może być używany z każdym z algorytmów wykrywania, działa nieco inaczej, w zależności od tego, czy fałszywe przeterminowanie zostało wykryte wcześniej (np. przez algorytm wykrywania *Eifel* lub F-RTO), czy później (np. przez analizę informacji DSACK). Przypadki pierwszego typu są nazywane fałszywymi przeterminowaniami i polegają na sprawdzaniu potwierdzeń ACK dla pierwotnych transmisji. Przypadki drugiego rodzaju są nazywane **późnymi** fałszywymi przeterminowaniami i opierają się na potwierdzeniach ACK dla retransmisji wywołanych w wyniku (fałszywych) przeterminowań.

Algorytm odpowiedzi reaguje tylko na pierwsze zdarzenie przeterminowania spowodowane przez licznik czasu retransmisji. Nie jest wykonywany, gdy kolejne przeterminowanie wydarzy się przed zakończeniem procesu odtwarzania. Po zgłoszeniu przeterminowania przez licznik czasu retransmisji pobierane są aktualne wartości zmiennych srtt oraz rttvar i zapisywane w nowych zmiennych srtt_prev i rttvar_prev według następujących wzorów:

$$srtt\_prev = srtt + 2(G)$$
$$rttvar\_prev = rttvar$$

Zmienne te mają przypisywane wartości przy każdym zgłoszeniu przeterminowania przez licznik czasu, ale są wykorzystywane tylko wtedy, gdy zostanie ustalone, że przeterminowanie było fałszywe. W tym przypadku pomagają utworzyć podstawę do ustawienia nowej wartości czasu RTO. Wartość G we wzorze przedstawia ziarnistość (*granularity*) zegara TCP. Zmiennej srtt_prev jest przypisywana wartość srtt plus podwojona wartość ziarnistości zegara na podstawie następującego ciągu rozumowania: fałszywe przeterminowanie mogło zostać wywołane, bo wartość srtt była odrobinę za mała. Jeśli byłaby choć trochę większa, nie doszłoby do żadnego przeterminowania. Dodanie wyrażenia 2(G) do wartości zmiennej srtt umożliwia poradzenie sobie z tą sytuacją przez zapamiętanie nieco zwiększonej wartości w zmiennej srtt_prev, która zostanie później użyta do ustalenia czasu RTO.

Po zapamiętaniu wartości srtt_prev i rttvar_prev wywoływany jest jeden z algorytmów wykrywania. W wyniku wykonania algorytmu otrzymujemy wartość przypisaną do specjalnej zmiennej o nazwie SpuriousRecovery. Jeśli algorytm wykryje fałszywe przeterminowanie, zmienna SpuriousRecovery otrzymuje wartość SPUR_TO. Jeśli zostaje wykryte późne fałszywe przeterminowanie, zmiennej SpuriousRecovery zostaje przypisana wartość LATE_SPUR_TO. W innym przypadku przeterminowanie nie jest fałszywe i kontynuowana jest zwykła obsługa przeterminowania przez protokół TCP.

Jeśli zmienna SpuriousRecovery ma wartość SPUR_TO, protokół TCP może podjąć działanie przed zakończeniem procesu odtwarzania. Wykonuje to przez przypisanie parametrowi oznaczającemu numer sekwencyjny następnego segmentu, który ma być wysłany (o nazwie SND.NXT), wartości numeru sekwencyjnego pierwszego nowego, niewysłanego segmentu (o nazwie SND.MAX). Pozwala to uniknąć omawianego wcześniej niepożądanego zachowania typu *go-back-N* po początkowej retransmisji. Gdy algorytm wykrywania znajdzie późne fałszywe przeterminowanie, potwierdzenie ACK dla początkowej retransmisji już zostało odebrane, więc wartość SND.NXT nie jest zmieniana. Jednak w każdym przypadku ma miejsce resetowanie stanu kontroli przeciążenia (patrz rozdział 16.). Ponadto, kiedy zostanie już odebrane akceptowalne potwierdzenie ACK dla segmentu przesłanego po zgłoszeniu przeterminowania przez licznik czasu retransmisji, wartości zmiennych srtt, rttvar i czasu RTO mogą zostać uaktualnione w następujący sposób:

$$srtt \leftarrow \max(srtt\_prev, m)$$
$$rttvar \leftarrow \max(rttvar\_prev, m/2)$$
$$\text{RTO} = srtt + \max(G, 4(rttvar))$$

Zmienna m jest wartością próbki czasu RTT połączenia wyznaczoną na podstawie momentu przyjścia pierwszego akceptowalnego potwierdzenia ACK dla danych wysłanych po przeterminowaniu. Motywacja tych modyfikacji polega na tym, że rzeczywisty czas RTT mógł zmienić się tak znacznie, iż historia czasów RTT uwzględniona w bieżących estymatorach nie stanowi już uzasadnionej podstawy do ustalenia wartości RTO. Jeśli rzeczywisty czas RTT ścieżki zwiększył się gwałtownie (np. z powodu zmiany stacji bazowej w połączeniu bezprzewodowym), bieżące wartości srtt i rttvar są prawdopodobnie zbyt małe i zmienne te powinny być zainicjowane na nowo. Z drugiej strony, wzrost czasu RTT ścieżki mógł być tylko chwilowy, a w takim przypadku ponowna inicjacja zmiennych srtt i rttvar mogłaby nie być takim dobrym pomysłem, ponieważ prawdopodobnie ich wartości byłyby w przybliżeniu poprawne.

Powyższe równania próbują utrzymać równowagę między tymi dwoma sytuacjami przez ponowne zainicjowanie średnich ruchomych srtt i rttvar tylko wtedy, gdy nowe próbki czasu RTT są większe. Ten sposób postępowania w rzeczywistości odrzuca wcześniejszą historię czasów RTT (i wariancji RTT). Wartości zmiennych srtt i rttvar mogą w wyniku działania algorytmu odpowiedzi tylko ulec zwiększeniu. Jeżeli nie pojawia się wzrost czasu RTT, bieżące estymatory pozostają niezmienione, zasadniczo ignorując fakt wystąpienia przeterminowania. Zmiana czasu RTO jest wykonana w każdym przypadku w konwencjonalny sposób, a nowy licznik czasu retransmisji zostaje ustawiony na tę nową wartość limitu czasu.

14.8. Zmiana kolejności i powielanie pakietów

Większość dotąd analizowanych kwestii wiąże się ze sposobem obsługi utraty pakietów przez protokół TCP. Jest to problem stosunkowo często występujący i dotychczas włożono bardzo wiele pracy, by protokół TCP stał się odporny na przypadki gubienia pakietów. Jak zaczęliśmy dostrzegać w poprzednim podrozdziale, inne anomalie w dostarczaniu pakietów, takie jak powielanie i zmiana kolejności, również mogą niekorzystnie wpłynąć na funkcjonowanie protokołu TCP. W obu tych przypadkach chcielibyśmy, aby protokół TCP potrafił odróżnić pakiety, których kolejność została zmieniona lub które zostały zdublowane, od tych pakietów, które zostały utracone. Już wkrótce przekonamy się, że czasem nie jest to takie proste.

14.8.1. Zmiana kolejności pakietów

Zmiana kolejności pakietów może wystąpić w sieci IP, ponieważ protokół IP nie daje żadnej gwarancji, że względne uporządkowanie pakietów zostanie zachowane podczas ich dostarczania. Bywa to korzystne (przynajmniej dla protokołu IP), ponieważ protokół IP może wybrać inną ścieżkę dla ruchu sieciowego (np. taką, która jest szybsza) i nie martwić się o wynikające z tego konsekwencje polegające na tym, że dane świeżo wprowadzone do sieci wyprzedzą starsze dane, co sprawi, że porządek przychodzenia pakietów do odbiorcy nie będzie odpowiadał porządkowi ich transmisji po stronie nadawcy. Są też inne przyczyny stwarzające możliwość zmiany kolejności pakietów. Przykładowo pewne wysoko wydajne routery wykorzystują wiele równoległych ścieżek danych wewnątrz urządzenia ([patrz BPS99]), a różnice opóźnień przetwarzania między pakietami mogą prowadzić do kolejności wysyłania nieodpowiadającej kolejności przychodzenia pakietów.

Zmiana kolejności może mieć miejsce w ścieżce docelowej lub w ścieżce zwrotnej połączenia TCP (lub, w niektórych przypadkach, w obu). Zmiana kolejności segmentów danych ma nieco inny wpływ na działanie TCP niż zmiana kolejności pakietów ACK. Przypomnijmy sobie, że z powodu asymetrycznego wyznaczania tras mamy często do czynienia z sytuacją, gdzie potwierdzenia ACK wędrują przez inne łącza sieciowe (i przez inne routery) niż pakiety z danymi na ścieżce docelowej.

Zmiana kolejności w ruchu sieciowym może wpłynąć na działanie protokołu TCP na kilka sposobów. Jeśli zmiana porządku dotyczy kierunku odwrotnego (tzn. kierunku przepływu potwierdzeń ACK), sprawia, że nadawczy protokół TCP najpierw odbiera

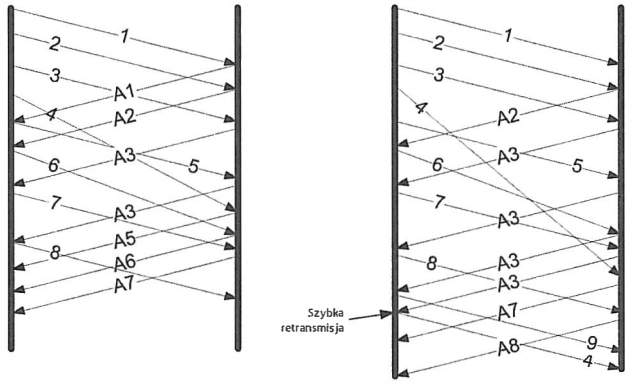
pewne potwierdzenia ACK, które przesuwają okno danych znacząco do przodu, a następnie jakieś ewidentnie stare, zbędne już potwierdzenia, które są odrzucane. Może to prowadzić do niepożądanej **impulsowości** (*burstiness*; jest to chwilowe, nagłe zwiększenie szybkości transmisji) w schemacie nadawania protokołu TCP, a także do kłopotów z wykorzystaniem dostępnej przepustowości sieci wynikających ze sposobu funkcjonowania kontroli przeciążenia protokołu TCP (patrz rozdział 16.).

Jeżeli zmiana kolejności pakietów ma miejsce w kierunku docelowym, protokół TCP może mieć kłopot z odróżnieniem tego stanu od utraty pakietów. Zarówno utrata pakietów, jak i zmiana ich kolejności powodują, że odbiorca otrzymuje poza kolejnością pakiety, które tworzą luki między następnym oczekiwanym pakietem a pozostałymi dotychczas odebranymi. Jeśli zmiana porządku jest umiarkowana (np. dwa sąsiednie pakiety zamieniają się kolejnością), sytuacja może być opanowana dość szybko. Kiedy zmiany kolejności są poważniejsze, protokół TCP może zostać oszukany, bo fałszywie przyjmie, że dane zostały utracone. To może skutkować zbędnymi retransmisjami, głównie w wyniku działania algorytmu szybkiej retransmisji.

Wcześniej pisaliśmy, że algorytm szybkiej retransmisji polega na obserwowaniu otrzymywanych od protokołu TCP odbiorcy zduplikowanych potwierdzeń w celu ewentualnego wywnioskowania utraty pakietu i zainicjowania retransmisji bez potrzeby oczekiwania na sygnalizację przeterminowania przez licznik czasu retransmisji. Ponieważ z założenia protokół TCP odbiorcy ma natychmiast potwierdzać przez ACK wszystkie dane otrzymane poza kolejnością, aby pomóc w doprowadzeniu do szybkiej retransmisji, która powinna być uruchomiona w następstwie utraty pakietu, każdy pakiet, którego kolejność została zmieniona przy przejściu przez sieć powoduje wygenerowanie przez odbiorcę zduplikowanego potwierdzenia ACK. Jeśli szybka retransmisja byłaby wywoływana za każdym razem, gdy nadawca odbierze zduplikowane potwierdzenie ACK, dochodziłoby do wielkiej liczby niepotrzebnych retransmisji na tych ścieżkach w sieci, gdzie niewielkie zmiany kolejności pakietów zdarzają się często. Aby zaradzić tej sytuacji, szybka retransmisja jest uruchamiana dopiero wtedy, kiedy liczba zduplikowanych potwierdzeń osiągnie założony próg (*dupthresh*).

Efekt zastosowania progu został pokazany na rysunku 14.13. Po lewej stronie rysunku pokazujemy, jak TCP zachowuje się w przypadku lekkiej zmiany porządku pakietów, gdy wartość progu dupthresh wynosi 3. W tym przypadku pojedyncze zduplikowane potwierdzenie ACK nie ma wpływu na działanie protokołu TCP. Jest ono praktycznie zignorowane, a TCP opanowuje sytuację wynikającą ze zmiany kolejności. Po prawej stronie rysunku pokazujemy, co się dzieje, kiedy zmiana kolejności jednego z pakietów jest poważniejsza. Ponieważ pakiet przesunął się w kolejności o trzy pozycje, wygenerowane zostają trzy zduplikowane potwierdzenia ACK. To uruchamia procedurę szybkiej retransmisji protokołu TCP nadawcy i dostarczenie odbiorcy zduplikowanego segmentu danych.

Problem odróżnienia utraty danych od zmiany kolejności ich dostarczania nie jest trywialny. Radzenie sobie z nim jest związane z próbą określenia momentu, w którym nadawca uznaje, że czekał już wystarczająco długo i może podjąć próbę wypełnienia oczywistych luk w danych odbiorcy. Na szczęście, poważna zmiana kolejności nie występuje w Internecie zbyt często (patrz [J03]), więc nadanie progowi dupthresh stosunkowo małej wartości (takiej jak domyślne 3) wystarcza w większości przypadków. Należy



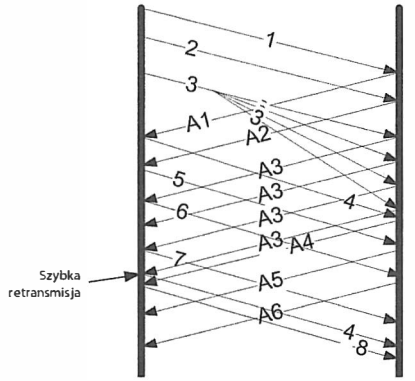
Rysunek 14.13. *Umiarkowana zmiana kolejności (po lewej) jest opanowana przez zignorowanie małej liczby zduplikowanych potwierdzeń ACK. Kiedy zmiana kolejności jest poważniejsza (po prawej), tak jak w tym przypadku, gdy pakiet 4. jest opóźniony o trzy miejsca, może zostać wywołana zbędna szybka retransmisja*

dodać, że istnieje szereg projektów naukowych, które modyfikują protokół TCP w kierunku obsługi poważniejszych zmian kolejności (patrz [LLY07]). Niektóre z tych modyfikacji dynamicznie regulują wartość progu `dupthresh`, tak jak implementacja protokołu TCP systemu Linux.

14.8.2. Powielanie pakietów

Chociaż jest to rzadkie, protokół IP może dostarczyć pojedynczy pakiet więcej niż jeden raz. Może się to zdarzyć np. wtedy, gdy protokół warstwy łącza danych wykonuje retransmisje i tworzy dwa egzemplarze tego samego pakietu. Kiedy powstają duplikaty, protokół TCP może zostać zmylony na jeden ze sposobów, które już omawialiśmy. Rozważmy przypadek pokazany na rysunku 14.14, w którym pakiet numer 3 został powielony trzy razy.

Rysunek 14.14. *Powielenie pakietu w sieci spowodowało zbędną szybką retransmisję na skutek wystąpienia zduplikowanych potwierdzeń ACK*



Jak możemy zobaczyć, skutek powielenia pakietu 3. polega na wygenerowaniu serii zduplikowanych potwierdzeń ACK przez odbiorcę. Wystarczy to do uruchomienia zbędnej szybkiej retransmisji, ponieważ nadawca nieobsługujący opcji SACK może błędnie sądzić, że pakiety 5. i 6. dotarły wcześniej do odbiorcy. Przy użyciu opcji SACK (a szczególnie informacji DSACK) sytuacja jest łatwiejsza do zdiagnozowania dla nadawcy. Po zastosowaniu metody DSACK każde ze zduplikowanych potwierdzeń dla segmentu 3. (oznaczonych na rysunku etykietą A3) zawiera informację DSACK sygnalizującą, że segment 3. został już wcześniej odebrany. Co więcej, żadne z nich nie zawiera informacji wskazującej na jakiekolwiek dane poza kolejnością, co oznacza, że przychodzące do odbiorcy pakiety (lub ich potwierdzenia ACK) musiały być duplikatami. W takich przypadkach protokół TCP często może powstrzymać zbędne retransmisje.

14.9. Mierniki punktu docelowego

Jak już zobaczyliśmy, protokół TCP „uczy się" właściwości ścieżki sieciowej między nadawcą i odbiorcą w miarę upływu czasu. Wiedza ta jest przechowywana w zmiennych stanu u nadawcy, takich jak srtt i rttvar. Niektóre implementacje protokołu TCP prowadzą również bieżące szacowanie ilości zmian kolejności pakietów, które ostatnio wystąpiły w trakcie transmisji. Dawniej wiedza ta była tracona z chwilą zamknięcia połączenia. Oznaczało to, że po otwarciu nowego połączenia TCP z tym samym odbiorcą musiało rozpocząć się ustalanie wartości zmiennych stanu od zera.

Nowsze implementacje protokołu TCP utrzymują wiele spośród opisanych przez nas w tym rozdziale mierników w tablicy routingu, w tablicy przesyłania lub w innej globalnej strukturze danych systemowych, która istnieje nawet po zamknięciu połączeń TCP. Kiedy tworzone jest nowe połączenie, protokół TCP sprawdza zawartość tej struktury danych, żeby zobaczyć, czy nie ma w niej uprzednio istniejącej informacji dotyczącej ścieżki do hosta docelowego, z którym będzie się teraz komunikować. Jeśli taka informacja istnieje, zmienne srtt, rttvar i pozostałe mogą otrzymać wartości początkowe oparte na wcześniejszym, stosunkowo świeżym doświadczeniu. Kiedy połączenie TCP jest zamykane, jest okazja do aktualizacji odpowiednich statystyk. Może być to wykonane albo przez zastąpienie istniejących danych statystycznych, albo przez ich uaktualnienie w jakiś inny sposób. W systemie Linux 2.6 nowe wartości są wyznaczone jako maksimum z wartości już istniejącej i zmierzonej w trakcie ostatniego połączenia TCP. Wartości te mogą być sprawdzone przy użyciu programu ip pochodzącego z zestawu narzędzi iproute2 (patrz [IPR2]):

```
Linux% ip route show cache 132.239.50.184
132.239.50.184 from 10.0.0.9 tos 0x10 via 10.0.0.1 dev eth0
    cache mtu 1500 rtt 29ms rttvar 29ms cwnd 2 advmss 1460 hoplimit 64
```

Polecenie to pokazuje zbuforowaną informację o poprzednich połączeniach z konkretną wartością DSCP (tj. 16, wskazującą klasę CS2, ale reprezentowaną zgodnie ze starszą terminologią przez bajt ToS z wartością 0x10) między lokalnym systemem a hostem 132.239.50.184 przy użyciu adresu IPv4 następnego przeskoku 10.0.0.1, z dostępem przez urządzenie sieciowe eth0. Widzimy informacje dotyczące rozmiaru pakietów (MTU ścieżki uzyskane przez zastosowanie techniki PMTUD, wartość MSS ogłaszaną przez stronę zdalną), maksymalną liczbę przeskoków do wykorzystania (dla protokołu IPv6; nie ma tu zastosowania), wartości estymatorów srtt i rttvar, razem z informacją związaną z kontrolą przeciążenia, taką jak cwnd, którą omawiamy w rozdziale 16.

14.10. Przepakietowanie

Kiedy protokół TCP stwierdza przeterminowanie i wykonuje retransmisję, nie musi ponownie transmitować identycznego segmentu. Zamiast tego protokół TCP może wykonać **przepakietowanie** (*repacketization*) i wysłać większy segment, co może zwiększyć wydajność. (Naturalnie, rozmiar tego większego segmentu nie może przekroczyć wartości MSS ogłoszonej przez odbiorcę i nie powinien przekroczyć wartości MTU ścieżki). Jest to dozwolone w protokole, ponieważ TCP identyfikuje wysyłane i potwierdzane dane za pomocą numerów bajtów, a nie numerów segmentu (lub pakietu).

Zdolność protokołu TCP do retransmitowania segmentu z rozmiarem zmienionym w stosunku do oryginalnego segmentu dostarcza jeszcze jednego sposobu rozwiązywania problemu niejednoznaczności retransmisji. Stała się ona podstawą pomysłu nazwanego STODER (patrz [TZZ05]), który wykorzystuje przepakietowanie do wykrywania fałszywych przeterminowań.

Możemy łatwo zobaczyć przepakietowanie w działaniu. Używamy naszego programu sock jako serwera i łączymy się z nim za pomocą programu Telnet. Najpierw wprowadzamy wiersz `hello there`. To tworzy segment zawierający 13 bajtów danych, łącznie ze znakami powrotu karetki i nowego wiersza wygenerowanymi po naciśnięciu klawisza *Enter*. Następnie rozłączamy połączenie z siecią i wprowadzamy tekst `line number 2` (14 bajtów łącznie ze znakiem nowego wiersza). Potem czekamy ok. 45 s i wpisujemy tekst `and 3`, po czym kończymy połączenie:

```
Linux% telnet 169.229.62.97 6666
hello there                    (pierwszy wiersz zostaje wysłany z sukcesem)
                               (następnie odłączamy kabel Ethernetu)
line number 2                  (ten wiersz jest retransmitowany)
and 3                          (ponownie podłączamy Ethernet)
^] telnet> quit
```

Wyniki możemy zobaczyć przy użyciu programu tcpdump:

```
1 19:51:47.674418 IP 10.0.0.7.1029 > 169.229.62.97.6666:
     P 1:14(13) ack 1 win 5840
     <nop,nop,timestamp 2343578137 596377728>

2 19:51:47.788992 IP 169.229.62.97.6666 > 10.0.0.7.1029:
     . ack 14 win 58254 <nop,nop,timestamp 596378252 2343578137>

3 19:52:35.130837 IP 10.0.0.7.1029 > 169.229.62.97.6666:
     FP 29:36(7) ack 1 win 5840
     <nop,nop,timestamp 2343602439 596378252>

4 19:52:35.146358 IP 169.229.62.97.6666 > 10.0.0.7.1029:
     . ack 14 win 58254
     <nop,nop,timestamp 596382987 2343578137,nop,nop,
     sack sack 1 {29:36}>

5 19:52:39.414253 IP 10.0.0.7.1029 > 169.229.62.97.6666:
     FP 14:36(22) ack 1 win 5840
     <nop,nop,timestamp 2343604633 596382987>
```

hello there\r\n

and 3\r\n

2\r\n

```
6 19:52:39.429228 IP 169.229.62.97.6666 > 10.0.0.7.1029:
      ack 37 win 58248 <nop,nop,timestamp 596383416 2343604633>

7 19:52:39.429696 IP 169.229.62.97.6666 > 10.0.0.7.1029:
     F 1:1(0) ack 37 win 58254
     <nop,nop,timestamp 596383416 2343604633>

8 19:52:39.430119 IP 10.0.0.7.1029 > 169.229.62.97.6666:
      ack 2 win 5840 <nop,nop,timestamp 2343604641 596383416>
```

Z powyższego śladu transmisji została usunięta początkowa wymiana segmentów SYN. Pierwsze dwa segmenty zawierają odpowiednio ciąg znaków `hello there` i jego potwierdzenie. Następny pakiet śladu nie zachowuje kolejności: rozpoczyna się od numeru sekwencyjnego 29 i zawiera ciąg znaków `and 3` (7 bajtów). Odpowiadające mu potwierdzenie ACK zawiera numer ACK równy 14, ale również blok SACK ze względnymi numerami sekwencyjnymi {29,36}. Środkowa sekwencja znaków została zgubiona. Protokół TCP retransmituje ją, ale używa większego pakietu, zawierającego numery sekwencyjne 14:36. W ten sposób możemy zobaczyć, jak retransmisja dla numeru sekwencyjnego 14 spowodowała przepakietowanie prowadzące do utworzenia większego pakietu o rozmiarze 22 bajtów. Co interesujące, pakiet ten zachodzi na dane wskazane w bloku SACK oraz ma włączony bit FIN, co wskazuje, że zawiera ostatnie dane tego połączenia.

14.11. Ataki związane z mechanizmem retransmisji protokołu TCP

Istnieje klasa ataków typu DoS zwana atakami ***low-rate*** DoS (ataki DoS spowalniające wysyłanie danych; patrz [KK03]). Przeprowadzając taki atak, napastnik wysyła do bramy lub hosta duże ilości danych w krótkich okresach czasu, powodując w zaatakowanym systemie przekroczenie limitu czasu oczekiwania na retransmisję. Posiadając możliwość przewidzenia, kiedy protokół TCP będący ofiarą ataku podejmie próbę retransmisji, napastnik generuje kolejną porcję zwiększonego ruchu sieciowego przy każdej próbie retransmisji. W konsekwencji zaatakowany protokół TCP stwierdza przeciążenie sieci, wyhamowuje swoją szybkość wysyłania danych do wartości bliskiej zeru, kontynuuje zwiększanie czasu RTO przez zastosowanie rosnącego wykładniczo współczynnika odczekiwania zgodnie z algorytmem Karna i w efekcie uzyskuje bardzo niską przepustowość. Proponowanym mechanizmem postępowania z tego typu atakiem jest dodanie randomizacji do sposobu wyznaczania czasu RTO i utrudnienie napastnikowi odgadnięcia dokładnych czasów, w których będą miały miejsce retransmisje.

Zbliżona, ale odrębna forma ataku DoS wiąże się ze spowolnieniem przepływu segmentów wysyłanych przez zaatakowany protokół TCP, tak żeby oszacowanie czasu RTT było za wysokie. Wówczas zaatakowany protokół TCP jest mniej agresywny w retransmitowaniu swoich własnych pakietów, kiedy są gubione. Odwrotny typ ataku jest również możliwy: napastnik podrabia potwierdzenia ACK, wysyłając je, kiedy dane zostały wysłane, ale w rzeczywistości nie dotarły jeszcze do odbiorcy. W tym przypadku napastnik sprawia, że zaatakowany protokół TCP sądzi, że czas RTT połączenia jest znacząco mniejszy od jego prawdziwej wartości i staje się nadmiernie agresywny, generując liczne niepotrzebne retransmisje.

14.12. Podsumowanie

W rozdziale tym szczegółowo omówiliśmy strategię przeterminowania i retransmisji protokołu TCP. Nasz pierwszy przykład był ilustracją przypadku, w którym po prostu odłączyliśmy sieć, kiedy protokół TCP miał pakiet do przesłania. W rezultacie licznik czasu retransmisji zainicjował retransmisję na podstawie przeterminowania. Każda kolejna retransmisja następowała po dwukrotnie większym odstępie czasu niż poprzednia retransmisja w wyniku zastosowania drugiej części algorytmu Karna zawierającego procedurę binarnego odczekiwania wykładniczego.

Protokół TCP mierzy czas RTT, a następnie używa tych pomiarów do bieżącej aktualizacji wygładzonego estymatora czasu RTT i wygładzonego estymatora średniego odchylenia. Te dwa estymatory zostają potem użyte do obliczenia następnej wartości czasu oczekiwania na retransmisję. Bez opcji *Znaczniki czasu* protokół TCP mierzy tylko jeden czas RTT na okno danych. Algorytm Karna usuwa problem niejednoznaczności retransmisji, bo nie pozwala na użycie pomiarów czasu RTT wykonanych dla segmentów, które zostały utracone. Obecnie większość implementacji protokołu TCP korzysta z opcji *Znaczniki czasu*, która umożliwia indywidualny pomiar czasu dla każdego segmentu. Opcja *Znaczniki czasu* działa prawidłowo nawet w obliczu zmiany kolejności i powielania pakietów.

Przyjrzeliśmy się algorytmowi szybkiej retransmisji, która może być uruchomiona bez wymagania sygnalizacji przeterminowania przez licznik czasu. Jest to najbardziej efektywna metoda (i najczęściej wykorzystywana) umożliwiająca protokołowi TCP wypełnianie luk w danych odbiorcy spowodowanych przez brakujące pakiety. Procedura szybkiej retransmisji może być ulepszona przez użycie selektywnych potwierdzeń ACK. Opcje SACK przenoszą dodatkową informacje w potwierdzeniach ACK i pozwalają nadawcy zdolnemu do obsługi tych opcji na naprawienie więcej niż jednej luki w pojedynczym czasie RTT. W pewnych okolicznościach takie działanie może prowadzić do poprawienia wydajności.

Jeżeli oszacowanie RTT wypada poniżej rzeczywistego czasu RTT połączenia, może mieć miejsce zbędna retransmisja. Jeśli w takich przypadkach protokół TCP poczekałby nieco dłużej, (niepotrzebna) retransmisja nie doszłaby do skutku. Opracowano szereg algorytmów do wykrywania przypadków wystąpienia fałszywego przeterminowania. Metoda DSACK wymaga przyjścia do odbiorcy zdublowanego segmentu. Algorytm wykrywania *Eifel* polega na znacznikach czasu protokołu TCP, ale potrafi reagować szybciej niż procedura wykorzystująca informacje DSACK, ponieważ wykrywa fałszywe przeterminowania na podstawie potwierdzeń ACK zwracanych dla segmentów, które zostały wysłane przed wystąpieniem przeterminowania. F-RTO jest jeszcze jednym algorytmem, który zachowuje się podobnie do algorytmu *Eifel*, ale nie wymaga użycia znaczników czasu. Algorytm ten dodatkowo nakazuje nadawcy, zmieniając jego normalne zachowanie, wysłanie nowych danych po przeterminowaniu, które zostało uznane za fałszywe. Wszystkie te algorytmy wykrywania mogą być połączone z algorytmem odpowiedzi. Głównym dotychczas opisanym algorytmem jest algorytm odpowiedzi *Eifel*, który może zresetować oszacowania czasu RTT i wariancji RTT, jeśli wystąpił znaczny wzrost opóźnienia (w przeciwnym przypadku algorytm „wycofuje" wszystkie zmiany dotyczące czasu RTO, które ewentualnie wykonałby protokół TCP).

Przyjrzeliśmy się również, jak stan protokołu TCP może być przechowywany między połączeniami, w jaki sposób protokół TCP może „przepakietować" swoje dane, a także opisaliśmy pewne ataki, które mogą zostać przeprowadzone w celu oszukania protokołu TCP, aby zaczął działać w nieodpowiedni sposób, np. zachowując się zbyt pasywnie lub zbyt agresywnie. Więcej o konsekwencjach tych ataków dowiemy się w rozdziale 16., w którym badamy procedury kontroli przeciążenia protokołu TCP.

14.13. Bibliografia

[G04] S. Gorard, *Revisiting a 90-Year-Old Debate: The Advantages of the Mean Deviation*, Department of Educational Studies, University of York, referat zaprezentowany na dorocznej konferencji British Educational Research Association, University of Manchester, 16 – 18 wrzesień, 2004.

[BPS99] J. Bennett, C. Partridge, N. Shectman, *Packet Re-ordering Is Not Pathological Network Behavior*, „IEEE/ACM Transactions on Networking", grudzień 1999, nr 7(6).

[F68] W. Feller, *An Introduction to Probability Theory and Its Applications, Volume 1*, Wiley, 1968.

[ID1323b] V. Jacobson, B. Braden, D. Borman, *TCP Extensions for High Performance*, (stracił ważność), Internet draft-jacobson-tsvwg-1323bis-01, prace w toku, marzec 2009. Dostępny pod adresem: *http://tools.ietf.org/html/draft-ietf-tcpm-1323bis-01*

[IPR2] *http://www.linuxfoundation.org/collaborate/workgroups/networking/iproute2*

[J88] V. Jacobson, *Congestion Avoidance and Control*, „Proc. ACM SIGCOMM", sierpień 1988.

[J90] V. Jacobson, *Berkeley TCP Evolution from 4.3-Tahoe to 4.3 Reno*, „Proc. 18th IETF", wrzesień 1990.

[J03] S. Jaiswal i in., *Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone*, „Proc. IEEE INFOCOM", kwiecień 2003.

[KK03] A. Kuzmanovic, E. Knightly, *Low-Rate TCP-Targeted Denial of Service Attacks*, „Proc. ACM SIGCOMM", sierpień 2003.

[KP87] P. Karn, C. Partridge, *Improving Round-Trip Time Estimates in Reliable Transport Protocols*, „Proc. ACM SIGCOMM", sierpień 1987.

[LLY07] K. Leung, V. Li, D. Yang, *An Overview of Packet Reordering in Transmission Control Protocol (TCP): Problems, Solutions and Challenges*, „IEEE Trans. Parallel and Distributed Systems", kwiecień 2007, nr 18(4).

[LS00] R. Ludwig, K. Sklower, *The Eifel Retransmission Timer*, „ACM Computer Communication Review", lipiec 2000, nr 30(3).

[RFC0793] J. Postel, *Transmission Control Protocol*, Internet RFC 0793/STD0007, wrzesień 1981.

[RFC1122] R. Braden, red., *Requirements for Internet Hosts*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1323] V. Jacobson, R. Braden, D. Borman, *TCP Extensions for High Performance*, Internet RFC 1323, maj 1992.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgment Options*, Internet RFC 2018, październik 1996.

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, *An Extension to the Selective Acknowledgement (SACK) Option for TCP*, Internet RFC 2883, lipiec 2000.

[RFC3517] E. Blanton, M. Allman, K. Fall, L. Wang, *A Conservative Selective Acknowledgment (SACK)-Based Loss Recovery Algorithm for TCP*, Internet RFC 3517, kwiecień 2003.

[RFC3522] R. Ludwig, M. Meyer, *The Eifel Detection Algorithm for TCP*, Internet RFC 3522 (experimental), kwiecień 2003.

[RFC3708] E. Blanton, M. Allman, *Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions*, Internet RFC 3708 (experimental), luty 2004.

[RFC3782] S. Floyd, T. Henderson, A. Gurtov, *The NewReno Modification to TCP's Fast Recovery Algorithm*, Internet RFC 3782, kwiecień 2004.

[RFC4015] R. Ludwig, A Gurtov, *The Eifel Response Algorithm for TCP*, Internet RFC 4015, luty 2005.

[RFC5681] M. Allman, V. Paxson, E. Blanton, *TCP Congestion Control*, Internet RFC 5681, wrzesień 2009.

[RFC5682] P. Sarolahti, M. Kojo, K. Yamamoto, M. Hata, *Forward RTORecovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP*, Internet RFC 5682, wrzesień 2009.

[RFC6298] V. Paxson, M. Allman, J. Chu, *Computing TCP's Retransmission Timer*, Internet RFC 6298, czerwiec 2011.

[RKS07] S. Rewaskar, J. Kaur, F.D. Smith, „Performance Study of Loss Detection/ Recovery in Real-World TCP Implementations", *Proc. IEEE ICNP*, październik 2007.

[SK02] P. Sarolahti, A. Kuznetsov, „Congestion Control in Linux TCP", *Proc. Usenix Freenix Track*, czerwiec 2002.

[TZZ05] K. Tan, Q. Zhang, „STODER: A Robust and Efficient Algorithm for Handling Spurious Timeouts in TCP", *Proc. IEEE Globecomm*, grudzień 2005.

[V09] V. Vasudevan i in., „Safe and Fine-Grained TCP Retransmissions for Datacenter Communication", *Proc. ACM SIGCOMM*, sierpień 2009.

[WINREG] *TCP/IP Registry Values for Microsoft Windows Vista and Windows Server 2008*, styczeń 2008. Patrz *http://www.microsoft.com/download/en/details.aspx?id=9152*

Rozdział 15. Przepływ danych i zarządzanie oknem w protokole TCP

15.1. Wprowadzenie

W rozdziale 13. zajmowaliśmy się ustanawianiem i kończeniem połączeń TCP, a w rozdziale 14. badaliśmy, jak protokół TCP zapewnia niezawodne dostarczanie, stosując retransmisję danych, które zostały utracone. Teraz zbadamy dynamikę transferów danych w protokole TCP; skupimy się najpierw na połączeniach interaktywnych, a następnie przedstawimy powiązane ze sobą procedury sterowania przepływem i zarządzania oknem używane w połączeniu z kontrolą przeciążenia (patrz rozdział 16.) do obsługi transferów danych masowych.

„Interaktywnym" połączeniem TCP nazywamy takie połączenie, w którym dane wprowadzane przez użytkownika, takie jak naciśnięcia klawiszy, krótkie komunikaty, ruchy joysticka lub myszy, muszą być dostarczane od klienta do serwera. Jeśli do przenoszenia takich danych wejściowych użytkownika używane są małe segmenty, protokół wprowadza większy narzut z powodu mniejszej ilości bajtów ładunku użytecznego przypadającej na pojedynczy przesyłany pakiet. Z drugiej strony, wypełnianie pakietów większą ilością danych zwykle wiąże się z opóźnieniem w ich dostarczaniu, co może mieć negatywny wpływ na aplikacje wrażliwe na opóźnienie, takie jak gry online i narzędzia współpracy grupowej. Zbadamy techniki, z wykorzystaniem których aplikacja może znaleźć kompromis między tymi dwoma uwarunkowaniami.

Po omówieniu komunikacji interaktywnych przeanalizujemy metody, których w protokole TCP używa się do realizacji kontroli przepływu, a które polegają na dynamicznym dostosowaniu rozmiaru okna w taki sposób, by zapewnić, że nadawca nie zaleje odbiorcy danymi. Problem ten przede wszystkim dotyczy transferu danych masowych (tzn. komunikacji nieinteraktywnych), ale może również dotknąć aplikacji interaktywnych. W rozdziale 16. zbadamy, jak pojęcie kontroli przepływu może zostać rozszerzone i obejmować nie tylko ochronę odbiorcy, ale również sieci między nadawcą i odbiorcą.

15.2. Komunikacja interaktywna

Ilość ruchu sieciowego przenoszonego w konkretnej części Internetu, w ciągu określonego czasu, jest zwykle mierzona w bajtach lub w pakietach. Liczby te charakteryzują się znaczną zmiennością. Przykładowo ruch w sieciach lokalnych różni się od ruchu w sieciach rozległych, a ruch między poszczególnymi miejscami przejawia tendencję do zróżnicowania. Badania ruchu generowanego przez protokół TCP (patrz [P05], [F03]) zwykle stwierdzają, że 90% lub więcej segmentów TCP zawiera **dane masowe** (*bulk data*; np. strony WWW, dystrybucja plików, poczta elektroniczna, kopie bezpieczeństwa), a pozostała ich część zawiera **dane interaktywne** (*interactive data*; np. zdalne logowania, gry sieciowe). Segmenty zawierające dane masowe są na ogół stosunkowo duże (1500 bajtów lub więcej), gdy tymczasem segmenty z danymi interaktywnymi są ogólnie dużo mniejsze (dziesiątki bajtów danych użytkowych).

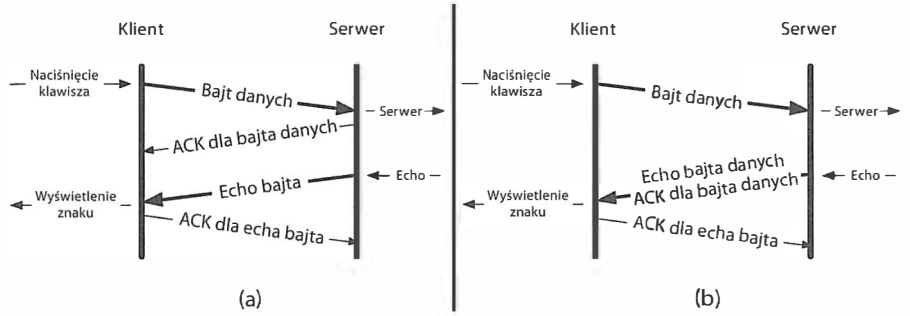
TCP obsługuje oba typy danych przy użyciu tego samego protokołu i formatu pakietu, ale dla każdego przypadku mają zastosowanie inne algorytmy. W tym podrozdziale zobaczymy, jak protokół TCP wykonuje transfer danych interaktywnych; użyjemy jako przykładu aplikacji `ssh` (czyli bezpiecznej powłoki). Bezpieczna powłoka, opisana w dokumencie [RFC4251], jest protokołem zdalnego logowania, który zapewnia wysoki poziom bezpieczeństwa (poufność i uwierzytelnianie oparte na kryptografii). W większości przypadków aplikacja ta zastąpiła używane w systemie UNIX programy `rlogin` i Telnet, które dostarczają usługi zdalnego logowania bez silnych zabezpieczeń.

Badając działanie programu `ssh`, zobaczymy, jak funkcjonują opóźnione potwierdzenia i w jaki sposób **algorytm Nagle'a** redukuje liczbę małych pakietów przesyłanych w sieciach rozległych. Takie same algorytmy mają zastosowanie w innych aplikacjach obsługujących możliwość zdalnego logowania, takich jak Telnet, `rlogin` i usługi terminalowe systemu Windows.

Spójrzmy na przepływ danych wynikający z wprowadzenia interaktywnego polecenia w ramach połączenia `ssh`. Klient przechwytuje wszystko, co wpisuje użytkownik, i przesyła do serwera do interpretacji, a serwer wysyła odpowiedzi z powrotem do klienta. Klient szyfruje wysyłane dane, co oznacza, że znaki wprowadzone przez użytkownika są specjalnie kodowane przed ich przesłaniem przez sieć (patrz rozdział 18.). Kodowanie utrudnia podsłuchującemu identyfikację znaków wprowadzonych przez użytkownika. Klient obsługuje kilka algorytmów szyfrowania i różne metody uwierzytelniania, a także kilka innych zaawansowanych mechanizmów, takich jak tunelowanie innych protokołów (patrz rozdział 3. i [RFC4254]).

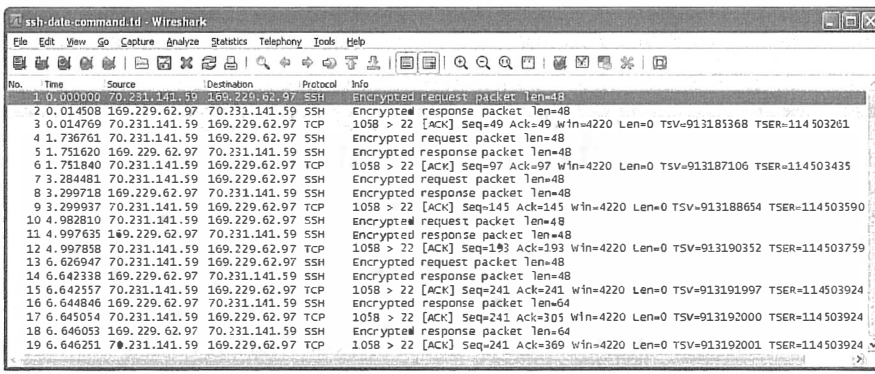
Wiele osób nieposiadających doświadczenia w dziedzinie protokołów TCP/IP jest zaskoczonych odkryciem, że każde interaktywne naciśnięcie klawisza generuje oddzielny pakiet danych. To oznacza, że naciśnięcia klawiszy są przesyłane od klienta do serwera pojedynczo (jednorazowo jeden znak, a nie jeden wiersz). Co więcej, program `ssh` wywołuje powłokę (interpreter poleceń) w zdalnym systemie (serwerze), która powtarza znaki wprowadzone w systemie klienta. W ten sposób pojedynczy znak wprowadzony przez użytkownika mógłby wygenerować cztery segmenty TCP: interaktywnie wprowadzony znak przesyłany przez klienta, potwierdzenie odebrania tego znaku przez serwer, echo znaku wygenerowane przez serwer i potwierdzenie echa przesłane przez klienta z powrotem do serwera.

Normalnie jednak segmenty numer 2 i 3 są łączone — jak to pokazujemy na rysunku 15.1 (b), potwierdzenie naciśnięcia klawisza jest przesyłane razem z echem wprowadzonych znaków. Technikę, która łączy te transmisje (zwaną **potwierdzeniem opóźnionym z jazdą „na barana"**, *delayed acknowledgement with piggybacking*), opiszemy w następnym podrozdziale.



Rysunek 15.1. *Jednym z możliwych sposobów zdalnej implementacji echa naciśnięcia klawisza w trybie interaktywnym jest wysłanie oddzielnego pakietu ACK i oddzielnego pakietu z echem znaku (a). Typowe zachowanie protokołu TCP polega na połączeniu potwierdzenia ACK dla bajta danych i echa tego bajta w pojedynczym pakiecie (b)*

W naszym przykładzie wykorzystaliśmy program `ssh` celowo, ponieważ generuje pakiet przesyłany od klienta do serwera dla każdego wprowadzonego znaku. Jeśli jednak użytkownik pisze na klawiaturze szczególnie szybko, w pojedynczym pakiecie może zostać przeniesiony więcej niż jeden znak. Na rysunku 15.2 pokazujemy przy użyciu programu Wireshark przepływ danych mający miejsce, gdy przesyłamy do linuksowego serwera polecenie `date` poprzez aktywne połączenie `ssh`.



Rysunek 15.2. *Segmenty TCP przesłane w wyniku wprowadzenia polecenia date, w ramach już ustanowionego połączenia ssh*

Na rysunku 15.2 pakiet numer 1 przenosi znak `d` od klienta do serwera. Pakiet numer 2 jest potwierdzeniem dla tego znaku, zawiera również jego echo (łącząc w jednym dwa środkowe segmenty, co przedstawiamy na rysunku 15.1). Pakiet numer 3 jest potwierdzeniem

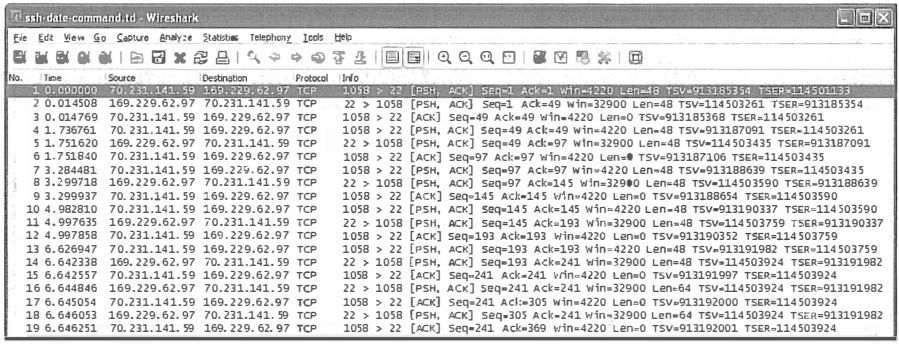
przesłanego echa znaku. Pakiety od 4. do 6. odpowiadają znakowi a, pakiety od 7. do 9. — znakowi t, a pakiety od 10. do 12. — znakowi e. Pakiety od 13. do 15. odpowiadają naciśnięciu klawisza *Enter* (powrót karetki). Opóźnienia między pakietami 3. i 4., 6. i 7., 9. i 10. oraz 12. i 13. są odstępami czasu między kolejnymi znakami wprowadzanymi przez człowieka, które, w tym przypadku, celowo zostały nienormalnie wydłużone (ok. 1,5 s) dla uzyskania lepszego obrazu.

Zauważmy, że pakiety od 16. do 19. różnią się nieco, ponieważ ich rozmiar wzrósł z 48 do 64 bajtów. Pakiet 16. zawiera dane wyjściowe polecenia `date`, wysłane przez serwer. Te 64 bajty zawierają zaszyfrowaną wersję następujących 28 znaków otwartego (jeszcze niezaszyfrowanego) tekstu:

```
Wed Dec 28 22:47:16 PST 2005
```

plus znaki powrotu karetki i nowego wiersza na końcu. Następny pakiet przesłany od serwera do klienta (pakiet 18.) zawiera zgłoszenie gotowości klienta na hoście serwera: `Linux%`. Pakiet numer 19 potwierdza te dane.

Na rysunku 15.3 przedstawiamy ten sam ślad, co na rysunku 15.2, ale tym razem pokazujemy więcej informacji dotyczących warstwy protokołu TCP, ukazujących sposób działania potwierdzeń TCP i rozmiary pakietów używane przez `ssh`. Pakiet 1. (zawierający znak `d`) rozpoczyna od względnego numeru sekwencyjnego 1. Pakiet 2. potwierdza przez `ACK` pakiet z wiersza numer 1, przypisując numerowi ACK wartość 49, równą numerowi sekwencyjnemu ostatniego prawidłowo odebranego bajta plus 1. Pakiet numer 2 przesyła także, od serwera do klienta, bajt danych z numerem sekwencyjnym 1, zawierający echo znaku `d`. Echo znaku `d` jest potwierdzone przez klienta sygnałem `ACK` w pakiecie numer 3 przez nadanie numerowi ACK wartości 49. Widzimy, że połączenie używa dwóch strumieni numerów sekwencyjnych — jeden dotyczy danych przesyłanych od klienta do serwera, a drugi odwrotnego kierunku transmisji. Zbadamy tę kwestię bardziej szczegółowo, kiedy będziemy omawiać propozycje okna.


ssh-date-command.td - Wireshark

File Edit View Go Capture Analyze Statistics Telephony Tools Help

No.	Time	Source	Destination	Protocol	Info
1	0.000000	70.231.141.59	169.229.62.97	TCP	1058 > 22 [PSH, ACK] Seq=1 Ack=1 Win=4220 Len=48 TSV=913185354 TSER=114501133
2	0.014508	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=1 Ack=49 Win=32900 Len=48 TSV=114503261 TSER=913185354
3	0.014769	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=49 Ack=49 Win=4220 Len=0 TSV=913185368 TSER=114503261
4	1.736761	70.231.141.59	169.229.62.97	TCP	1058 > 22 [PSH, ACK] Seq=49 Ack=49 Win=4220 Len=48 TSV=913187091 TSER=114503261
5	1.751620	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=49 Ack=97 Win=32900 Len=48 TSV=114503435 TSER=913187091
6	1.751840	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=97 Ack=97 Win=4220 Len=0 TSV=913187106 TSER=114503435
7	3.284481	70.231.141.59	169.229.62.97	TCP	1058 > 22 [PSH, ACK] Seq=97 Ack=97 Win=4220 Len=48 TSV=913188639 TSER=114503435
8	3.299718	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=97 Ack=145 Win=32900 Len=48 TSV=114503590 TSER=913188639
9	3.299937	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=145 Ack=145 Win=4220 Len=0 TSV=913188654 TSER=114503590
10	4.982810	70.231.141.59	169.229.62.97	TCP	1058 > 22 [PSH, ACK] Seq=145 Ack=145 Win=4220 Len=48 TSV=913190337 TSER=114503590
11	4.997635	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=145 Ack=193 Win=32900 Len=48 TSV=114503759 TSER=913190337
12	4.997858	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=193 Ack=193 Win=4220 Len=0 TSV=913190352 TSER=114503759
13	6.626947	70.231.141.59	169.229.62.97	TCP	1058 > 22 [PSH, ACK] Seq=193 Ack=193 Win=4220 Len=48 TSV=913191982 TSER=114503759
14	6.642338	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=193 Ack=241 Win=32900 Len=48 TSV=114503924 TSER=913191982
15	6.642557	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=241 Ack=241 Win=4220 Len=0 TSV=913191997 TSER=114503924
16	6.644846	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=241 Ack=241 Win=32900 Len=64 TSV=114503924 TSER=913191982
17	6.645054	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=241 Ack=305 Win=4220 Len=0 TSV=913192000 TSER=114503924
18	6.646053	169.229.62.97	70.231.141.59	TCP	22 > 1058 [PSH, ACK] Seq=305 Ack=241 Win=32900 Len=64 TSV=114503924 TSER=913191982
19	6.646251	70.231.141.59	169.229.62.97	TCP	1058 > 22 [ACK] Seq=241 Ack=369 Win=4220 Len=0 TSV=913192001 TSER=114503924

Rysunek 15.3. *Ten sam ślad, co na rysunku 15.2, z tym wyjątkiem, że w opcjach wyświetlania programu Wireshark wyłączono identyfikowanie pakietów jako należących do protokołu ssh, odsłaniając informacje protokołu TCP dotyczące numerów sekwencyjnych. Zauważmy, że wszystkie pakiety z danymi mają rozmiar 48 bajtów, z wyjątkiem dwóch ostatnich. Rozmiar 48 bajtów wiąże się z zastosowanym w ssh szyfrowaniem (patrz rozdział 18.)*

Kolejne spostrzeżenie odnoszące się do powyższego śladu dotyczy tego, że każdy pakiet zawierający dane (niezerowej długości) ma także ustawiony bit PSH. Jak wcześniej wspominaliśmy, flaga ta jest konwencjonalnie używana do wskazania, że bufor strony wysyłającej pakiet został całkowicie opróżniony w związku z wysłaniem tego pakietu. Mówiąc inaczej, kiedy pakiet z ustawionym bitem PSH opuszczał nadawcę, ten ostatni nie miał już żadnych danych do wysłania.

15.3. Potwierdzenia opóźnione

W wielu przypadkach protokół TCP nie dostarcza potwierdzenia dla każdego przychodzącego pakietu. Jest to możliwe, ponieważ pole *ACK* nagłówka TCP ma kumulatywny charakter (patrz rozdział 12.). Użycie kumulatywnego potwierdzenia ACK umożliwia protokołowi TCP celowe opóźnienie wysłania potwierdzenia ACK o pewien okres czasu w nadziei, że będzie mógł połączyć potwierdzenie ACK, które ma do wysłania, z jakimiś danymi, które lokalna aplikacja zechce wysłać w tym samym kierunku. To coś w rodzaju „jazdy na barana" (*piggybacking*) i jest używane najczęściej w związku z transferami danych masowych. Oczywiście, protokół TCP nie może opóźniać wysłania potwierdzeń ACK w nieskończoność; w takim przypadku druga strona mogłaby dojść do wniosku, że dane zostały utracone, i zainicjować niepotrzebną retransmisję.

Dokument [RFC1122] stwierdza, że protokół TCP powinien zaimplementować opóźnione potwierdzenia ACK, ale opóźnienie musi być mniejsze niż 500 ms. Wiele implementacji stosuje maksimum wynoszące 200 ms.

Opóźnianie potwierdzeń ACK powoduje zmniejszenie ruchu w sieci w porównaniu z sytuacją, gdy potwierdzenia ACK nie są opóźniane, ponieważ wysyła się wtedy mniej takich potwierdzeń. W przypadku transferów danych masowych występuje dość często stosunek 2 do 1. Użycie opóźnionych potwierdzeń ACK i maksymalna ilość czasu, przez jaki protokół TCP może czekać przed wysłaniem potwierdzenia ACK, mogą być konfigurowane w zależności od systemu operacyjnego hosta. W systemie Linux użyto algorytmu dynamicznej regulacji, dzięki któremu może zmieniać się sposób działania między potwierdzaniem każdego segmentu (tryb szybkich potwierdzeń, zwany krótko po angielsku trybem *quickack*) i konwencjonalnym trybem potwierdzeń opóźnionych. W systemie Mac OS X sposób użycia opóźnionych potwierdzeń ACK określa zmienna systemowa `net.inet.tcp.delayed_ack`. Znaczenie przyjmowanych przez nią wartości jest następujące: wyłącz opóźnienia (0), zawsze opóźniaj (1), potwierdzaj co drugi pakiet (2), wykrywaj automatycznie, kiedy należy odpowiedzieć (3). Wartość domyślna wynosi 3. W najnowszych wersjach systemu Windows wpisy w kluczach rejestru

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\IG
```

(gdzie *IG* oznacza identyfikator GUID konkretnego interfejsu sieciowego) odpowiadające poszczególnym interfejsom sieciowym działają nieco inaczej. Parametr TcpAckFrequency (który należy dodać) przyjmuje wartości z zakresu od 0 do 255, a jego wartość domyślna wynosi 2. Określa on liczbę zaległych potwierdzeń ACK wystarczających do zignorowania licznika czasu opóźnienia ACK. Przypisanie mu wartości 1 faktycznie powoduje generowanie potwierdzenia ACK dla każdego odebranego segmentu. Jeśli jest używany licznik czasu ACK, może być regulowany przez wpis rejestru TcpDelAckTicks. Wartość tego parametru musi mieścić się w zakresie od 2 do 6, a domyślnie wynosi 2. Oznacza ona liczbę setek milisekund, które należy odczekać przed wysłaniem opóźnionego potwierdzenia ACK.

Ze wspomnianych wcześniej powodów protokół TCP jest na ogół skonfigurowany tak, żeby opóźniać potwierdzenia ACK w pewnych okolicznościach, ale żeby nie opóźniać ich zbyt długo. Szerokie zastosowanie opóźnionych potwierdzeń ACK opisujemy w rozdziale 16., omawiając funkcjonowanie kontroli przeciążenia w protokole TCP podczas masowych transferów przy użyciu dużych pakietów. Kiedy używane są mniejsze pakiety, takie jak w przypadku aplikacji interaktywnych, do gry wchodzi inny algorytm. Połączenie tego algorytmu z opóźnionymi potwierdzeniami ACK może prowadzić do kiepskiej wydajności, jeśli nie jest obsługiwane ostrożnie, więc przyjrzymy się mu teraz bardziej szczegółowo.

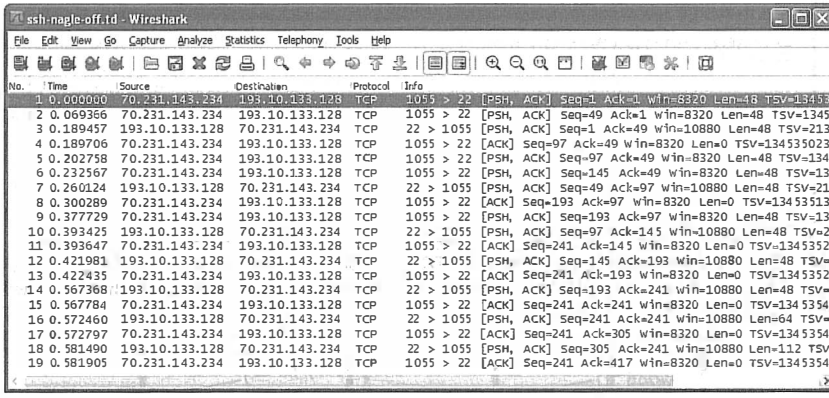
15.4. Algorytm Nagle'a

Jak widzieliśmy w poprzednim podrozdziale, w połączeniu ssh pojedyncze naciśnięcie klawisza jest często od razu przesyłane przez sieć, od klienta do serwera. W przypadku użycia protokołu IPv4 wysłanie pojedynczego znaku z klawiatury generuje pakiet TCP/IPv4 o rozmiarze ok. 88 bajtów (przy zastosowaniu szyfrowania i uwierzytelniania, jak we wcześniejszym przykładzie): 20 bajtów dla nagłówka IP, 20 bajtów dla nagłówka TCP (zakładając brak opcji) i 48 bajtów danych. Te małe pakiety (noszące angielską nazwę *tinygram* = *tiny datagram*, maleńki datagram) mają stosunkowo duży narzut informacji opakowującej, wymaganej przy transmisji przez sieć. Znaczy to, że zawierają stosunkowo mało użytecznych danych aplikacji w porównaniu z resztą zawartości pakietu. Pakiety z dużym narzutem nie stanowią zwykle problemu w sieciach LAN, ponieważ większość sieci LAN nie jest przeciążona, a same pakiety nie muszą być transportowane zbyt daleko. Jednak te małe minipakiety mogą przyczynić się do wystąpienia przeciążenia i doprowadzić do nieefektywnego wykorzystania pojemności w sieciach rozległych. Proste i eleganckie rozwiązanie zostało zaproponowane przez Johna Nagle'a w dokumencie [RFC0896] i obecnie nazywane jest **algorytmem Nagle'a**. Najpierw opiszemy jego działanie, a następnie przeanalizujemy pewne pułapki i problemy, które mogą się zdarzyć w wyniku stosowania tego algorytmu razem z opóźnionymi potwierdzeniami ACK.

Zgodnie z algorytmem Nagle'a, jeśli w połączeniu TCP istnieją wysłane, a jeszcze niepotwierdzone dane, małe segmenty (o rozmiarze mniejszym od wartości parametru SMSS) nie mogą być transmitowane, dopóki wszystkie poprzednio wysłane dane nie zostaną potwierdzone. Wówczas małe ilości danych są kumulowane przez TCP i wysyłane w pojedynczym segmencie, kiedy nadejdzie potwierdzenie. Procedura ta praktycznie wymusza pracę protokołu TCP w trybie *stop-and-wait* (zatrzymaj się i czekaj) — protokół wstrzymuje transmisję, dopóki nie otrzyma potwierdzenia dla wszystkich już wysłanych danych. Piękno tego algorytmu polega na samoregulacji w aspekcie czasowym (stąd angielskie określenie *self-clocking*): im szybciej wracają potwierdzenia ACK, tym szybciej wysyłane są dane. W sieciach WAN, charakteryzujących się stosunkowo dużym opóźnieniem, gdzie wskazana jest redukcja liczby małych pakietów, w jednostce czasu jest przesyłanych mniej segmentów. Inaczej mówiąc, czas RTT steruje szybkością transmisji pakietów.

Widzieliśmy na rysunku 15.3, że czas RTT potrzebny na wysłanie, potwierdzenie i przesłanie echa pojedynczego bajta może być mały (poniżej 15 ms). Aby generować dane w szybszym tempie, musielibyśmy wprowadzać więcej niż 60 znaków w ciągu sekundy. To oznacza, że rzadko możemy zauważyć widoczne efekty działania tego algorytmu, przesyłając dane między dwoma hostami w połączeniu z małym czasem RTT występującym wówczas, gdy hosty znajdują się w tej samej sieci LAN.

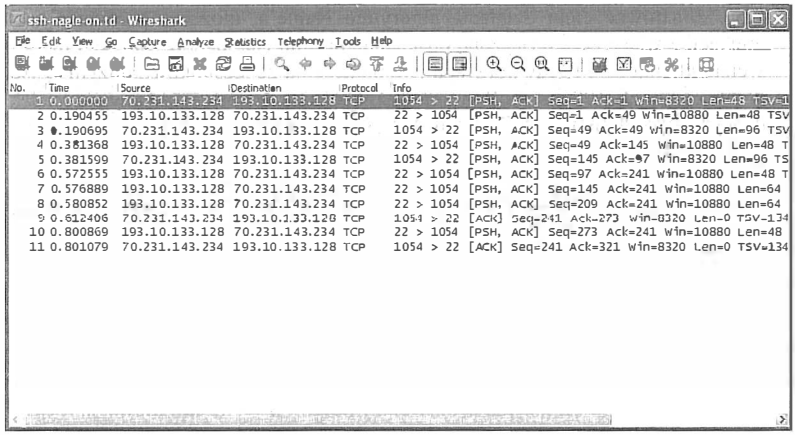
Aby zilustrować efekt działania algorytmu Nagle'a, możemy porównać zachowania aplikacji z włączoną i wyłączoną obsługą algorytmu Nagle'a. W tym celu modyfikujemy wersję klienta `ssh`. Kiedy korzystamy z połączenia ze stosunkowo dużym czasem RTT, wynoszącym ok. 190 ms, możemy zauważyć różnicę. Najpierw badamy przypadek z wyłączonym algorytmem Nagle'a (stan domyślny w protokole `ssh`), co pokazujemy na rysunku 15.4.



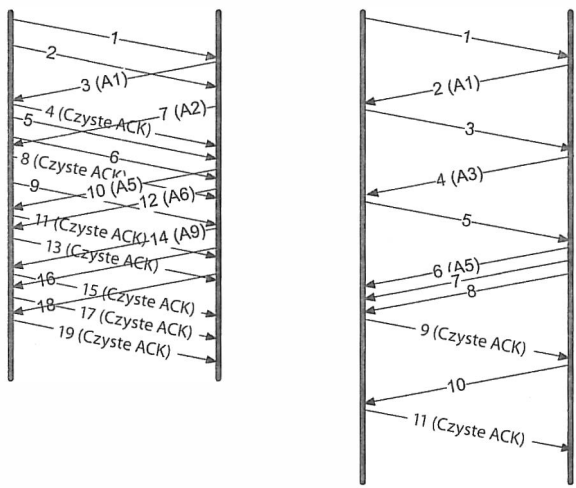
Rysunek 15.4. *Ślad ssh pokazujący połączenie TCP z czasem RTT wynoszącym ok. 190 ms. Algorytm Nagle'a jest wyłączony. Transmisje danych i potwierdzenia ACK przeplatają się, a cała wymiana trwa 0,58 s i obejmuje 19 pakietów. Wiele pakietów ma stosunkowo mały rozmiar (48 bajtów danych użytkownika). Czyste potwierdzenia ACK (segmenty, które nie zawierają danych) wskazują, że dane wyjściowe serwera (echo wprowadzanych znaków) zostały przetworzone przez klienta*

Ślad przedstawiony na rysunku 15.4 rozpoczyna się po zakończeniu początkowego protokołu uwierzytelniania i rozpoczęciu zdalnej sesji. Wtedy zostaje wprowadzone z klawiatury polecenie `date`. Widzimy, że zostało przechwycone 19 pakietów, a cała wymiana danych trwała 0,58 s. Pięć pakietów to żądania `ssh`, siedem z nich zawiera odpowiedzi `ssh`, a siedem pozostałych to czyste potwierdzenia `ACK` (niezawierające danych), wygenerowane na poziomie protokołu TCP. Jeśli powtórzymy ten pomiar wkrótce potem (tzn. w zbliżonych warunkach panujących w sieci), ale tym razem bez wyłączania algorytmu Nagle'a, zobaczymy działanie przedstawione na rysunku 15.5.

Natychmiast możemy zauważyć, że liczba pakietów na rysunku 15.5 jest mniejsza niż na rysunku 15.4 (o osiem). Drugą uderzającą różnicą jest regularność uporządkowania i odstępów czasowych żądań i odpowiedzi. Przypominamy, że algorytm Nagle'a wymusza funkcjonowanie protokołu TCP w trybie *stop-and-wait* (stój i czekaj), tak że protokół TCP nie może przejść do dalszego działania, dopóki nie zostaną odebrane potwierdzenia `ACK`. Jeśli przyjrzymy się czasom odpowiadającym każdej parze żądanie-odpowiedź — 0,0, 0,19, 0,38 i 0,57 — zauważymy, że naśladują one pewien wzorzec; każdą ich parę dzieli prawie dokładnie 190 ms, wartość bardzo zbliżona do czasu RTT połączenia. Konsekwencją konieczności odczekiwania jednego czasu RTT dla każdej pary żądanie-odpowiedź jest zwiększenie całkowitego czasu wymaganego do zakończenia wymiany (0,80 s zamiast 0,58 s przy wyłączonym algorytmie Nagle'a). Na tym polega kompromis wprowadzony przez algorytm Nagle'a: używa się mniejszej liczby większych pakietów, ale wymagane opóźnienie jest większe. Te dwa różne sposoby działania są widoczne wyraźniej na rysunku 15.6.



Rysunek 15.5. *Ślad ssh pokazujący połączenie TCP z czasem RTT wynoszącym 190 ms i działającym algorytmem Nagle'a. Każdemu żądaniu regularnie towarzyszy następująca po nim odpowiedź, a wymiana zajmuje 0,80 s przy użyciu 11 pakietów*



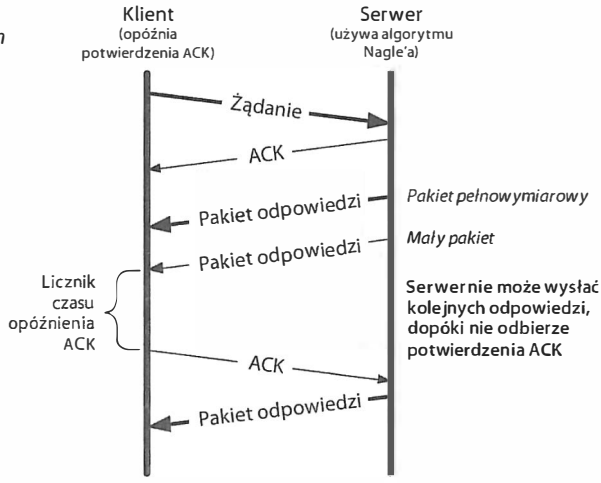
Wyłączony algorytm Nagle'a Włączony algorytm Nagle'a

Rysunek 15.6. *Porównanie przypadków użycia i nieużycia algorytmu Nagle'a w połączeniach TCP działających w podobnym środowisku operacyjnym. Gdy algorytm Nagle'a jest włączony, co najwyżej jeden pakiet może czekać na potwierdzenie w dowolnym momencie czasu. Zmniejsza to ilość małych pakietów, ale wydłuża czas opóźnienia*

Efekt działania według zasady *stop-and-wait* (stój i czekaj) właściwej dla algorytmu Nagle'a jest wyraźnie widoczny na rysunku 15.6. Wymiana pakietów przedstawiona po lewej stronie utrzymuje obydwa kierunki połączenia w stanie dużej aktywności, natomiast przy włączonym algorytmie Nagle'a w dowolnym czasie jest aktywny tylko jeden z kierunków połączenia.

15.4.1. Interakcja opóźnionych potwierdzeń ACK i algorytmu Nagle'a

Jeśli zastanowimy się nad tym, co się może zdarzyć, gdy opóźnione potwierdzenia ACK są używane razem z algorytmem Nagle'a, możemy łatwo skonstruować niepożądany scenariusz. Weźmy pod uwagę klienta używającego opóźnionych potwierdzeń ACK, który wysyła żądanie do serwera, a serwer odpowiada taką ilością danych, która nie mieści się w całości w pojedynczym pakiecie (patrz rysunek 15.7).



Rysunek 15.7. *Interakcja między algorytmem Nagle'a i opóźnionymi potwierdzeniami ACK. Może wystąpić chwilowe zakleszczenie, dopóki nie „wypali" licznik czasu opóźnienia ACK*

Widzimy tu, że klient po otrzymaniu dwóch pakietów od serwera wstrzymuje potwierdzenie ACK, mając nadzieję na zabranie „na barana" dodatkowych danych skierowanych do serwera. W zasadzie od protokołu TCP wymaga się wysłania potwierdzenia ACK po odebraniu dwóch pakietów, ale tylko wtedy, gdy są one pełnowymiarowe, a w tym przypadku tak nie jest. Z kolei po stronie serwera, gdzie działa algorytm Nagle'a, żadne kolejne pakiety nie mogą być wysłane do klienta, dopóki nie zostanie zwrócone potwierdzenie ACK, ponieważ co najwyżej jeden „mały" pakiet może być niepotwierdzony. Połączenie opóźnionych potwierdzeń ACK i algorytmu Nagle'a prowadzi do pewnej formy **zakleszczenia** (*deadlock*; każda ze stron czeka na akcję drugiej strony; patrz [MMSV99], [MM01]). Na szczęście, to zakleszczenie nie jest definitywne i zostaje przełamane, kiedy pojawia się sygnał od licznika czasu opóźnienia ACK, który zmusza klienta do wysłania potwierdzenia ACK, nawet jeśli nie ma dodatkowych danych do przesłania. Jednak cały transfer danych zostaje wstrzymany na czas trwania tego zakleszczenia, co jest zazwyczaj niepożądane. Algorytm Nagle'a może zostać w takich warunkach wyłączony, co widzieliśmy w przypadku protokołu ssh.

15.4.2. Wyłączenie algorytmu Nagle'a

Jak mogliśmy wywnioskować z poprzedniego przykładu, są okoliczności, w których algorytm Nagle'a musi zostać wyłączony. Typowe przykłady obejmują przypadki, gdzie wymaga się możliwie najmniejszego opóźnienia, np. kiedy ruch myszy lub naciśnięcie

klawisza muszą być przekazane bez opóźnienia, aby dostarczyć odpowiedzi w czasie rzeczywistym użytkownikowi, którego terminal jest obsługiwany zdalnie. Inny przykład dotyczy gier w trybie online z udziałem wielu użytkowników, w których ruchy postaci muszą być przekazywane tak szybko, jak to możliwe, aby nie zakłócać efektów czynności użytkownika (i nie opóźniać za bardzo ich pojawienia się u innych graczy).

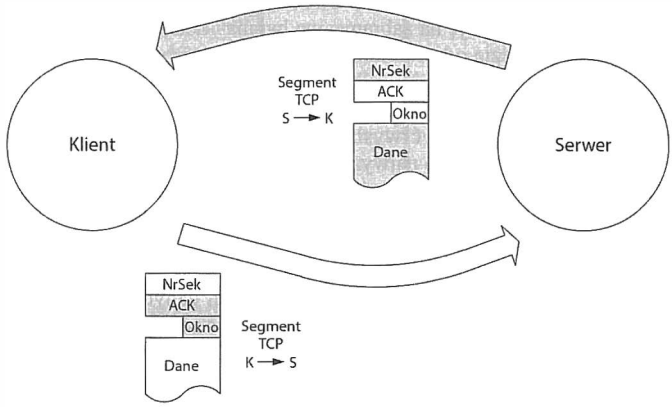
Algorytm Nagle'a może zostać wyłączony na szereg sposobów. Możliwość wyłączenia go jest wymagana przez dokument [RFC1122]. Aplikacja korzystająca z interfejsu API gniazd Berkeley może użyć opcji `TCP_NODELAY`. Ponadto istnieje możliwość wyłączenia algorytmu Nagle'a dla całego systemu. W systemie Windows można to osiągnąć, korzystając z następującego klucza rejestru:

```
HKLM\SOFTWARE\Microsoft\MSMQ\Parameters\TCPNoDelay
```

Wartość typu DWORD, która musi być dodana przez użytkownika, powinna być ustawiona na 1, aby wyłączyć algorytm Nagle'a. Żeby ta zmiana była efektywna, należy być może zainstalować MS Message Queuing (implementację kolejki komunikatów stosowaną w systemach Windows; patrz [MMQ]).

15.5. Sterowanie przepływem i zarządzanie oknem

W rozdziale 12. pisaliśmy, że w celu zaimplementowania sterowania przepływem można użyć okna przesuwnego o zmiennej szerokości. Na rysunku 15.8 klient i serwer TCP oddziałują na siebie wzajemnie, wymieniając między sobą informacje o przepływie danych, łącznie z numerami sekwencyjnymi segmentów, numerami ACK i rozmiarem okna (tzn. wielkością dostępnego miejsca u odbiorcy).



Rysunek 15.8. *Każde połączenie TCP jest dwukierunkowe. Dane płynące w jednym kierunku sprawiają, że strona docelowa odpowiada potwierdzeniami ACK i propozycjami okna. To samo dotyczy przeciwnego kierunku połączenia*

Dwie duże strzałki na rysunku 15.8 pokazują kierunek przepływu danych (tzn. kierunek, w którym są wysyłane segmenty TCP). Ponieważ, jak pamiętamy, w każdym połączeniu TCP dane przepływają w obu kierunkach, mamy dwie strzałki, jedną wskazującą kie-

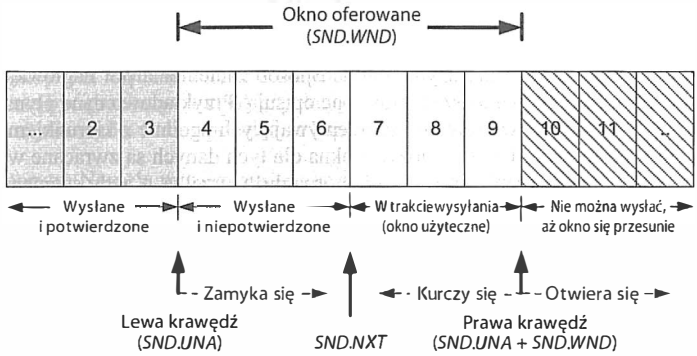
runek od klienta do serwera (K→S) i drugą pokazującą kierunek od serwera do klienta (S→K). Każdy segment zawiera numer ACK i informację dotyczącą okna, może również zawierać jakieś dane użytkownika. Sposób zacienienia pól nagłówka TCP zależy od kierunku przepływu danych, który one opisują. Przykładowo dane płynące w kierunku K→S są zawarte w segmentach przepływających zgodnie z kierunkiem dolnej dużej strzałki, ale numer ACK i propozycja okna dla tych danych są zwracane w segmentach, których kierunek przepływu wskazuje górna duża strzałka. Każdy segment TCP (z wyjątkiem tych, które są wymieniane podczas ustanawiania połączenia) zawiera istotne pola: *Numer sekwencyjny*, *Numer ACK*, czyli *Numer potwierdzenia*, oraz *Rozmiar okna* (pole zawierające propozycję okna).

W każdym z przykładów dotyczących protokołu ssh, przedstawionych dotychczas w tym rozdziale, widzieliśmy niezmieniającą się propozycję okna przekazywaną przez jedną stronę połączenia TCP drugiej stronie. Przykłady te zawierają wartości 8320 bajtów, 4220 bajtów i 32 900 bajtów. Rozmiary te reprezentują ilość miejsca zarezerwowanego przez nadawcę segmentu (z propozycją okna) na przechowanie przychodzących danych wysyłanych przez partnera w połączeniu. Kiedy aplikacje korzystające z protokołu TCP nie są zajęte wykonywaniem innych zadań, są zazwyczaj zdolne do skonsumowania wszystkich bez wyjątku danych odebranych przez protokół TCP i dla nich zakolejkowanych, co prowadzi do braku zmiany wartości pola *Rozmiar okna* w trakcie połączenia. W wolnych systemach lub gdy aplikacja ma równocześnie inne rzeczy do wykonania, może wystąpić taka sytuacja, że przyszły dane dla niej przeznaczone, które już zostały potwierdzone przez TCP i zajmują miejsce w kolejce, czekając na aplikację, która je odczyta lub „skonsumuje". Kiedy protokół TCP zaczyna w ten sposób kolejkować dane, ilość wolnego miejsca zdolnego pomieścić nowe przychodzące dane ulega zmniejszeniu, a zmiana ta znajduje odbicie w zmniejszeniu wartości pola *Rozmiar okna*. W końcu, jeśli aplikacja wcale nie odczytuje ani w inny sposób nie konsumuje tych danych, protokół TCP musi podjąć jakieś działanie, które spowoduje całkowite zaprzestanie wysyłania przez nadawcę nowych danych, ponieważ nie byłoby dla nich miejsca, gdyby dotarły do odbiorcy. Jest to realizowane przez wysłanie propozycji okna o wartości 0 (brak miejsca).

Pole *Rozmiar okna* w każdym nagłówku TCP pokazuje wyrażoną w bajtach ilość wolnego miejsca , które pozostaje w buforze odbiorcy. Pole zajmuje 16 bitów w nagłówku TCP, ale w połączeniu z opcją skalowania okna możliwe jest użycie wartości większych niż 65 535 (patrz rozdział 13.). Wartość największego numeru sekwencyjnego, który nadawca segmentu jest gotów przyjąć z przeciwnego kierunku, jest równa sumie pól *Numer potwierdzenia* i *Rozmiar okna* (odpowiednio przeskalowany) nagłówka TCP.

15.5.1. Okna przesuwne

Każdy z punktów końcowych połączenia TCP jest zdolny do wysyłania i odbierania danych. Kontrola ilości danych wysyłanych lub odbieranych w połączeniu jest utrzymywana przez **struktury okna**. W każdym aktywnym połączeniu TCP każdy z punktów końcowych tego połączenia utrzymuje strukturę **okna nadawczego** (*send window*) oraz strukturę **okna odbiorczego** (*receive window*). Struktury te są podobne do koncepcyjnych struktur okna opisanych w rozdziale 12., ale w tym miejscu omówimy je bardziej szczegółowo. Na rysunku 15.9 pokazujemy hipotetyczną strukturę okna nadawczego protokołu TCP.



Rysunek 15.9. *Struktura okna przesuwnego protokołu TCP po stronie nadawcy śledzi, które numery sekwencyjne zostały już potwierdzone, które są w trakcie wysyłania, a które jeszcze są do wysłania. Rozmiar oferowanego okna jest regulowany przez pole Rozmiar okna, przesyłane przez odbiorcę w każdym potwierdzeniu ACK*

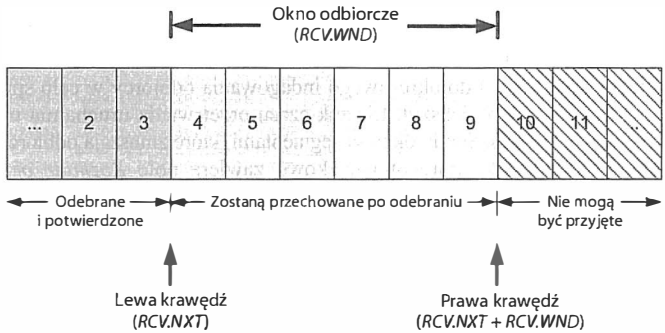
Protokół TCP utrzymuje swoje struktury okien, określając ich wielkość w bajtach (nie w pakietach). Na rysunku 15.9 ponumerowaliśmy bajty od 2 do 11. Okno proponowane przez odbiorcę nazywa się **oknem oferowanym** i obejmuje bajty od 4. do 9., co oznacza, że odbiorca potwierdził wszystkie bajty do bajta numer 3 włącznie i zaproponował rozmiar okna równy 6. Przypomnijmy sobie z rozdziału 12., że pole *Rozmiar okna* zawiera wyrażone w bajtach przesunięcie względem numeru ACK. Nadawca oblicza swoje **okno użyteczne**, które określa, ile danych może wysłać natychmiast. Rozmiar okna użytecznego równa się rozmiarowi okna oferowanego minus ilość danych już wysłanych, ale jeszcze niepotwierdzonych. Zmienne `SND.UNA` i `SND.WND` przechowują, odpowiednio, pozycję lewej krawędzi i rozmiar okna oferowanego. Zmienna `SND.NXT` zawiera następny numer sekwencyjny do wysłania, więc rozmiar okna użytecznego jest równy (`SND.UNA` + `SND.WND` – `SND.NXT`).

Z upływem czasu okno przesuwne przemieszcza się w prawo w miarę, jak odbiorca potwierdza dane. Zmiana położenia dwóch końców okna względem siebie zwiększa lub zmniejsza rozmiar okna. Do opisania ruchu prawej i lewej krawędzi okna są używane trzy określenia.

- Okno **zamyka się**, kiedy lewa krawędź przesuwa się w prawo. Sytuacja ta występuje, gdy wysłane przedtem dane zostają potwierdzone i rozmiar okna zmniejsza się.
- Okno **otwiera się**, kiedy prawa krawędź przesuwa się w prawo, pozwalając na wysłanie kolejnych danych. To się dzieje, kiedy proces odbierający dane po drugiej stronie odczytuje potwierdzone dane, zwalniając miejsce w buforze odbiorczym swojego protokołu TCP.
- Okno **kurczy się** (*shrinks*), kiedy prawa krawędź przesuwa się w lewo. Dokument [RFC1122] usilnie odradza tę ewentualność, ale protokół TCP musi być zdolny do jej obsługi. W punkcie 15.5.3 opisującym syndrom głupiego okna podajemy przykład, w którym jedna ze stron chciałaby spowodować skurczenie się okna przez przesunięcie prawej krawędzi w lewo, ale nie może tego zrobić.

Ponieważ każdy segment TCP zawiera zarówno numer ACK, jak i propozycję okna, nadawca TCP reguluje strukturę okna na podstawie tych dwóch wartości zawsze wtedy, kiedy przychodzi nowy segment. Lewa krawędź okna nie może przesunąć się w lewo, ponieważ jest określana przez numer ACK otrzymywany od drugiej strony połączenia, który ma kumulatywny charakter i nigdy nie zmniejsza swojej wartości. Kiedy numer ACK przesuwa lewą krawędź okna do przodu (czyli w prawo), a rozmiar okna nie ulega zmianie (nagminny przypadek), mówi się, że okno przesuwa się do przodu. Jeśli w innych przychodzących potwierdzeniach ACK numer ACK zwiększa swą wartość, ale propozycja okna maleje, lewa krawędź okna zbliża się do jego prawej krawędzi. Jeśli lewa krawędź okna pokryje się z prawą krawędzią, mamy do czynienia z oknem zerowym. Sytuacja ta zatrzymuje jakiekolwiek transmisje danych ze strony nadawcy. W tym przypadku protokół nadawczy TCP zaczyna **sondować** (*probe*) okno odbiorcy (patrz punkt 15.5.2) w oczekiwaniu na zwiększenie oferowanego okna.

Odbiorca również utrzymuje strukturę okna, która jest nieco prostsza niż u nadawcy. Struktura okna odbiorcy śledzi, jakie dane zostały już odebrane i potwierdzone, i aktualizuje na bieżąco wartość maksymalnego numeru sekwencyjnego, jaki może zostać przyjęty. Odbiorca TCP, polegając na tej strukturze, stara się zapewnić poprawność odbieranych danych. Szczególnie zaś pragnie uniknąć przechowywania duplikatów bajtów, które wcześniej zostały odebrane i potwierdzone, a także chciałby uniknąć przechowywania bajtów, których nie powinien był otrzymać (chodzi tu o bajty danych posiadające numery sekwencyjne przekraczające prawą krawędź okna nadawcy). Struktura okna odbiorcy została zilustrowana na rysunku 15.10.



Rysunek 15.10. *Struktura okna przesuwnego protokołu TCP po stronie odbiorcy pomaga temu odbiorcy w ustaleniu, których numerów sekwencyjnych ma oczekiwać w następnej kolejności. Dane z numerami sekwencyjnymi mieszczącymi się w oknie odbiorczym są przechowywane po odebraniu. Dane wykraczające poza zakres okna są odrzucane*

Powyższa struktura także zawiera lewą i prawą krawędź okna, podobnie jak okno nadawcy, ale bajty mieszczące się w oknie (bajty od 4. do 9. na rysunku) nie podlegają podziałowi na kategorie, tak jak to się dzieje w strukturze okna nadawcy. W przypadku odbiorcy wszystkie odebrane bajty z numerami sekwencyjnymi mniejszymi niż lewa krawędź okna (wskazywana przez zmienną RCV.NXT) są odrzucane jako duplikaty, a wszystkie bajty odebrane z numerami sekwencyjnymi przekraczającymi prawą krawędź okna (RCV.WND bajtów powyżej wartości RCV.NXT) są odrzucane jako wykraczające poza dopuszczalny zakres. Przychodzące bajty, posiadające numer sekwencyjny mieszczący się w zakresie

okna odbiorczego, są akceptowane. Zauważmy, że numer ACK generowany przez odbiorcę może zostać zwiększony tylko wtedy, gdy segmenty wypełnią obszar przylegający bezpośrednio do lewej krawędzi okna, z powodu kumulatywnej struktury potwierdzeń ACK w protokole TCP. Jeśli używane są selektywne potwierdzenia ACK, pozostałe mieszczące się w oknie segmenty mogą zostać potwierdzone przy użyciu opcji SACK protokołu TCP, ale sam numer potwierdzenia ACK jest zwiększany tylko wtedy, kiedy zostaną odebrane dane graniczące z lewą krawędzią okna (więcej szczegółów na temat opcji SACK można znaleźć w rozdziale 14.).

15.5.2. Okna zerowe i licznik czasu przetrwania w protokole TCP

Zobaczyliśmy, że protokół TCP implementuje sterowanie przepływem, wymagając od odbiorcy określenia ilości danych, jaką jest gotów przyjąć od nadawcy, czyli propozycji okna. Kiedy wartość okna proponowanego przez odbiorcę spada do 0, nadawca jest praktycznie powstrzymany przed transmitowaniem danych, dopóki okno nie stanie się niezerowe. Kiedy odbiorca odzyskuje wolne miejsce, wysyła do nadawcy **aktualizację okna** (*window update*), aby pokazać, że dane mogą znów płynąć. Ponieważ tego typu aktualizacje na ogół nie zawierają danych (stanowią formę „czystego potwierdzenia ACK"), nie są dostarczane przez protokół TCP w sposób niezawodny. Protokół TCP musi więc obsługiwać przypadek, gdy aktualizacje, które otworzyłyby okno, są gubione.

Jeżeli potwierdzenie (zawierające aktualizację okna) zostanie utracone, może się skończyć na tym, że obie strony połączenia czekają jedna na drugą: odbiorca czeka na otrzymanie danych (ponieważ dostarczył nadawcy niezerowe okno i spodziewa się przyjścia danych), a nadawca czeka na aktualizację okna pozwalającą mu wznowić transmisję. Aby zapobiec wystąpieniu tej formy zakleszczenia, nadawca używa **licznika czasu przetrwania** (*persist timer*) do okresowego indagowania odbiorcy w celu sprawdzenia, czy rozmiar okna się nie zwiększył. Licznik czasu przetrwania uruchamia transmisję **sond okna** (*window probes*). Sondy okna są segmentami, które zmuszają odbiorcę do dostarczenia potwierdzenia ACK, które obowiązkowo zawiera pole *Rozmiar okna*. Dokument [RFC1122] zaleca, aby pierwsze sondowanie miało miejsce po upływie jednego czasu RTO, a kolejne próby powinny być podejmowane w odstępach czasu wzrastających wykładniczo (tzn. podobnie jak w „drugiej części" algorytmu Karna, który omawialiśmy w rozdziale 14.).

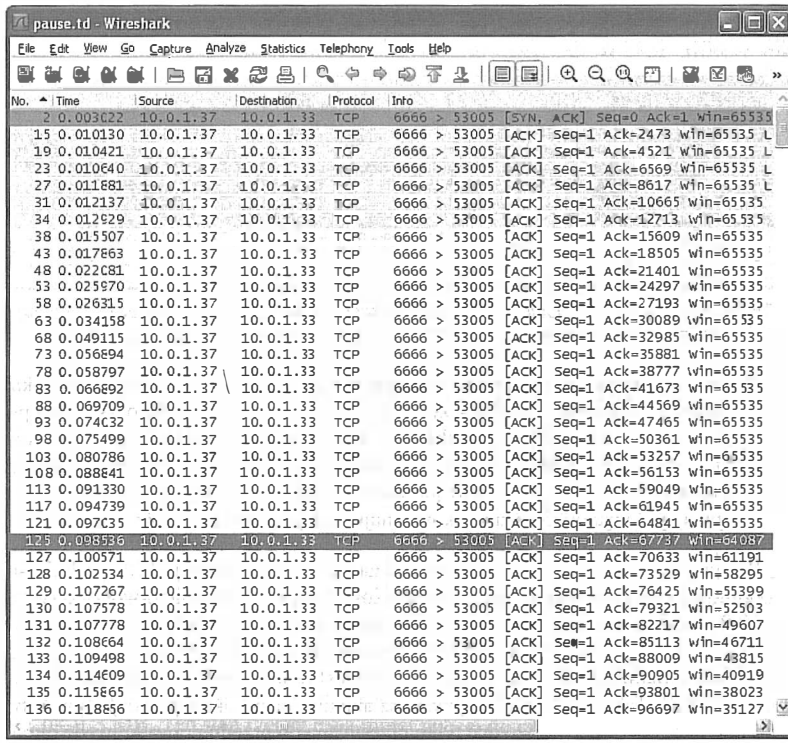
Sondy okna zawierają pojedynczy bajt danych i dlatego są dostarczane w niezawodny sposób (z retransmisją w przypadku ich utraty) przez protokół TCP, eliminując w ten sposób potencjalny stan zakleszczenia spowodowany przez utratę aktualizacji okna. Sondy są wysyłane za każdym razem, gdy wygasa licznik czasu przetrwania, a zawarty w nich bajt danych może zostać zaakceptowany przez odbiorcę lub nie, w zależności od tego, ile wolnego miejsca pozostaje w jego buforze. Podobnie jak w przypadku licznika czasu retransmisji (patrz rozdział 14.), można zastosować normalne odczekiwanie wykładnicze przy obliczaniu limitu czasu dla licznika czasu przetrwania. Istnieje jednak ważna różnica polegająca na tym, że normalny protokół TCP nigdy nie zaprzestaje wysyłania sond okna, podczas gdy może ostatecznie zrezygnować z prób wykonania retransmisji. Może to prowadzić do pewnej podatności na wyczerpanie zasobów, którą analizujemy w podrozdziale 15.7.

15.5.2.1. Przykład

Aby zilustrować użycie dynamicznej regulacji rozmiaru okna w protokole TCP, tworzymy połączenie TCP i sprawiamy, że proces odbierający dane wstrzymuje na pewien czas działanie przed skonsumowaniem danych z sieci. Do tego eksperymentu używamy systemu Mac OS X 10.6 jako nadawcy i systemu Windows 7 w roli odbiorcy. Odbiorca uruchamia program `sock` z flagą `-P` następująco:

```
C:\> sock -i -s -P 20 6666
```

Ten sposób wywołania sprawia, że odbiorca zawiesza działanie na okres 20 s przed skonsumowaniem danych z sieci. W rezultacie proponowane przez odbiorcę okno zaczyna się zamykać, począwszy od pakietu 125., co pokazujemy na rysunku 15.11.

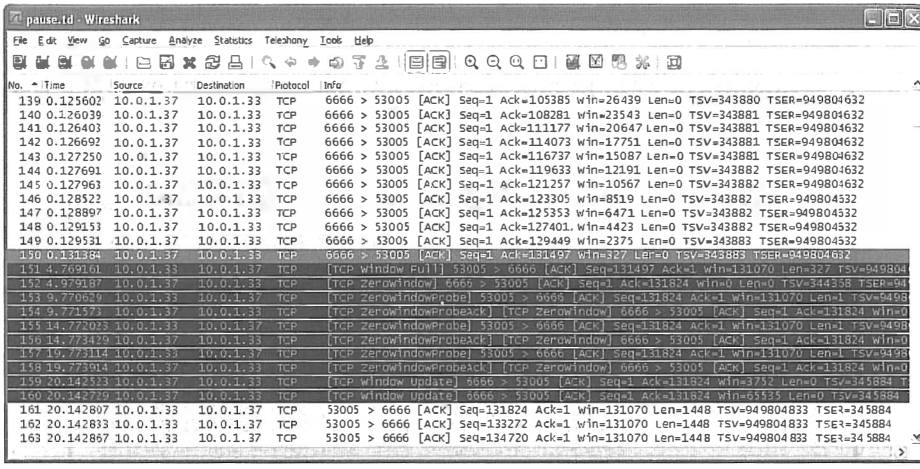


No.	Time	Source	Destination	Protocol	Info
2	0.003022	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [SYN, ACK] Seq=0 Ack=1 Win=65535
15	0.010130	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=2473 Win=65535 L
19	0.010421	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=4521 Win=65535 L
23	0.010640	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=6569 Win=65535 L
27	0.011881	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=8617 Win=65535 L
31	0.012137	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=10665 Win=65535
34	0.012929	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=12713 Win=65535
38	0.015507	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=15609 Win=65535
43	0.017863	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=18505 Win=65535
48	0.022081	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=21401 Win=65535
53	0.025970	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=24297 Win=65535
58	0.026315	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=27193 Win=65535
63	0.034158	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=30089 Win=65535
68	0.049115	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=32985 Win=65535
73	0.056894	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=35881 Win=65535
78	0.058797	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=38777 Win=65535
83	0.066892	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=41673 Win=65535
88	0.069709	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=44569 Win=65535
93	0.074032	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=47465 Win=65535
98	0.075499	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=50361 Win=65535
103	0.080786	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=53257 Win=65535
108	0.088641	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=56153 Win=65535
113	0.091330	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=59049 Win=65535
117	0.094739	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=61945 Win=65535
121	0.097035	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=64841 Win=65535
125	0.098536	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=67737 Win=64087
127	0.100571	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=70633 Win=61191
128	0.102534	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=73529 Win=58295
129	0.107267	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=76425 Win=55399
130	0.107578	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=79321 Win=52503
131	0.107778	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=82217 Win=49607
132	0.108664	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=85113 Win=46711
133	0.109498	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=88009 Win=43815
134	0.114609	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=90905 Win=40919
135	0.115865	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=93801 Win=38023
136	0.118856	10.0.1.37	10.0.1.33	TCP	6666 > 53005 [ACK] Seq=1 Ack=96697 Win=35127

Rysunek 15.11. *Po okresie, w którym proponowane okno nie zmienia rozmiaru, dalej wysyłane są potwierdzenia, ale rozmiar okna zmniejsza się w miarę zapełniania bufora odbiorcy. Jeśli odbierająca dane aplikacja nie konsumuje żadnych danych, a nadawca kontynuuje transmisję, rozmiar okna w końcu osiąga wartość 0*

W powyższym śladzie transmisji możemy zauważyć, że dla więcej niż 100 pakietów okno odbiorcy zachowuje stałą wartość 64 kB. Dzieje się tak za przyczyną algorytmu automatycznej regulacji okna, który przydziela pamięć odbiorczemu protokołowi TCP nawet bez żądania ze strony aplikacji. W końcu jednak tej pamięci zaczyna brakować,

więc widzimy, że okno zaczyna się zmniejszać, począwszy od pakietu 125. Potem następuje duża liczba potwierdzeń ACK, z których każde dalej zmniejsza okno, gdy tymczasem numery ACK rosną w tempie 2896 bajtów na pojedyncze potwierdzenie ACK. To pokazuje, że odbiorczy protokół TCP przechowuje dane, ale aplikacja ich nie konsumuje. Patrząc na dalszy ciąg śladu, widzimy, że w końcu odbiorca nie ma już miejsca, by pomieścić przychodzące dane (patrz rysunek 15.12).



Rysunek 15.12. *Bufor odbiorcy zapełnił się. Kiedy aplikacja odbierająca dane wznawia ich odczytywanie, aktualizacja okna informuje odbiorcę, że jest teraz okazja do transferu następnych danych*

Widzimy na rysunku, że pakiet 151. zapełnia małe 327-bajtowe okno, jak pokazuje pochodzący z programu Wireshark komentarz TCP Window Full (okno TCP jest pełne). Po ok. 200 ms, w czasie 4,979, przedstawiona jest propozycja zerowego okna, pokazująca, że nie można już odebrać żadnych danych. Nie jest to zaskoczeniem, kiedy weźmiemy pod uwagę, że nadawca zapełnił już ostatnie dostępne okno, o którym otrzymał informację, a aplikacja odbiorcza nie konsumuje żadnych danych, aż do czasu 20,143.

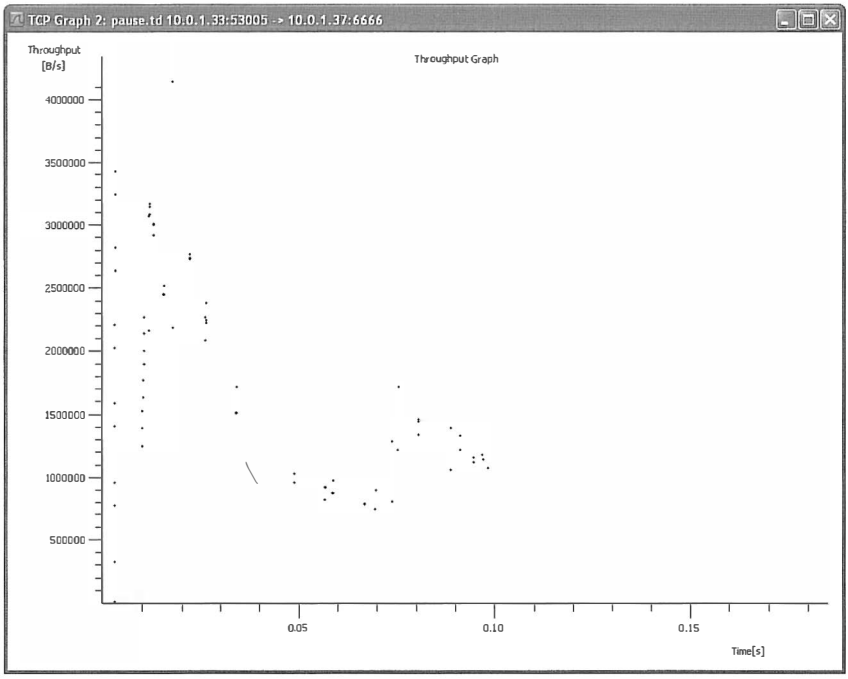
Po otrzymaniu propozycji zerowego okna protokół nadawczy TCP próbuje sondować odbiorcę trzy razy w odstępach 5 s, aby zobaczyć, czy okno nie otworzyło się. W czasie 20,0, zgodnie z wprowadzonym z klawiatury poleceniem, odbiorca zaczyna konsumować dane znajdujące się w kolejce protokołu TCP. Powoduje to wysłanie do nadawcy dwóch aktualizacji okna wskazujących, że jest teraz możliwa dalsza transmisja danych (do 64 kB). Takie segmenty nazywane są aktualizacjami okna, ponieważ nie potwierdzają żadnych nowych danych — one tylko przesuwają do przodu prawą krawędź okna. W tym momencie nadawca może wznowić normalną transmisję danych i dokończyć transfer.

Są liczne kwestie, które możemy podsumować, korzystając z rysunków 15.11 i 15.12.

- Nadawca nie musi transmitować pełnego okna danych.
- Pojedynczy segment od odbiorcy potwierdza dane i jednocześnie przesuwa okno w prawo. Dzieje się tak, ponieważ propozycja okna jest określona względem numeru ACK przesyłanego w tym samym segmencie.

- Rozmiar okna może zmniejszyć się, jak to pokazują serie potwierdzeń ACK na rysunku 15.11, ale prawa krawędź okna nie przesuwa się w lewo, aby uniknąć kurczenia się okna.
- Odbiorca nie musi czekać na zapełnienie okna przed wysłaniem potwierdzenia ACK.

W uzupełnieniu powyższych punktów warto spojrzeć na przepustowość osiąganą przez to połączenie jako na funkcję czasu. Używając funkcji programu Wireshark *Statistics/TCP Stream Graph/Throughput Graph*, obserwujemy szeregi czasowe pokazane na rysunku 15.13.



Rysunek 15.13. *Przy relatywnie dużym buforze odbiorczym znaczne ilości danych można przesłać nawet przed odczytaniem jakichkolwiek danych z sieci przez aplikację odbiorczą*

W powyższym przykładzie widzimy interesujące zachowanie protokołu TCP. Nawet zanim aplikacja skonsumowała **jakiekolwiek** dane, połączenie osiągnęło przepustowość ok. 1,3 MB. Ten stan rzeczy trwa mniej więcej do czasu 0,10. Potem przepustowość jest w zasadzie równa zero, dopóki odbiorca nie zacznie konsumować danych, co następuje dużo później (po czasie 20,0).

15.5.3. Syndrom głupiego okna (SWS)

Systemy sterowania przepływem oparte na strukturze okna, szczególnie te, w których nie używa się segmentów stałego rozmiaru (takie jak protokół TCP), mogą paść ofiarą patologii znanej jako **syndrom głupiego okna** (SWS, *Silly Window Syndrome*). Kiedy

to nastąpi, w trakcie połączenia wymieniane są małe segmenty danych zamiast segmentów pełnowymiarowych (patrz [RFC0813]). Prowadzi to do niepożądanego braku efektywności, ponieważ każdy segment posiada stosunkowo duży narzut — małą liczbę bajtów danych w stosunku do liczby bajtów nagłówków.

Syndrom SWS może być spowodowany przez którykolwiek z końców połączenia: odbiorca może wysyłać propozycje małych okien (zamiast poczekać, aż będzie można zaproponować większe okno), a nadawca może transmitować małe segmenty danych (zamiast poczekać na dodatkowe dane, aby wysłać większy segment). Prawidłowe unikanie syndromu głupiego okna wymaga od protokołu TCP wdrożenia reguł specjalnie przeznaczonych do tego celu, niezależnie od tego, czy działa on jako nadawca, czy jako odbiorca. Protokół TCP nigdy nie wie z wyprzedzeniem, jak zachowa się jego odpowiednik po drugiej stronie połączenia. Stosowane są następujące reguły.

- Kiedy TCP działa jako odbiorca, nie wysyła propozycji małych okien. Algorytm odbiorczy zdefiniowany w dokumencie [RFC1122] polega na niewysyłaniu segmentu z propozycją większego okna niż okno aktualnie proponowane (którego rozmiar może wynosić 0), dopóki okno nie może zostać powiększone albo o długość jednego pełnowymiarowego segmentu (tj. wartość MSS odbiorcy), albo o połowę przestrzeni bufora odbiorcy, w zależności od tego, która z tych wartości jest mniejsza. Zauważmy, że występują dwa przypadki, kiedy ta reguła może znaleźć zastosowanie: kiedy przestrzeń bufora staje się dostępna za przyczyną aplikacji konsumującej dane z sieci i kiedy protokół TCP musi odpowiedzieć na sondę okna.
- W przypadku protokołu wysyłającego dane nie są wysyłane małe segmenty, a decydowaniem, kiedy należy wykonać transmisję, zarządza algorytm Nagle'a. Nadawcy unikają syndromu SWS, nie transmitując segmentu, dopóki nie zostanie spełniony przynajmniej jeden z następujących warunków:
  - może zostać wysłany pełnowymiarowy segment (zawierający liczbę bajtów równą wartości MSS nadawcy),
  - protokół TCP może wysłać dane w ilości równej przynajmniej połowie maksymalnego rozmiaru okna, który druga strona kiedykolwiek zaproponowała w trakcie danego połączenia,
  - protokół TCP może wysłać wszystko, co ma do wysłania, i albo (i) potwierdzenie ACK nie jest aktualnie oczekiwane (tzn. nie mamy wcześniej wysłanych i niepotwierdzonych danych), albo (ii) algorytm Nagle'a jest wyłączony w tym połączeniu.

Warunek (a) jest najprostszy i w sposób bezpośredni unika problemu dużego narzutu w segmencie. Warunek (b) dotyczy hostów, które zawsze proponują bardzo małe okna, być może mniejsze niż rozmiar segmentu. Warunek (c) zapobiega wysyłaniu przez protokół TCP małych segmentów, w sytuacji gdy niepotwierdzone dane oczekują na przyjście potwierdzenia ACK i włączony jest algorytm Nagle'a. Jeśli wysyłająca dane aplikacja wykonuje małe zapisy (tzn. mniejsze niż rozmiar segmentu), warunek (c) pozwala uniknąć syndromu głupiego okna.

Te trzy warunki pozwalają również odpowiedzieć na następujące pytanie: jeśli algorytm Nagle'a nie pozwala na wysyłanie małych segmentów, w sytuacji gdy istnieją już wysłane, a niepotwierdzone dane, to jak małe muszą być te małe segmenty? Z warunku (a)

odczytujemy, że słowo „mały" oznacza liczbę bajtów mniejszą niż wartość SMSS (tzn. największy rozmiar pakietu, który nie przekracza wartości PMTU, ani parametru MSS odbiorcy). Warunek (b) wchodzi w grę tylko w przypadku starych, prymitywnych hostów lub kiedy używane jest małe okno proponowane z powodu ograniczonego rozmiaru bufora odbiorczego.

Warunek (b) punktu 2. wymaga, aby nadawca śledził maksymalny rozmiar okna proponowany przez drugą stronę. Jest to ze strony nadawcy próba odgadnięcia rozmiaru bufora odbiorczego drugiej strony. Chociaż rozmiar bufora odbiorczego mógł się zmniejszyć w trakcie już ustanowionego połączenia, w praktyce jest to rzadkie. Ponadto pamiętajmy, że protokół TCP unika sytuacji kurczenia się okna.

15.5.3.1. Przykład

Przedstawimy teraz szczegółowy przykład, w którym zobaczymy unikanie syndromu głupiego okna w działaniu; przykład ten obejmuje również użycie licznika czasu przetrwania (*persist timer*). Użyjemy programu `sock` z systemem Windows XP w roli hosta nadawczego, który wykonuje trzy 2048-bajtowe operacje zapisu do sieci, i systemem FreeBSD w charakterze odbiorcy. Polecenie w systemie nadawcy wygląda następująco:

```
C:\> sock -i -n 3 -w 2048 10.0.0.8 6666
```

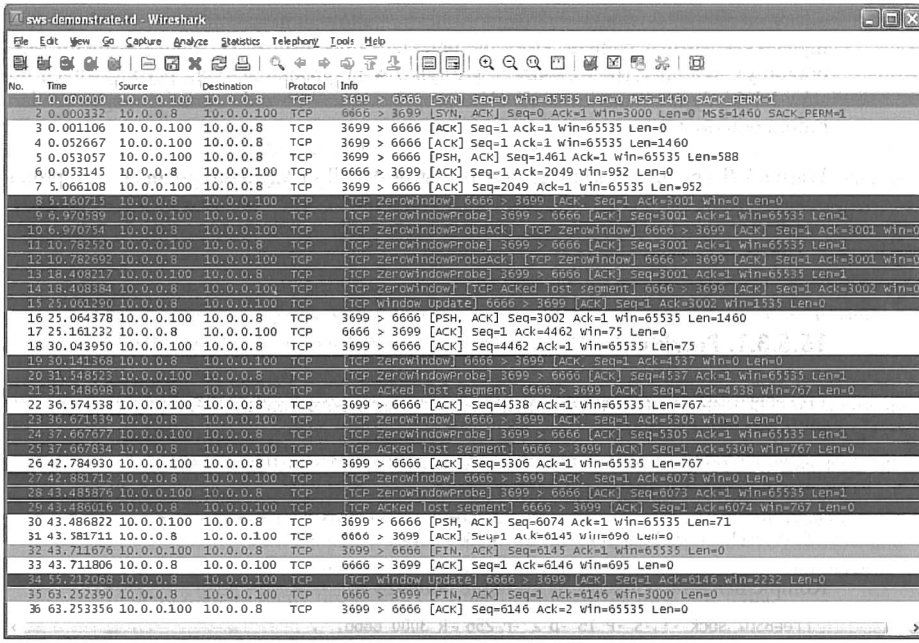
Komplementarne polecenie w systemie odbiorcy to:

```
FreeBSD% sock -i -s -P 15 -p 2 -r 256 -R 3000 6666
```

Ustala ono rozmiar bufora odbiorczego na 3000 bajtów, generuje zwłokę 15 s przed rozpoczęciem odczytywania danych z sieci, wstawia 2 s odstępu między kolejnymi odczytami i ustawia wielkość każdego odczytu na 256 bajtów. Powodem zastosowania początkowej pauzy jest stworzenie warunków do zapełnienia bufora odbiorcy, co w końcu zmusza nadawcę do wstrzymania transmisji. Sprawiając, że odbiorca wykonuje małe odczyty z sieci, spodziewamy się zobaczyć, w jaki sposób unika syndromu głupiego okna. Na rysunku 15.14 przedstawiamy ślad transmisji wyświetlony za pomocą programu Wireshark.

Na rysunku została wyświetlona treść całego połączenia. Długości pakietów są prezentowane przez podanie ilości bajtów ładunku użytecznego TCP zawartej w każdym segmencie. W czasie ustanawiania połączenia odbiorca proponuje okno o rozmiarze 3000 i wartość MSS wynoszącą 1460 bajtów. Nadawca wysyła 1460-bajtowy pakiet (pakiet 4.) w czasie 0,052 i kolejne 588 bajtów (pakiet 5.) w czasie 0,053. Suma rozmiarów tych pakietów jest równa 2048-bajtowej wielkości pojedynczego zapisu używanej przez aplikację. Pakiet 6. potwierdza oba pakiety danych otrzymane od nadawcy i przekazuje propozycję okna w wysokości 952 bajtów (3000–1460–588 = 952).

Okno 952-bajtowe (pakiet 6.) nie jest tak duże, jak pełna wartość MSS, więc algorytm Nagle'a działający u nadawcy zapobiega natychmiastowemu jego zapełnieniu. Zamiast tego mamy zwłokę 5 s przed podjęciem jakiegokolwiek dalszego działania. Nadawca czeka przez 5 s, dopóki nie wyzeruje się licznik czasu przetrwania, przed wysłaniem sondy okna. Zważywszy, że nadawca i tak wysyła pakiet, nadawczy protokół TCP dodaje do niego dozwolone 952 bajty, wypełniające dostępne okno. Zapełnienie okna zostaje potwierdzone przez propozycję zerowego okna zawartą w pakiecie 8.



Rysunek 15.14. *Ślad transferu wykonywanego w protokole TCP ilustrujący unikanie syndromu głupiego okna. Nadawca unika zapełnienia oferowanego okna w czasie 0,053, stosując procedurę unikania SWS po stronie nadawcy. Czeka z transmisją do czasu 5,066, kiedy to wysyła segment danych, występujący faktycznie w roli sondy okna. Unikanie syndromu SWS po stronie odbiorcy można dostrzec, patrząc na pakiet 14., który proponuje okno zerowe, mimo że odbiorca skonsumował jakieś dane*

Następnym zdarzeniem uwidocznionym w śladzie jest moment, kiedy protokół TCP wysyła sondę okna w czasie 6,970, ok. 2 s po otrzymaniu pierwszej propozycji zerowego okna. Sama sonda zawiera pojedynczy bajt danych i została oznaczona przez program Wireshark etykietą *TCP ZeroWindowProbe*, ale potwierdzenie ACK dla tego segmentu nie zwiększa wartości numeru ACK (program Wireshark oznacza je etykietą *TCP ZeroWindowProbeAck*), więc przesłany bajt danych nie został zachowany przez odbiorcę. Następna, 1-bajtowa sonda zostaje wygenerowana w czasie 10,782 (ok. 4 s później) i jeszcze jedna w czasie 18,408 (ok. 8 s później), co pokazuje charakterystyczne, rosnące wykładniczo odczekiwanie przed wysyłaniem kolejnych sond. Zauważmy, że w przypadku tej ostatniej sondy okna przesłany w niej pojedynczy bajt danych został potwierdzony przez odbiorcę.

W czasie 25,061, po tym jak aplikacja miała możliwość wykonania sześciu 256-bajtowych odczytów (w odstępach 2 s), aktualizacja okna wskazuje, że w buforze odbiorcy jest teraz 1535 bajtów (plus 1 dla numeru ACK) wolnego miejsca. Jest to „wystarczająco dużo", zgodnie z procedurą unikania syndromu SWS po stronie odbiorcy. Nadawca zaczyna zapełniać okno, zaczynając od wysłania 1460-bajtowego pakietu w czasie 25,064, skutkującego potwierdzeniem ACK w czasie 25,161 dla bajta 4462, zawierającym propozycję okna wielkości zaledwie 75 bajtów (pakiet 17.). Ta propozycja okna w widoczny spo-

sób narusza naszą regułę, że zaproponowana wielkość okna powinna być przynajmniej równa wartości MSS lub (w przypadku systemu FreeBSD) jednej czwartej całkowitego rozmiaru bufora. Powodem tego jest unikanie skurczenia się okna. W ostatniej aktualizacji okna (pakiet 15.) odbiorca proponuje prawą krawędź okna wskazującą bajt (3002+1535) = 4537. Jeśli aktualne potwierdzenie ACK (pakiet 17.) proponowałoby mniej niż 75 bajtów, czego wymagałaby procedura unikania syndromu SWS, prawa krawędź okna przesunęłaby się w lewo, tworząc sytuację, do której protokół TCP nie powinien dopuścić. W rezultacie 75-bajtowa propozycja okna przedstawia formę wyboru mniejszego zła: unikanie kurczenia się okna ma wyższy priorytet niż unikanie syndromu SWS.

Efekt unikania syndromu SWS widzimy raz jeszcze w opóźnieniu między pakietami 17. i 18., wynoszącym 5 s. Nadawca jest zmuszony do wysłania 75-bajtowego pakietu, a odbiorca odpowiada kolejną propozycją zerowego okna. Pakiet 20., który pojawia się sekundę później, jest następną sondą okna, której wynikiem jest okno o rozmiarze 767 bajtów. Kolejna runda procesu unikania syndromu SWS po stronie nadawcy skutkuje opóźnieniem 5 s; nadawca zapełnia okno, co znowu kończy się propozycją zerowego okna, i schemat się powtarza. Schemat ten zostaje w końcu przełamany, ponieważ nadawca nie ma więcej danych do wysłania. Pakiet 30. reprezentuje ostatnie wysłane dane, a połączenie zostaje ostatecznie zamknięte jakieś 20 s później (z powodu 2-sekundowych przerw między wszystkim kolejnymi odczytami wykonywanymi przez aplikację odbiorczą).

Aby lepiej zrozumieć związki między zachowaniem aplikacji, proponowanym oknem i unikaniem syndromu SWS, możemy przedstawić dynamikę połączenia w formie tabeli. W tabeli 15.1 prezentujemy działania po stronie nadawcy i po stronie odbiorcy, a także szacunkowe czasy, w których aplikacja wykonuje swoje odczyty.

Tabela 15.1. *Dynamika regulowania propozycji okna i działań aplikacji w celu uniknięcia syndromu głupiego okna*

Czas	Numer pakietu	Działanie			Bufor odbiorczy	
		Nadawca TCP	Odbiorca TCP	Aplikacja	Dane	Wolne miejsce
0,000	**1**	SYN			0	3000
0,000	**2**		SYN+ACK **1 okno 3000**		0	3000
0,001	**3**	ACK			0	3000
0,052	**4**	**1:1460 (1460)**			1460	1539
0,053	**5**	**1461:2048 (588)**			2048	952
0,053	**6**		ACK **2049 okno 952**		2048	952
5,066	**7**	**2049:3000 (952)**			3000	0
5,160	**8**		ACK **3001 okno 0**		3000	0
6,970	**9**	**3001 :3001 (1)**			3000	0
6,970	**10**		ACK **3001 okno 0**		3000	0
10,782	**11**	**3001:3001 (1)**			3000	0

Tabela 15.1. *Dynamika regulowania propozycji okna i działań aplikacji w celu uniknięcia syndromu głupiego okna — ciąg dalszy*

Czas	Numer pakietu	Działanie			Bufor odbiorczy	
		Nadawca TCP	Odbiorca TCP	Aplikacja	Dane	Wolne miejsce
10,782	**12**		ACK **3001 okno 0**		3000	0
15				odczyt 256 bajtów	2744	256
17				odczyt 256 bajtów	2488	512
18,408	**13**	**3001:3001 (1)**			2489	511
18,408	**14**		ACK **3002 okno 0**		2489	511
19				odczyt 256 bajtów	2233	767
21				odczyt 256 bajtów	1977	1023
23				odczyt 256 bajtów	1721	1279
25				odczyt 256 bajtów	1465	1535
25,061	**15**		ACK **3002 okno 1535**		1465	1535
25,064	**16**	**3002:4461 (1460)**			2925	75
25,161	**17**		ACK **4462 okno 75**		2925	75
27				odczyt 256 bajtów	2669	331
29				odczyt 256 bajtów	2413	587
30,043	**18**	**4462:4536 (75)**			2488	512
30,141	**19**		ACK **4537 okno 0**		2488	512
31				odczyt 256 bajtów	2232	768
31,548	**20**	**4537:4537 (1)**			2233	767
31,548	**21**		ACK **4538 okno 767**		2233	767
33				odczyt 256 bajtów	1977	1023
35				odczyt 256 bajtów	1721	1279
36,574	**22**	**4538:5304 (767)**			2488	512
36,671	**23**		ACK **5305 okno 0**		2488	512
37				odczyt 256 bajtów	2232	768
37,667	**24**	**5305:5305 (1)**			2233	767
37,667	**25**		ACK **5306 okno 767**		2233	767
39				odczyt 256 bajtów	1977	1023
41				odczyt 256 bajtów	1721	1279
42,784	**26**	**5306:6072 (767)**			2488	512

Tabela 15.1. *Dynamika regulowania propozycji okna i działań aplikacji w celu uniknięcia syndromu głupiego okna — ciąg dalszy*

Czas	Numer pakietu	Działanie			Bufor odbiorczy	
		Nadawca TCP	Odbiorca TCP	Aplikacja	Dane	Wolne miejsce
42,881	**27**		ACK **6073 okno 0**		2488	512
43				odczyt 256 bajtów	2232	768
43,485	**28**	**6073:6073 (1)**			2233	767
43,485	**29**		ACK **6074 okno 767**		2233	767
43,486	**30**	**6074:6144 (71)**			2304	696
43,581	**31**		ACK **6145 okno 696**		2304	696
43,711	**32**	**6145 (FIN)**				
43,711	**33**		ACK **6146 okno 695**		2305	695
45, 47, 49, 51, 53, 55				6 x odczyt 256 bajtów	769	2231
55,212	**34**		ACK **6146 okno 2232**		768	2232
57, 59, 61				3 x odczyt 256 bajtów	0	3000
63				odczyt 0 bajtów	0	3000
63,252	**35**		**FIN**		0	3000

Pierwsza kolumna w tabeli 15.1 przedstawia względną wartość momentu czasu dla każdej akcji pojawiającej się w śladzie transmisji. Czasy, prezentowane z dokładnością trzech cyfr po przecinku, zostały wzięte z danych wyjściowych programu Wireshark (patrz rysunek 15.14). Te czasy, które zostały podane bez żadnych cyfr po przecinku, są wydedukowanymi czasami działań podejmowanych na hoście odbiorczym, które nie są przedstawione w śladzie.

Ilość danych w buforze odbiorcy (kolumna w tabeli oznaczona tytułem „Dane") zwiększa się, kiedy przychodzą dane od nadawcy, a zmniejsza, kiedy aplikacja odczytuje (konsumuje) dane z bufora. Chcemy śledzić propozycje okna przesyłane przez odbiorcę do nadawcy oraz to, co te propozycje okna zawierają. To pozwoli zobaczyć, jak odbiorca unika syndromu SWS.

Jak już pisaliśmy we wcześniejszej analizie, pierwszą oznaką unikania syndromu SWS jest 5-sekundowe opóźnienie między segmentami 6. i 7., gdy nadawca unika podejmowania prób transmisji, mając do dyspozycji 952-bajtowe okno, aż zostanie do tego zmuszony. Kiedy dochodzi do transmisji, bufor odbiorcy zapełnia się, wywołując serię wymian propozycji zerowego okna i sond okna. Zauważamy rosnące wykładniczo odczekiwanie w sposobie działania licznika czasu przetrwania: sondy są wysyłane w czasach 6,970, 10,782 i 18,408. Kolejne odstępy czasu wynoszą mniej więcej 2, 4 i 8 s, od momentu kiedy nadawca po raz pierwszy otrzymał propozycję zerowego okna, w czasie 5,160.

Chociaż aplikacja odczytuje dane w czasach 15,0 i 17,0, do czasu 18,408 zdążyła przeczytać tylko 512 bajtów. Reguły unikania syndromu SWS po stronie odbiorcy dyktują, że nie powinna być dostarczona do nadawcy żadna aktualizacja okna, ponieważ 512 bajtów dostępnego miejsca w buforze nie stanowi ani połowy całkowitego rozmiaru bufora (3000 bajtów), ani co najmniej pojedynczej wartości parametru MSS (1460 bajtów). Wobec braku aktualizacji okna nadawca wysyła sondę okna w czasie 18,408 (segment 13.). Sonda zostaje odebrana, a zawarty w niej bajt danych zostaje zachowany przez odbiorcę, ponieważ dostępna jest pewna ilość miejsca w buforze, co jest potwierdzone przez zwiększenie się numeru ACK między segmentami 12. a 14.

Chociaż w buforze odbiorcy pozostaje 511 bajtów dostępnego miejsca, procedura unikania syndromu SWS po stronie odbiorcy włącza się kolejny raz. Implementacja unikania przez odbiorcę syndromu SWS w systemie FreeBSD wprowadza rozróżnienie między wyborem momentu wysłania aktualizacji okna a sposobem odpowiedzi na sondę okna. Chociaż, zgodnie z regułami zawartymi w dokumencie [RFC1122], system wysyła aktualizację okna tylko wtedy, gdy można zaproponować co najmniej połowę całkowitego rozmiaru bufora odbiorczego (lub wartość MSS), to jednak, odpowiadając na sondę okna, wysyła propozycję większego okna wtedy, kiedy okno osiągnie rozmiar MSS albo będzie możliwe zaproponowanie co najmniej **jednej czwartej** całkowitego rozmiaru bufora odbiorczego. W każdym przypadku 511 bajtów to mniej niż pełny rozmiar MSS, a także mniej niż 3000/4 = 750 bajtów, więc wyżej przedstawiona forma unikania syndromu SWS po stronie odbiorcy dyktuje, że propozycja okna zawarta w potwierdzeniu ACK dla segmentu 13. musi mieć wartość 0.

Po wykonaniu przez aplikację szóstego odczytu w czasie 25 bufor odbiorczy posiada 1535 bajtów wolnego miejsca (więcej niż połowę całkowitego rozmiaru 3000 bajtów), więc zostaje wysłana aktualizacja okna (segment 15.). Wtedy nadawca wysyła pełnowymiarowy segment (segment 16.), dla którego otrzymuje potwierdzenie ACK, lecz z propozycją okna w wysokości tylko 75 bajtów. W ciągu następnych 5 s ma miejsce unikanie syndromu SWS zarówno po stronie nadawcy, jak i po stronie odbiorcy. Nadawca czeka na propozycję większego okna, a aplikacja wykonuje odczyty w czasach 27 i 29, ale 587 bajtów wolnego miejsca w buforze odbiorczym to za mało, aby można było wysłać aktualizację okna. Dlatego nadawca musi czekać całe 5 s i w końcu wysyła swoje 75 bajtów, zmuszając znów odbiorcę do podjęcia procedury unikania syndromu SWS.

W sytuacji gdy odbiorca nie dostarcza aktualizacji okna, licznik czasu przetrwania nadawcy powoduje wysłanie sondy okna w czasie 31,548. W tym przypadku odbiorca, czyli system FreeBSD, odpowiada niezerowym oknem o rozmiarze 767 bajtów (jest to więcej niż jedna czwarta całkowitego rozmiaru bufora odbiorczego). Okno to nie jest jednak wystarczająco duże z punktu widzenia procedury unikania syndromu SWS używanej przez nadawcę, więc nadawca odczekuje kolejne 5 s i proces się powtarza. W końcu, w czasie 43,486, zostaje wysłane i potwierdzone ostatnie 71 bajtów. Potwierdzenie zawiera aktualizację okna w wysokości 696 bajtów. Chociaż jest to mniej niż jedna czwarta całkowitego rozmiaru bufora odbiorcy, propozycja okna nie zostaje zmniejszona do zera przez procedurę unikania syndromu SWS po stronie odbiorcy, aby zapobiec skurczeniu się okna.

Zamykanie połączenia rozpoczyna się od segmentu 32., który nie zawiera danych. Zostaje on natychmiast potwierdzony z propozycją okna wynoszącą 695 bajtów (segment FIN skonsumował jeden numer sekwencyjny u odbiorcy). Po wykonaniu przez aplikację

kolejnych sześciu odczytów odbiorca dostarcza aktualizację okna, ale nadawca skończył już wysyłanie danych i zachowuje ciszę. Aplikacja wykonuje następne cztery odczyty, z których trzy zwracają 256 bajtów, a ostatni, który nic nie zwraca, wskazuje na koniec przychodzących danych. W tym momencie odbiorca zamyka połączenie, powodując wysłanie segmentu FIN do nadawcy. Nadawca odpowiada końcowym potwierdzeniem ACK, dopełniając obustronne zamknięcie połączenia.

Ponieważ aplikacja wysyłająca dane inicjuje operację zamknięcia po wykonaniu swych trzech 2048-bajtowych zapisów, nadawcza strona połączenia przechodzi ze stanu ESTABLISHED do stanu FIN_WAIT_1 po wysłaniu segmentu 32. (patrz rozdział 13.). Następnie, po otrzymaniu segmentu 33. przechodzi do stanu FIN_WAIT_2. Chociaż otrzymuje aktualizację okna, w czasie gdy znajduje się w tym stanie, nie jest podejmowane żadne działanie, ponieważ nadawca już wysłał segment FIN, który został potwierdzony (w tym stanie nie ma licznika czasu). Pozostaje w tym stanie bezczynnie, dopóki nie otrzyma segmentu FIN od drugiej strony. To dlatego nie widzimy już żadnych dalszych transmisji wykonywanych przez nadawcę, aż do momentu odebrania przez niego segmentu FIN (segment 35.).

15.5.4. Duże bufory i automatyczne dostrajanie okna

Widzieliśmy w tym rozdziale, że aplikacja, która korzysta z bufora odbiorczego małego rozmiaru, może być skazana na znaczną degradację przepustowości w porównaniu do innych aplikacji używających protokołu TCP w podobnych warunkach. Jeśli nawet odbiorca wyznaczy odpowiednio duży bufor, to zbyt mały rozmiar bufora może określić nadawca, co i tak w końcu doprowadzi do złej wydajności. Problem ten stał się tak ważny, że obecnie wiele stosów TCP przydziela bufor odbiorczy niezależnie od rozmiaru określonego przez aplikację. W większości przypadków rozmiar określony przez aplikację jest faktycznie ignorowany, a system operacyjny używa albo dużej ustalonej wartości, albo rozmiar bufora jest obliczany dynamicznie.

W nowszych wersjach systemów Windows (Vista/7) i Linux obsługiwane jest **automatyczne dostrajanie** okna (*auto-tuning*; patrz [S98]). W przypadku automatycznego dostrajania ilość wysłanych i niepotwierdzonych danych w połączeniu (iloczyn przepustowości i opóźnienia w połączeniu, *bandwidth-delay product*, ważne pojęcie, które omawiamy w rozdziale 16.) jest szacowana w ciągły sposób, a rozmiar proponowanego okna jest tak ustalany, by był co najmniej tak duży, jak wartość tego oszacowania. Zaletą tego rozwiązania jest umożliwienie protokołowi TCP osiągnięcia maksymalnej dostępnej przepustowości (zależnej od dostępnej pojemności sieci) bez konieczności przedwczesnego przydzielania nadmiernie dużych buforów u nadawcy i u odbiorcy. W systemie Windows rozmiar bufora odbiorcy jest domyślnie regulowany automatycznie przez system operacyjny. Jednak zachowanie to może zostać zmodyfikowane przy użyciu polecenia netsh:

```
C:\> netsh interface tcp set heuristics disabled
C:\> netsh interface tcp set global autotuninglevel=X
```

gdzie *X* przyjmuje jedną z następujących wartości: disabled, highlyrestricted, restricted, normal lub experimental. Ustawienie to wpływa na sposób automatycznego wyboru proponowanego okna odbiorcy. W stanie disabled automatyczne dostrajanie nie jest używane, a rozmiar okna przyjmuje wartość domyślną. Tryby restricted i highlyrestricted

spowalniają zwiększanie się okna, a ustawienie `normal` pozwala oknu rosnąć stosunkowo szybko. Tryb `experimental` umożliwia bardzo agresywny wzrost rozmiaru okna, ale nie jest zalecany do normalnego użytku, ponieważ wiele witryn internetowych i niektóre zapory sieciowe z nim kolidują albo nie implementują opcji *Skalowanie okna* we właściwy sposób.

W systemie Linux 2.4 i późniejszych jego wersjach jest obsługiwane automatyczne dostrajanie okna po stronie nadawcy. W wersji 2.6.7 i późniejszych obsługiwane jest automatyczne dostrajanie okna zarówno po stronie odbiorcy, jak i po stronie nadawcy. Jednakże automatyczne dostrajanie podlega ograniczeniom nałożonym na rozmiary buforów. Wartości podane po znaku równości są wartościami domyślnymi (które mogą się różnić w zależności od poszczególnych dystrybucji Linuksa), które powinny zostać zwiększone, jeśli system ma być używany w środowiskach charakteryzujących się wysoką wartością iloczynu przepustowości i opóźnienia:

```
net.core.rmem_max = 131071
net.core.wmem_max = 131071
net.core.rmem_default = 110592
net.core.wmem_default = 110592
```

Dodatkowo parametry automatycznego dostrajania są określane przez następujące zmienne:

```
net.ipv4.tcp_rmem = 4096 87380 174760
net.ipv4.tcp_wmem = 4096 16384 131072
```

Każda z tych zmiennych zawiera trzy wartości, określające minimalny, domyślny i maksymalny rozmiar bufora, używane przez procedurę automatycznego dostrajania.

15.5.4.1. Przykład

Aby zademonstrować działanie automatycznego dostrajania po stronie odbiorcy, używamy systemu Windows XP jako nadawcy (skonfigurowanego do korzystania z dużych okien i opcji skalowania okna) i systemu Linux z jądrem 2.6.11 w roli odbiorcy, w którym zastosowano automatyczne dostrajanie okna. W systemie nadawcy wprowadzamy następujące polecenie:

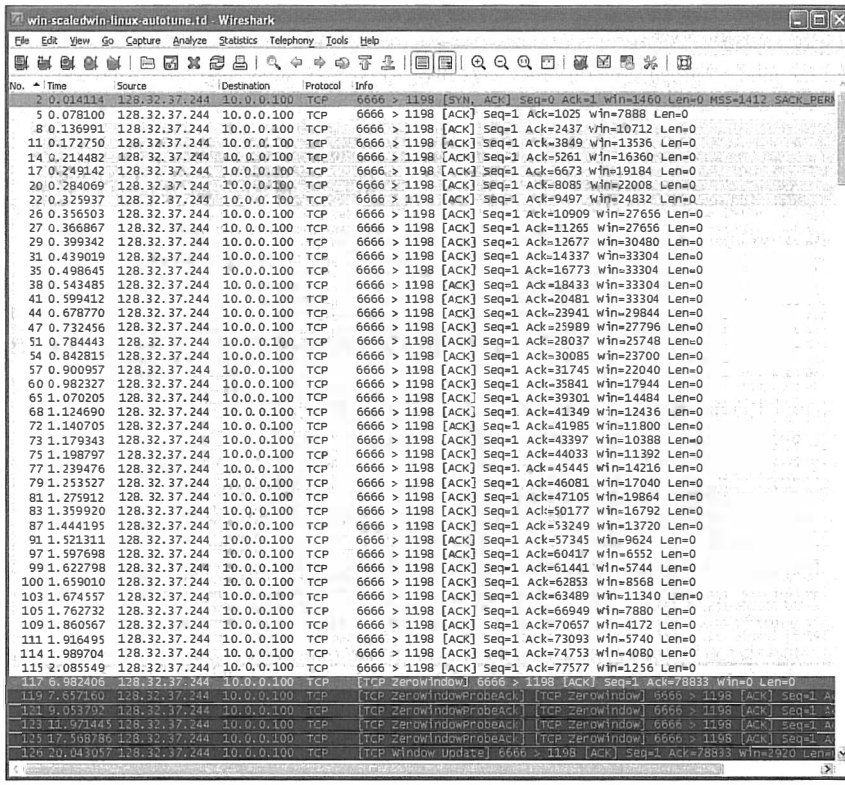
```
C:\> sock -n 512 -i 10.0.0.1 6666
```

Po stronie odbiorcy nie specyfikujemy żadnych ustawień dotyczących bufora odbiorczego, ale ustawiamy początkowe opóźnienie wielkości 20 s przed wykonaniem jakichkolwiek odczytów:

```
Linux% sock -i -s -v -P 20 6666
```

Aby zilustrować powiększanie się proponowanego przez odbiorcę okna, możemy użyć programu Wireshark, sortując wyświetlane pakiety według adresu odbiorcy (patrz rysunek 15.15). Podczas ustanawiania połączenia odbiorca rozpoczyna od początkowego rozmiaru okna wielkości 1460 bajtów i początkowej wartości MSS wynoszącej 1412 bajtów. Używa też skalowania okna z wartością przesunięcia równą 2 (co nie zostało pokazane), co pozwala uzyskać maksymalną wielkość okna użytkowego wynoszącą 256 kB. Widzimy, że po transmisji początkowych pakietów okno powiększa się, co odpowiada wzrostowi szybkości transmisji danych u nadawcy. Sterowanie szybkością transmisji

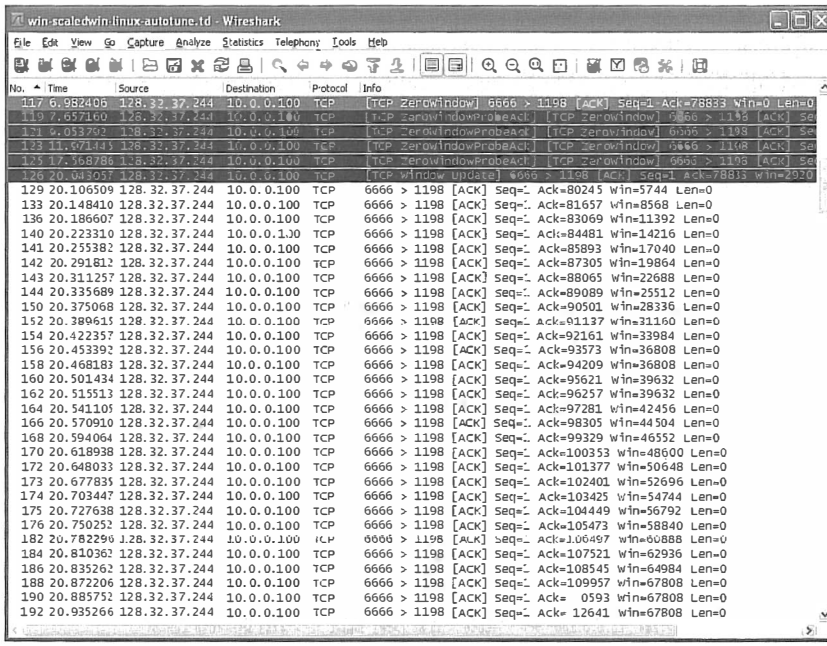
przez nadawcę dokładnie przeanalizujemy, gdy będziemy badać kontrolę przeciążenia w protokole TCP w rozdziale 16. Na razie musimy jedynie wiedzieć, że kiedy nadawca rozpoczyna transmitowanie danych, zwykle zaczyna od wysłania jednego pakietu, a potem zwiększa ilość danych wymagających potwierdzenia o jeden pakiet wielkości MSS po każdym odebranym potwierdzeniu ACK, które pokazuje postęp. Tak więc, na ogół wysyła dwa segmenty o rozmiarze MSS po każdym potwierdzeniu ACK, które odbierze.



Rysunek 15.15. *System Linux w roli odbiorcy wykonuje automatyczne dostrojenie okna, zwiększając rozmiar okna w miarę wzrostu ilości odebranych danych. Ponieważ aplikacja nie wykonuje odczytów przez 20 s, okno w końcu zamyka się*

Patrząc na schemat przejawiający się w propozycjach okna — 10712, 13536, 16360, 19184... — możemy dostrzec, że proponowane okno jest powiększane o podwójną wartość parametru MSS przy każdym kolejnym potwierdzeniu ACK, co naśladuje sposób działania systemu kontroli przeciążenia u nadawcy, jak zobaczymy w rozdziale 16. Pod warunkiem, że odbiorca posiada wystarczająco dużo pamięci, proponowane okno jest zawsze większe od tego, co wolno wysłać nadawcy, zgodnie z jego własnymi ograniczeniami wynikającymi z kontroli przeciążenia. Jest to najlepszy układ — odbiorca używa i proponuje minimalną ilość pamięci buforowej, która zapewnia nadawcy utrzymanie szybkości transmisji na maksymalnym możliwym poziomie.

Jeśli odbiorca wyczerpie wolne miejsce w swoich buforach, automatyczne dostrajanie przestaje dobrze funkcjonować. W omawianym przykładzie schemat zmian wielkości okna ulega odwróceniu przed czasem 0,678, po osiągnięciu maksimum wynoszącego 33 304 bajty. Rozmiar okna już nie rośnie, a zamiast tego bufor zapełnia się, w czasie gdy aplikacja pauzuje przed rozpoczęciem odczytów. Kiedy aplikacja rozpoczyna odczytywanie danych w czasie 20, rozmiar okna znowu zwiększa się i przekracza wartość osiągniętą poprzednio (patrz rysunek 15.16).



Rysunek 15.16. *W związku z tym, że aplikacja robi pauzę przed rozpoczęciem odczytywania danych, automatyczne dostrajanie przestaje dobrze funkcjonować z powodu zapełnienia się bufora. Kiedy aplikacja rozpoczyna odczytywanie, proponowane okno zwiększa się, przekraczając swój poprzedni rozmiar*

Propozycja zerowego okna (pakiet 117.) zmusza nadawcę do wykonania serii sond okna, generujących w odpowiedzi serię kolejnych propozycji zerowego okna. Po tym, jak aplikacja rozpoczyna odczytywanie danych w czasie 20,043, do nadawcy wysłana zostaje aktualizacja okna. Okno zaczyna znowu rosnąć, zwiększając się o podwójną wartość MSS wyrażoną w bajtach przy każdym kolejnym potwierdzeniu ACK. Ponieważ nadawca dalej wysyła dane, a odbiorca je konsumuje, odbiorca kontynuuje powiększanie proponowanego okna, aż zostaje osiągnięta wartość 67808, która jest najwyższą wartością proponowaną przez odbiorcę w czasie całego połączenia. Używana w przykładzie wersja Linuksa mierzy również czas między kolejnymi odczytami wykonywanymi przez aplikację i porównuje tę wartość z szacunkowym czasem transmisji „w obie strony". Jeśli szacunkowy czas RTT wzrasta, zwiększany jest także rozmiar bufora (który nie jest jednak zmniejszany, kiedy czas RTT maleje). To pomaga procedurze automa-

tycznego dostrajania utrzymywać wielkość proponowanego okna odbiorcy powyżej rozmiaru okna nadawcy, nawet kiedy wzrasta w połączeniu wartość iloczynu przepustowości i opóźnienia (*bandwidth-delay product*).

Problem używania przez aplikacje korzystające z protokołu TCP zbyt małych buforów okazał się znaczący, kiedy stały się dostępne szybsze połączenia w sieciach rozległych Internetu. W przypadku połączeń między odległymi regionami Stanów Zjednoczonych, gdzie czasy transmisji w obie strony wynoszą ok. 100 ms, użycie okna wielkości 64 kB przy transferze przez sieć o szybkości transmisji wynoszącej 1 GB/s ogranicza efektywną przepustowość połączenia TCP do ok. 640 kB/s zamiast wynikającego z obliczeń maksimum, wynoszącego ok. 130 MB (99 % niewykorzystanej szerokości pasma). Praktycznie rzecz biorąc, nie należy do rzadkości obserwowanie w tego typu sieciach 100-krotnego wzrostu przepustowości po zamianie protokołu TCP z buforami o ograniczonym rozmiarze na taki, który posiada większe bufory. Szczególne uznanie należy się projektowi Web100 (patrz [W100]). Dzięki niemu stworzono zestaw narzędzi i aktualizacji oprogramowania służących do maksymalizacji przepustowości, jaką aplikacja może uzyskać w różnych implementacjach protokołu TCP.

15.6. Mechanizm pilnych danych

W rozdziale 12. pisaliśmy, że nagłówek protokołu TCP posiada specjalne 1-bitowe pole *URG* do wskazania „pilnych danych". Aplikacja może oznaczyć dane jako pilne, wykorzystując specjalną opcję interfejsu API gniazd Berkeley (`MSG_OOB`), kiedy wykonuje operację zapisu, chociaż używanie koncepcji pilnych danych nie jest już dłużej zalecane (patrz [RFC6093]). Kiedy protokół TCP nadawcy otrzymuje takie żądanie zapisu, wchodzi w specjalny stan zwany **trybem pilnych danych** (*urgent mode*). Wchodząc w tryb pilnych danych, TCP rejestruje ostatni bajt, który aplikacja określiła jako pilne dane. Jest to wykorzystywane do ustawienia wartości pola *Wskaźnik pilnych danych* (*Urgent Pointer*) w każdym kolejnym nagłówku TCP generowanym przez nadawcę, dopóki aplikacja nie przestanie zapisywać pilnych danych i wszystkie numery sekwencyjne, aż do wskaźnika pilnych danych, nie zostaną potwierdzone przez odbiorcę. Według dokumentu [RFC6093] wskaźnik pilnych danych wskazuje numer sekwencyjny pierwszego bajta danych za ostatnim bajtem pilnych danych. Rozstrzyga to długotrwałą niejednoznaczność w różnych dokumentach RFC, które zawierały sprzeczne stwierdzenia dotyczące semantyki pola *Wskaźnik pilnych danych*. W jumbogramie IPv6 do wskazania, że koniec pilnych danych znajduje się na końcu obszaru danych TCP (patrz [RFC2675]), poza zakresem 64-kilobajtowego przesunięcia, które można określić przy użyciu konwencjonalnego 16-bitowego pola *Wskaźnik pilnych danych*, można umieścić w tym polu wartość 65 535.

Protokół odbiorczy TCP przechodzi w tryb pilnych danych, kiedy odbiera segment z ustawionym bitem *URG*. Aplikacja odbiorcza może wykryć, czy protokół TCP przeszedł do trybu pilnych danych, używając standardowej funkcji interfejsu gniazd (`select()`). Działanie mechanizmu pilnych danych stało się źródłem zamieszania, ponieważ interfejs API gniazd Berkeley i dokumentacja używają terminu dane **poza pasmem** (OOB, *Out-Of-Band*), chociaż w rzeczywistości protokół TCP nie implementuje żadnej prawdziwej możliwości transmisji OOB. Natomiast praktycznie wszystkie implementacje protokołu TCP dostarczają aplikacji ostatni bajt pilnych danych przy użyciu oddzielnego

parametru wywołania interfejsu API w systemie odbiorcy. Odbiorca musi użyć albo opcji MSG_OOB w celu pobrania tego specjalnego bajta w trybie „poza pasmem", albo opcji MSG_OOBINLINE, aby specjalny bajt pozostał w regularnym strumieniu danych (jest to obecnie wymagana metoda, przy założeniu, że mechanizm pilnych danych jest w ogóle używany).

15.6.1. Przykład

W celu uzyskania lepszego zrozumienia mechanizmu pilnych danych używamy systemu Mac OS X jako nadawcy oraz systemu Linux w roli odbiorcy, by pokazać, jak funkcjonuje tryb pilnych danych łącznie z tym, co się dzieje, gdy zachodzi zdarzenie zerowego okna. Żeby osiągnąć zamierzony cel, najpierw ograniczamy automatyczne dostrajanie okna odbiorczego w systemie Linux odbiorcy:

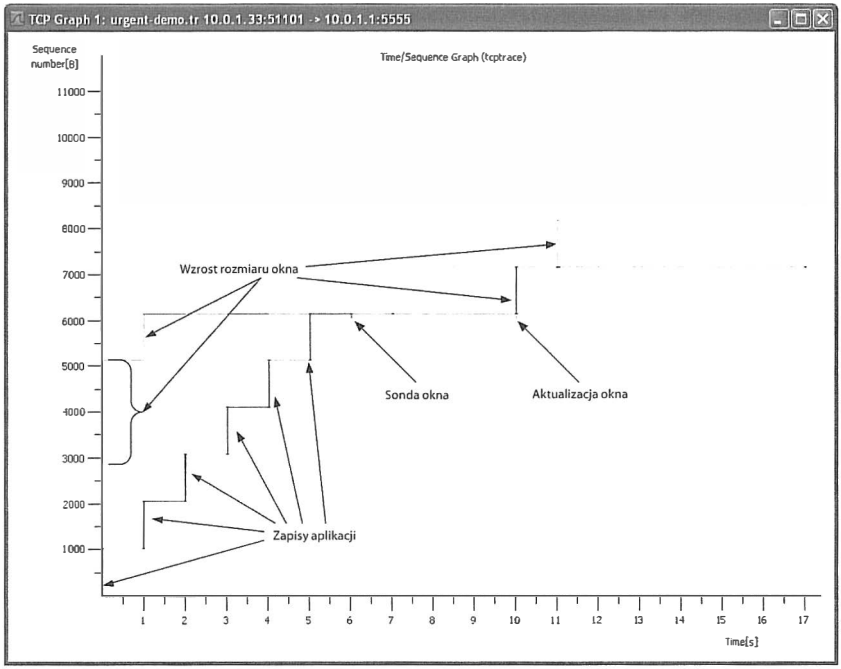
```
Linux# sysctl -w net.ipv4.tcp_rmem='4096 4096 174760'
Linux% sock -i -v -s -p 1 -P 10 5555
```

Pierwsze polecenie zapewnia, że żadna automatyczna regulacja okna odbiorczego nie przekroczy 4 kB. Pomoże nam to zobaczyć, co się dzieje, kiedy okno się zamyka. Drugie polecenie uruchamia serwer i nakazuje mu odczekanie 10 s przed wykonywaniem jakichkolwiek odczytów oraz zachowanie 1-sekundowych odstępów między wszystkimi kolejnymi operacjami odczytu. W systemie klienta wykonujemy następujące polecenie:

```
Mac% sock -i -n 7 -U 7 -p 1 -S 8192 10.0.1.1 5555
SO_SNDBUF = 8192
connected on 10.0.1.33.51101 to 10.0.1.1.5555
TCP_MAXSEG = 1448
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1024 bytes
wrote 1 byte of urgent data
wrote 1024 bytes
```

To polecenie tworzy klienta, który wykonuje siedem 1024-bajtowych zapisów w 1-sekundowych odstępach, ponadto wykonuje zapis 1 bajta pilnych danych przed ostatnią operacją zapisu. Bufor klienta jest wystarczająco duży (z rozmiarem ustalonym na 8192 bajty), aby aplikacja mogła natychmiast zakończyć działanie, ponieważ wszystkie wysyłane dane są buforowane przez nadawczy protokół TCP.

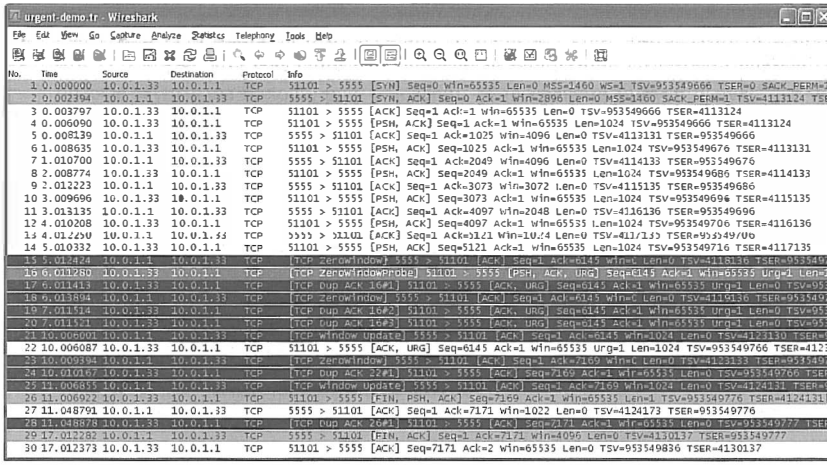
Na rysunku 15.17 możemy zobaczyć, że początkowa pozycja prawej krawędzi okna zaproponowana przez odbiorcę wynosi 2800 i zostaje szybko zwiększona do 5121. W czasie 1,0 aplikacja wykonuje operację zapisu i prawa krawędź okna przesuwa się mniej więcej na pozycję 6145. Od tego momentu okno odbiorcy już nie zwiększa się, ponieważ automatyczne dostrojenie zostało praktycznie wyłączone powyżej poziomu 4192 bajtów, a aplikacja odbiorcza nie wykonała żadnych odczytów. Do czasu 10,0 nadawca sonduje odbiorcę, ale okno dalej nie rośnie. W końcu, kiedy odbiorca zaczyna wykonywanie operacji odczytu po czasie 10,0, okno otwiera się, a nadawca kończy transfer. Pakiety wymieniane w trakcie połączenia zostały pokazane na rysunku 15.18.



Rysunek 15.17. *Po sześciu operacjach zapisu okno odbiorcy nie przesunęło się do przodu. Protokół TCP nadawcy wstrzymuje transmisję do czasu 10, kiedy okno otwiera się*

„Punkt wyjścia" dla trybu pilnych danych jest zdefiniowany jako suma pola *Numer sekwencyjny* i pola *Wskaźnik pilnych danych* w segmencie TCP. Tylko jeden „punkt" pilnych danych (przesunięcie w stosunku do numeru sekwencyjnego) jest utrzymywany dla pojedynczego połączenia TCP, więc pakiet przychodzący z istotnym polem *Wskaźnik pilnych danych* powoduje utratę informacji zawartej w dowolnym wcześniejszym wskaźniku pilnych danych. Segment 16. jest pierwszym segmentem zawierającym istotny wskaźnik pilnych danych, dający w efekcie względny numer sekwencyjny punktu wyjścia równy 6146. Zwróćmy uwagę, że ten numer sekwencyjny może nie mieścić się w segmencie zawierającym jego wskazanie, a znaleźć się w którymś z późniejszych segmentów. Ten przypadek ma miejsce np. w segmencie 17., który nie zawiera żadnych danych, ale za to zawiera wskaźnik pilnych danych (o wartości 1).

Jak już wcześniej wspominaliśmy, w przeszłości miało miejsce pewne zamieszanie dotyczące tego, czy punkt wyjścia wskazuje ostatni bajt pilnych danych, czy po nim następujący pierwszy bajt zwykłych danych. Według dokumentu [RFC1122] wskaźnik powinien wskazywać ostatni bajt pilnych danych. Jednak w zasadzie wszystkie implementacje protokołu TCP nie stosują się do tego określenia, więc dokument [RFC6093] uznaje ten fakt i tak zmienia różne określenia, aby wskaźnik wskazywał pierwszy bajt danych, które nie mają statusu pilnych. W prezentowanym przykładzie bajt z numerem sekwencyjnym 6145 jest pojedynczym bajtem pilnych danych wygenerowanym przez klienta programu `sock`, ale we wszystkich segmentach zawierających wartość 6145 w polu



Rysunek 15.18. *Całkowity transfer danych pokazujący propozycję zerowego okna od odbiorcy w czasie 5,012. Kiedy aplikacja wykonuje swoje kolejne zapisy, nadawczy protokół TCP wchodzi w tryb pilnych danych, w wyniku czego ustawiany jest bit URG, począwszy od segmentu z sondą okna, w czasie 6,0113, zawierającego jeden numer sekwencyjny. W czasie 7 aplikacja wykonuje swój końcowy zapis i zamyka się, generując dwa puste segmenty. Aktualizacja okna, w czasie 10,006, wznawia transfer danych. Propozycja zerowego okna, w czasie 10,009, znów zatrzymuje transfer, ale jednocześnie wskazuje, że można teraz wyjść z trybu pilnych danych, ponieważ wskaźnik pilnych danych został potwierdzony. Segment FIN w czasie 11,007 zawiera ostatni bajt danych*

numeru sekwencyjnego wskaźnik pilnych danych ma wartość 1. Stąd widzimy, że w tej implementacji protokołu TCP, jak w przypadku większości implementacji, punktem wyjścia jest numer sekwencyjny pierwszego bajta danych, które już nie są danymi pilnymi.

Jak możemy zobaczyć w tym przykładzie, protokół TCP przekazuje pilne dane wewnątrz strumienia danych (a nie „poza pasmem"). Jeśli aplikacja naprawdę wymaga oddzielnego kanału poza pasmem, najłatwiejszym sposobem, aby to uzyskać, jest otwarcie drugiego połączenia TCP. (Niektóre protokoły warstwy transportowej faktycznie dostarczają coś, co większość ludzi uważa za dane OOB: to oddzielna logicznie ścieżka danych korzystająca z tego samego połączenia, co normalna ścieżka danych. To nie jest jednak rozwiązanie stosowane przez protokół TCP).

15.7. Ataki dotyczące zarządzania oknem

Procedury zarządzania oknem protokołu TCP są obiektem różnych ataków, przed wszystkim form prowadzących do wyczerpania zasobów. W zasadzie proponowanie małego okna spowalnia transfer w TCP, wiążąc zasoby, takie jak pamięć, przez potencjalnie długi czas. Metoda ta została użyta jako forma ataku na złośliwy ruch sieciowy (np. robaki). Przykładowo atak typu LaBrea *tarpit*[1] (patrz [L01]) polega na utworzeniu wirtualnej

[1] Nazwa pochodzi od asfaltowych jeziorek znajdujących się w Los Angeles, które od czasu plejstocenu były pułapką dla zwierząt, a szczególnie dla dużych drapieżników, wabionych odgłosami już unieruchomionych osobników — *przyp. tłum.*

maszyny, która po wykonaniu trójetapowego uzgodnienia właściwego dla protokołu TCP nie robi niczego albo generuje minimalne odpowiedzi, co sprawia, że nadawczy protokół TCP ustawicznie spowalnia swoje działanie. Utrzymuje to TCP złośliwego nadawcy w stanie ciągłej zajętości i istotnie spowalnia propagację robaków. Więc *tarpit* to pułapka na atakujący ruch sieciowy.

Nowszy atak, który został opublikowany w 2009 roku w artykule [I09], jest oparty na znanej luce w bezpieczeństwie dotyczącej licznika czasu przetrwania. Używa on działającej po stronie klienta odmiany techniki *SYN cookies* (patrz rozdział 13.). Cały niezbędny stan połączenia może być w ten sposób przerzucony na maszynę ofiary, minimalizując ilość zasobów zużywanych na maszynie napastnika. Sam atak jest podobny do koncepcji LaBrea, z tym wyjątkiem, że skupia się specjalnie na liczniku czasu przetrwania. Wiele takich ataków może zostać przeprowadzonych na ten sam serwer, co może doprowadzić do wyczerpania zasobów (np. może zabraknąć pamięci systemowej). Radą na ten atak jest, jak to sugeruje się w dokumencie [C723308], umożliwienie jakiemuś innemu procesowi zakończenia połączeń TCP, kiedy pojawiają się symptomy wyczerpywania zasobów.

15.8. Podsumowanie

Dane interaktywne są zwykle transmitowane w segmentach mniejszych niż wartość parametru SMSS. Odbiorca może użyć opóźnionych potwierdzeń, aby sprawdzić, czy potwierdzenie można przesłać metodą „na barana" razem z danymi wracającymi do nadawcy. To często redukuje liczbę segmentów, szczególnie w przypadku ruchu interaktywnego, gdzie serwer przesyła echo znaków wpisywanych z klawiatury w systemie klienta. Może to jednak wprowadzać dodatkowe opóźnienie.

W połączeniach ze stosunkowo dużymi czasami transmisji w obie strony, takich jak sieci WAN, w celu zmniejszenia liczby małych segmentów jest używany algorytm Nagle'a. Algorytm ten ogranicza nadawcę, pozwalając mu na transmisję tylko jednego małego pakietu niepotwierdzonych danych w każdym momencie czasu. Podczas gdy pozwala to zredukować liczbę małych pakietów z dużym narzutem w sieci i zmniejszyć całkowitą liczbę pakietów transmitowanych w czasie połączenia, dodaje jednak opóźnienie, które jest czasem nie do zaakceptowania dla aplikacji. W dodatku interakcja między opóźnionymi potwierdzeniami ACK i algorytmem Nagle'a może prowadzić do niepożądanej formy przejściowego zakleszczenia. Z tego powodu algorytm Nagle'a może być wyłączany przez aplikacje i większość aplikacji interaktywnych korzysta z tej opcji.

Protokół TCP implementuje sterowanie przepływem, umieszczając propozycję okna w każdym potwierdzeniu ACK, które wysyła. Takie propozycje okna sygnalizują protokołowi TCP po drugiej stronie, jak dużo zostało miejsca w buforze punktu końcowego, który wysłał potwierdzenie ACK z propozycją okna. Maksymalna propozycja okna wynosi 65 535 bajtów, chyba że używana jest opcja skalowania okna protokołu TCP. W takim przypadku maksymalna propozycja okna może być dużo większa (do ok. 1 GB).

Propozycja okna może być tak mała, że aż równa 0 bajtów, co wskazuje, że bufor odbiorcy zapełnił się całkowicie. Kiedy do tego dochodzi, nadawca przestaje wysyłać dane, a zamiast tego zaczyna sondowanie zamkniętego okna, używając odstępów między

retransmisjami z uwzględnieniem systemu odczekiwania podobnego do stosowanego przy retransmisjach opartych na liczniku czasu (patrz rozdział 14.). Sondowanie zamkniętego okna trwa przez nieokreślony czas, dopóki nie zostanie zwrócone potwierdzenie ACK wskazujące większe okno lub odbiorca nie wyśle propozycji okna z własnej inicjatywy (tj. aktualizacji okna) na skutek pojawienia się dostępnego miejsca w buforze. Ta nieokreśloność w działaniu procedury została wykorzystana do utworzenia prowadzącego do wyczerpania zasobów ataku na protokół TCP.

Podczas opracowywania protokołu TCP zaobserwowano osobliwe zjawisko. Kiedy były proponowane małe okna, nadawca zapełniał je natychmiast. Ten sposób działania, który powoduje używanie w trakcie połączenia dużej liczby małych pakietów obarczonych dużym narzutem trwa, dopóki połączenie nie stanie się bezczynne i został nazwany „syndromem głupiego okna". W celu uniknięcia tego syndromu zostały stworzone techniki zastosowane w logice protokołu TCP zarówno po stronie nadawcy, jak i odbiorcy. Nadawca unika wysyłania małych segmentów, kiedy proponowane jest małe okno; odbiorcy starają się zawsze unikać wysyłania propozycji małego okna.

Rozmiar okna odbiorcy jest ograniczony rozmiarem bufora odbiorcy. W przeszłości aplikacjom, które nie określały wielkości swoich buforów odbiorczych, przydzielany był stosunkowo mały bufor, jaki powodował pogorszenie przepustowości w ścieżkach sieciowych charakteryzujących się szerokim pasmem i dużym opóźnieniem. W nowszych systemach operacyjnych automatyczne dostrajanie ustala rozmiar pamięci przydzielonej na bufor w sposób automatyczny i efektywny, co sprawia, że obawy związane ze zbyt małym buforem są w dużej mierze kwestią przeszłości.

15.9. Bibliografia

[C723308] *US-CERT Vulnerability Note VU#723308*, listopad 2009.

[F03] C. Fraleigh i in., *Packet-Level Traffic Measurements from the Sprint IP Backbone*, „IEEE Network Magazine", listopad/grudzień 2003.

[I09] F. Hantzis (ithilgore), *Exploiting TCP and the Persist Timer Infiniteness*, „Phrack", czerwiec 2009, nr 66(9).

[L01] T. Liston, *LaBrea: „Sticky" Honeypot and IDS*, *http://labrea.sourceforge.net*

[MM01] J. Mogul, G. Minshall, *Rethinking the TCP Nagle Algorithm*, „ACM Computer Communication Review", styczeń 2001, nr 31(6).

[MMQ] *http://technet.microsoft.com/en-us/library/cc730960.aspx*

[MMSV99] G. Minshall, J. Mogul, Y. Saito, B. Verghese, „Application Performance Pitfalls and TCP's Nagle Algorithm", *Proc. Workshop on Internet Server Performance*, maj 1999.

[P05] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, B. Tierney, „A First Look at Modern Enterprise Traffic", *Proc. Internet Measurement Conference*, październik 2005.

[RFC0813] D. Clark, *Window and Acknowledgment Strategy in TCP*, Internet RFC 0813, lipiec 1982.

[RFC0896] J. Nagle, *Congestion Control in IP/TCP Internetworks*, Internet RFC 0896, styczeń 1984.

[RFC1122] R. Braden, (red.), *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC2675] D. Borman, S. Deering, R. Hinden, *IPv6 Jumbograms*, Internet RFC 2675, sierpień 1999.

[RFC4251] T. Ylonen, C. Lonvick, *The Secure Shell (SSH) Protocol Architecture*, Internet RFC 4251, styczeń 2006.

[RFC4254] T. Ylonen, C. Lonvick, (red.), *The Secure Shell (SSH) Connection Protocol*, Internet RFC 4254, styczeń 2006.

[RFC6093] F. Gont, A. Yourtchenko, *On the Implementation of the TCP Urgent Mechanism*, Internet RFC 6093, styczeń 2011.

[S98] J. Semke, J. Mahdavi, M. Mathis, „Automatic TCP Buffer Tuning", *Proc. ACM SIGCOMM*, październik 1998.

[W100] *http://www.web100.org*

Rozdział 16.
Kontrola przeciążenia w protokole TCP

16.1. Wprowadzenie

W tym rozdziale zbadamy, jak protokół TCP podchodzi do kwestii **kontroli przeciążenia** (*congestion control*), która ma największe znaczenie w kontekście transferu danych masowych. Kontrola przeciążenia jest zespołem zachowań zdeterminowanych przez algorytmy, które implementuje każdy protokół TCP, kiedy usiłuje zapobiec zalaniu sieci przez zbyt wielki ładunek całkowitego wygenerowanego ruchu. Podstawowe podejście polega na spowodowaniu spowolnienia protokołu TCP, kiedy istnieje uzasadnione przypuszczenie, że sieć jest bliska przeciążenia (lub już jest tak przeciążona, że routery odrzucają pakiety). Główna trudność polega na dokładnym określeniu, kiedy i w jaki sposób protokół TCP powinien spowolnić swoje działanie i kiedy znów może przyśpieszyć.

TCP jest protokołem zaprojektowanym w celu zapewnienia niezawodnego dostarczania danych z jednego systemu do drugiego. W rozdziale 15. pisaliśmy, jak można skłonić nadawczy protokół TCP do spowolnienia swoich transmisji, w sytuacji gdy jego odpowiednik (protokół odbiorczy po drugiej stronie połączenia) nie może nadążyć z odbieraniem danych. Zadanie to jest wykonywane przez procedury protokołu TCP odpowiedzialne za **sterowanie przepływem**, a jego realizacja polega na dostosowaniu przez nadawcę swojej szybkości transmisji na podstawie proponowanej wartości pola *Rozmiar okna*, dostarczanej przez odbiorcę w potwierdzeniach ACK. W ten sposób nadawca otrzymuje zwrotnie bezpośrednią informację o stanie odbiorcy i może uniknąć przekroczenia pojemności jego buforów.

Zastanówmy się, co się stanie, kiedy sieć łącząca zbiór nadawców i odbiorców zostanie obarczona większą ilością ruchu, niż może obsłużyć. Albo nadawcy muszą wolniej wysyłać dane, albo w ostateczności sieć musi odrzucać część danych (albo wystąpi połączenie jednego i drugiego). Fakt ten wynika z najbardziej elementarnej obserwacji dotyczącej teorii kolejek w zastosowaniu do routera: jeśli nawet router może przechować pewną ilość danych, to jeśli tempo przychodzenia danych przewyższa w długim okresie czasu tempo ich wysyłania, ilość niezbędnej pamięci pośredniej będzie rosła w sposób nieograniczony. Wyrażając to prościej, jeśli router otrzymuje więcej danych w jednostce

czasu, niż może wysłać, musi przechować te dane. Jeśli ta sytuacja utrzymuje się, w końcu pamięć przechowująca dane wyczerpie się i router będzie zmuszony do usunięcia części danych.

Taka sytuacja, kiedy router jest zmuszony do odrzucania danych, ponieważ nie potrafi sprostać szybkości, z jaką przychodzi nowy ruch sieciowy, nazywa się **przeciążeniem** (*congestion*). Kiedy router znajduje się w tym stanie, mówimy o nim, że jest **przeciążony**, a nawet pojedyncze połączenie może doprowadzić jeden lub więcej routerów do stanu przeciążenia. Pozostawione bez reakcji przeciążenie może spowodować tak poważne zmniejszenie wydajności sieci, że staje się niezdatna do użytku. W ekstremalnych przypadkach mówi się o stanie **zapaści z powodu przeciążenia** (*congestion collapse*). Aby unikać tej sytuacji lub przynajmniej skutecznie na nią reagować w celu złagodzenia, każdy protokół TCP implementuje **procedury kontroli przeciążenia**. Różne wersje i warianty protokołu TCP (oraz systemy operacyjne stanowiące platformę stosu TCP/IP) mają nieco inne procedury i sposoby działania. W tym rozdziale omówimy większość najbardziej znanych.

16.1.1. Wykrywanie przeciążenia w protokole TCP

Jak już widzieliśmy, podstawowym mechanizmem dostępnym w protokole TCP, który służy do walki z utratą pakietów, jest retransmisja wywoływana albo przez wyzerowanie licznika czasu retransmisji, albo przez algorytm szybkiej retransmisji (patrz rozdział 14.). Rozważmy przez chwilę konsekwencje sytuacji, w której wiele połączeń TCP współdzielących ścieżkę w Internecie po prostu retransmituje coraz więcej pakietów, gdy tymczasem sieć jest w stanie zapaści z powodu przeciążenia. Jak można sobie wyobrazić, to tylko pogarsza sprawę. Da się porównać do gaszenia pożaru benzyną i jest czymś, czego powinno się unikać. W celu uporania się z przeciążeniem chcielibyśmy sprawić, aby nadawcze protokoły TCP spowolniły swoje transmisje, kiedy do niego doszło (albo wkrótce ma dojść) i, jeśli przeciążenie osłabło, wykryły i wykorzystały odpowiednią ilość nowej przepustowości, kiedy taka stanie się dostępna. W Internecie mogło to być dość trudne wyzwanie, jako że tradycyjnie nie było bezpośredniego sposobu, w jaki nadawczy protokół TCP mógłby poznać stan pośredniczących routerów. Innymi słowy, nie istnieje **bezpośrednia sygnalizacja** (*explicit signaling*) przeciążenia. Jeśli natomiast typowy protokół TCP ma jakoś reagować na przeciążenie, musi najpierw dojść do wniosku, że ma ono miejsce. Jest to zwykle realizowane przez wykrycie, że jeden lub więcej pakietów zostało utraconych. W protokole TCP poczyniono założenie, że utracony pakiet jest oznaką przeciążenia i że jest wymagana jakaś odpowiedź (tzn. spowolnienie działania w pewien sposób). Zobaczymy, że protokół TCP funkcjonował w ten sposób od późnych lat 80. ubiegłego wieku. Inne metody wykrywania przeciążenia, w tym pomiar opóźnienia i obsługiwane przez sieć jawne powiadomienie o przeciążeniu (ECN, *Explicit Congestion Notification*), które omawiamy w podrozdziale 16.11, umożliwiają protokołowi TCP dowiedzenie się o przeciążeniu, zanim stanie się tak poważne, że powoduje utratę pakietów. Analizujemy te podejścia po przestudiowaniu algorytmów „klasycznych".

W dzisiejszych sieciach przewodowych utrata pakietów jest głównie powodowana przez przeciążenie routerów lub przełączników. W sieciach bezprzewodowych znaczącą przyczyną utraty pakietów stają się błędy transmisji i odbioru. Rozstrzyganie, czy utrata nastąpiła w wyniku przeciążenia, czy z powodu błędów transmisji, stanowi aktywne pole badań od środka lat 90. ubiegłego wieku, kiedy to sieci bezprzewodowe zaczęły zdobywać szerokie zastosowanie.

W rozdziale 14. pisaliśmy, w jaki sposób protokół TCP może używać liczników czasu, potwierdzeń i selektywnych potwierdzeń w celu wykrywania i odzyskiwania utraconych pakietów. Kiedy zostanie wykryte, że pewne pakiety zostały utracone, obowiązkiem protokołu TCP jest ich ponowne wysłanie. Teraz jesteśmy zainteresowani tym, co jeszcze robi protokół TCP, kiedy zauważa utratę pakietu. Szczególnie interesuje nas, jak TCP interpretuje sygnał, że doszło do przeciążenia i że powinien spowolnić transmitowanie danych. Sposób, w jaki protokół zwalnia (i jak z powrotem przyśpiesza), i czas, kiedy to ma miejsce, stanowią główne tematy następnych punktów i podrozdziałów. Zaczniemy od klasycznego algorytmu używanego w nowym połączeniu w celu ustanowienia bazowej szybkości transmisji danych, a następnie omówimy inny klasyczny algorytm, który jest używany przez TCP w stanie ustalonym, kiedy protokół wykonuje duże transfery danych. Do naszej analizy włączymy również zalecane odmiany tych algorytmów i omówimy inne modyfikacje wprowadzone przez lata. Zbadamy również szczegółowo obszerny ślad działania protokołu. Zakończymy omówieniem pewnych kwestii dotyczących bezpieczeństwa, powiązanych z kontrolą przeciążenia stosowaną przez protokół TCP, i podsumujemy najważniejsze tematy. Dziedzina kontroli przeciążenia jest płodnym obszarem działalności dla badaczy zagadnień sieciowych (patrz [RFC6077]) i kilka nowych prac na ten temat ukazuje się każdego roku.

16.1.2. Spowolnienie nadawcy w protokole TCP

Szczegółem, którym musimy zająć się natychmiast, jest właśnie to, jak spowolnić nadawcę w TCP. W rozdziale 15. pisaliśmy, że do zasygnalizowania nadawcy konieczności dostosowania swojego okna do wielkości dostępnego miejsca w buforze odbiorcy służy pole *Rozmiar okna* w nagłówku TCP. Możemy pójść krok dalej i ustalić, że nadawca spowalnia transmisję, jeśli odbiorca jest zbyt wolny albo sieć jest zbyt wolna. Jest to osiągane przez wprowadzenie zmiennej sterującej oknem nadawcy, której wartość oparta jest na oszacowaniu pojemności sieci, i zapewnienie, że rozmiar okna nadawcy nigdy nie przekroczy mniejszego z tych dwóch limitów. W efekcie nadawczy protokół TCP wysyła dane z szybkością równą tej, którą może obsłużyć odbiorca lub sieć, zależnie od tego, która z nich jest mniejsza.

Ta nowa zmienna przechowująca oszacowanie dostępnej pojemności sieci nazywa się **oknem przeciążenia** (*congestion window*), czy w krótszym zapisie po prostu ***cwnd***. Faktyczne (użyteczne) okno nadawcy W jest więc wyznaczane jako minimum z proponowanego okna odbiorcy (*advertised window*) ***awnd*** i okna przeciążenia:

$$W = \min(cwnd, awnd)$$

Zgodnie z powyższą relacją nadawcy TCP nie wolno mieć więcej niż W niepotwierdzonych pakietów lub bajtów zalegających w sieci. Całkowita ilość danych, które nadawca wprowadził do sieci, a dla których nie otrzymał jeszcze potwierdzenia, jest czasem nazywana **rozmiarem przelotu** (*flight size*), który jest zawsze mniejszy lub równy W. Wartość zmiennej W może być wyrażona albo w pakietach, albo w bajtach.

Kiedy protokół TCP nie wykorzystuje potwierdzenia selektywnego, ograniczenie nałożone na wartość W oznacza, że nadawcy nie wolno wysłać segmentu z numerem sekwencyjnym większym niż suma największego potwierdzonego numeru sekwencyjnego i wartości W. Nadawca TCP używający opcji SACK traktuje wartość W w nieco inny sposób, bo używa jej jako całkowitego limitu rozmiaru przelotu.

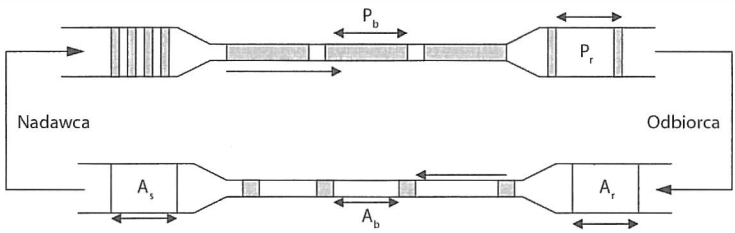
To wszystko wydaje się logiczne, ale jest dalekie od wyczerpania tematu. Ponieważ zarówno stan sieci, jak i stan odbiorcy zmieniają się z biegiem czasu, również wartości obu zmiennych, *awnd* i *cwnd*, zmieniają się w czasie. W dodatku, z powodu braku jawnych sygnałów (patrz poprzedni punkt), „prawidłowa" wartość *cwnd* nie jest na ogół bezpośrednio dostępna dla nadawczego protokołu TCP. Tak więc, wszystkie wartości, czyli *W*, *cwnd* i *awnd*, muszą być ustalane empirycznie i dynamicznie modyfikowane. W dodatku, jak już wcześniej powiedzieliśmy, nie chcemy, aby wartość *W* była zbyt duża albo zbyt mała — chcemy, aby w przybliżeniu była równa **iloczynowi przepustowości i opóźnienia** (BDP, *Bandwidth-Delay Product*) ścieżki sieciowej, określanemu również mianem **optymalnego rozmiaru okna**. Jest to ilość danych, która może zostać przechowana w sieci w drodze do odbiorcy. Jest ona równa iloczynowi czasu RTT i pojemności najgorszego pod tym względem łącza (tzw. „wąskiego gardła") w ścieżce od nadawcy do odbiorcy. Generalnie strategia nadawania polega na utrzymywaniu sieci w stanie aktywności przez taką organizację transmisji, która zapewni obecność w sieci ilości danych co najmniej równej parametrowi BDP. Jednak użycie limitu danych możliwych do wysłania przed otrzymaniem potwierdzenia ich odbioru, który znacznie przekracza wartość BDP, jest zazwyczaj niepożądane, gdyż może prowadzić do niepotrzebnych opóźnień (patrz podrozdział 16.10). W przypadku Internetu określenie wartości BDP dla połączenia może stanowić wyzwanie, gdy weźmiemy pod uwagę fakt, że trasy, opóźnienie i poziom multipleksowania statystycznego (tzn. współdzielenia pojemności) zmieniają się w czasie.

Chociaż naszym głównym obszarem zainteresowania jest obsługa przeciążenia przez nadawczy protokół TCP, zostały wykonane prace nad obsługą takich przypadków, w których przeciążenie pojawia się w odwrotnej ścieżce, z powodu potwierdzeń ACK. Dokument [RFC5690] wprowadza metodę informowania odbiorczego protokołu TCP o współczynniku potwierdzeń, którego powinien używać (tzn. ile pakietów powinien otrzymać przed wysłaniem potwierdzenia ACK).

16.2. Algorytmy klasyczne

Kiedy nowe połączenie TCP rozpoczyna działanie, zwykle nie wiadomo, jaka powinna być początkowa wartość *cwnd*, tak jak nie wiadomo, jak duża pojemność sieci jest dostępna dla wysyłanych danych. (Istnieją pewne wyjątki, takie jak systemy, które przechowują ustalone wcześniej wartości dotyczące wydajności, nazwane w rozdziale 14. miernikami punktu docelowego). Protokół TCP poznaje wartość *awnd* po jednej wymianie pakietów z odbiorcą, ale przy braku jakiejkolwiek bezpośredniej sygnalizacji jedynym oczywistym sposobem poznania dobrej wartości dla parametru *cwnd* są próby wysyłania danych z coraz to większą szybkością, aż do wystąpienia utraty pakietu (lub pojawienia się innej oznaki przeciążenia). Można to zrealizować albo przez wysyłanie danych od razu z maksymalną możliwą szybkością (ograniczoną przez wartość *awnd*), albo można rozpocząć wolniej. Z powodu negatywnych skutków dla wydajności innych połączeń współużytkujących tę samą ścieżkę sieciową, które mogłyby wystąpić przy rozpoczęciu od pełnej szybkości transmisji, protokół TCP na ogół używa specjalnego algorytmu umożliwiającego uniknięcie tak szybkiego startu w fazie wstępnej połączenia, przed osiągnięciem stanu ustalonego. Inny algorytm ma zastosowanie, kiedy zostanie osiągnięty stan ustalony.

Działanie kontroli przeciążenia protokołu TCP po stronie nadawcy jest sterowane lub „taktowane" przez odbiór potwierdzeń ACK. Jeśli protokół TCP działa w stanie ustalonym (z odpowiednią wartością *cwnd*), odebranie potwierdzenia ACK wskazuje, że jeden lub więcej pakietów usunięto z sieci i w konsekwencji powstała okazja do wysłania następnych. Idąc tym tokiem rozumowania, stwierdzamy, że obsługa przeciążenia protokołu TCP zmierza do **zachowania pakietów** w sieci (patrz rysunek 16.1). Termin „zachowanie" jest tu używany w znaczeniu występującym w fizyce — pewna wielkość (np. moment pędu, energia) wchodząca do systemu nie znika tak sobie, ani nie pojawia się, ale jest w nim stale odnajdywana, o ile metoda bilansowania jest właściwa.



Rysunek 16.1. *Kontrola przeciążenia w protokole TCP działa na zasadzie zachowania pakietów. Pakiety (P_b) są „rozciągnięte" w czasie, kiedy są przesyłane od nadawcy do odbiorcy przez łącza o ograniczonej pojemności. W miarę jak pakiety docierają do odbiorcy poprzedzielane odstępami czasu (P_r), generowane są potwierdzenia ACK (A_r), które wracają do nadawcy. Potwierdzenia ACK płynące od odbiorcy do nadawcy zostają rozłożone w czasie (A_b) w związku z odstępami między pakietami, które dotarły do odbiorcy. Kiedy potwierdzenia docierają do nadawcy (A_s), ich przyjścia stanowią dla nadawcy sygnał (czyli „takt zegara ACK"), że jest pora na wysłanie kolejnych danych. O całym systemie, będącym w stanie ustalonym, mówi się, że jest „samotaktujący" (*self-clocked*). Rysunek jest adaptacją rysunku pochodzącą z pracy [J88] i skopiowaną z wykładu S. Seshana opublikowanego w CMU Lecture Notes z datą 22 marzec 2005 roku*

Myśl ta została zilustrowana na rysunku 16.1. Nazwijmy górny i dolny obiekt przedstawiony na rysunku „lejkami". Górny lejek zawiera (większe) pakiety danych, przepływające wzdłuż ścieżki od nadawcy do odbiorcy. Względnie mała szerokość lejka pozwala zobrazować, jak pakiety „rozciągają się" w czasie, kiedy przepływają przez stosunkowo wolne łącze. Końce lejków (po stronie nadawcy i po stronie odbiorcy) pokazują kolejki, w których pakiety są trzymane przed podróżą wzdłuż ścieżki lub po tej podróży. Dolny lejek zawiera potwierdzenia ACK wysłane przez nadawcę z powrotem do odbiorcy, które odpowiadają pakietom danych znajdującym się w górnym lejku. Kiedy system pracuje efektywnie w stanie ustalonym, nie ma zbitych w grupę pakietów ani w górnym, ani w dolnym lejku. Poza tym nie ma znaczących dodatkowych odstępów między pakietami w górnym lejku. Zwróćmy uwagę, że przyjście potwierdzenia ACK do nadawcy „uwalnia" następny pakiet danych do przesłania przez górny lejek i dzieje się to akurat we właściwym momencie (tzn. wtedy, gdy sieć jest zdolna do przyjęcia kolejnego pakietu). To powiązanie czasowe jest czasem nazywane samotaktowaniem, ponieważ przyjście potwierdzenia ACK, zwane taktem zegara ACK, powoduje, że system podejmuje czynność wysłania kolejnego pakietu.

Teraz zajmiemy się dwoma głównymi algorytmami protokołu TCP: powolnym startem i unikaniem przeciążenia. Algorytmy te, oparte na zasadach zachowania pakietów i taktowania potwierdzeniami ACK, zostały po raz pierwszy w sposób formalny opisane w klasycznym referacie Jacobsona [J88].

Aktualizacja algorytmu unikania przeciążenia została przedstawiona przez Jacobsona dwa lata później (patrz [J90]). Te dwa algorytmy nie działają w tym samym czasie — protokół TCP wykonuje tylko jeden z nich w dowolnym momencie czasu, ale może się między nimi przełączać. Zbadamy je teraz bardziej szczegółowo i sprawdzimy, co decyduje o tym, kiedy każdy z nich jest używany. Przyjrzymy się również, jak te algorytmy zostały zmodyfikowane i rozszerzone od czasu ich pierwotnej implementacji. Każde połączenie TCP może wykonywać je samodzielnie.

16.2.1. Powolny start

Algorytm powolnego startu jest wykonywany, kiedy tworzone jest nowe połączenie TCP albo kiedy została wykryta utrata pakietu z powodu upłynięcia czasu oczekiwania na retransmisję (RTO). Może być również wywołany po pewnym okresie pozostawania w stanie bezczynności nadawczego protokołu TCP. Celem powolnego startu jest pomoc protokołowi TCP w znalezieniu wartości *cwnd* przed sondowaniem dostępności większej przepustowości przy użyciu algorytmu unikania przeciążenia oraz ustanowienie zegara ACK. Zazwyczaj protokół TCP rozpoczyna nowe połączenie, wykonując powolny start, w końcu traci pakiet, a potem przechodzi do działania właściwego dla stanu ustalonego, używając algorytmu unikania przeciążenia (patrz punkt 16.2.2). W tym miejscu zacytujmy dokument [RFC5681]:

> *Rozpoczęcie transmisji do sieci w nieznanych uwarunkowaniach wymaga, aby protokół TCP powoli sondował sieć, by ustalić dostępną pojemność, w celu uniknięcia zapchania sieci przez nieodpowiednio duży, nagły przypływ danych. Algorytm powolnego startu jest używany do tego celu na początku transferu lub po naprawieniu utraty danych wykrytej przez licznik czasu retransmisji.*

Protokół TCP rozpoczyna od powolnego startu, wysyłając pewną liczbę segmentów (po wymianie żądań SYN), zwaną **oknem początkowym** (IW, *Initial Window*). Pierwotnie wartość IW była równa pojedynczej wartości SMSS, chociaż od opublikowania dokumentu [RFC5681] dopuszczalne są większe wartości tego parametru. Sposób określania wartości IW wyrażają następujące wzory:

$$IW = 2 \cdot (SMSS) \text{ i nie więcej niż 2 segmenty } (\text{jeśli } SMSS > 2190 \text{ bajtów})$$
$$IW = 3 \cdot (SMSS) \text{ i nie więcej niż 3 segmenty } (\text{jeśli } 2190 \geq SMSS > 1095 \text{ bajtów})$$
$$IW = 4 \cdot (SMSS) \text{ i nie więcej niż 4 segmenty } (\text{w innym przypadku})$$

Chociaż podane wyżej określenie wartości IW pozwala na zawarcie kilku pakietów (np. trzech lub czterech) w oknie początkowym, dla uproszczenia omówimy przypadek, gdzie IW = 1 SMSS. Ponadto protokół TCP dopiero rozpoczynający połączenie określa wartość *cwnd* = 1 SMSS, co oznacza, że początkowe użyteczne okno *W* jest również równe SMSS. Zwróćmy uwagę, że w większości przypadków parametr SMSS jest równy mniejszej z wartości: MSS odbiorcy i MTU ścieżki (minus rozmiary nagłówków).

Zakładając, że nie są tracone żadne pakiety i każdy pakiet powoduje wysłanie w odpowiedzi potwierdzenia ACK, zostaje zwrócone potwierdzenie ACK dla pierwszego segmentu, co pozwala nadawczemu protokołowi TCP na wysłanie następnego segmentu. Jednak powolny start działa, zwiększając wartość *cwnd* o min(*N*, SMSS) dla każdego

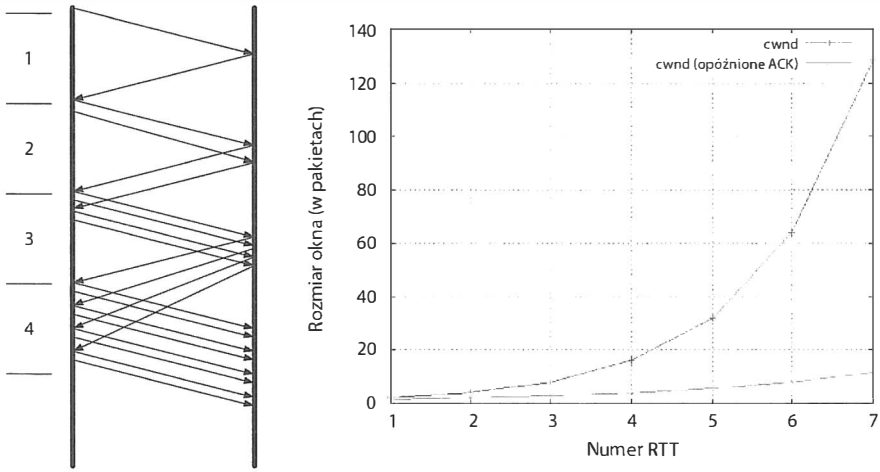
odebranego „dobrego" potwierdzenia ACK, gdzie N oznacza liczbę wcześniej niepotwierdzonych bajtów, które zostały potwierdzone dopiero przez teraz otrzymane, dobre potwierdzenie ACK. Dobre potwierdzenie ACK to takie, które zwraca numer ACK większy od wszystkich dotychczas odebranych.

Liczba bajtów potwierdzonych przez ACK jest wykorzystywana w obsłudze algorytmu *Appropriate Byte Counting* (odpowiednie zliczanie bajtów; ABC) opisanego w eksperymentalnej specyfikacji [RFC3465], zalecanego przez dokument [RFC5681]. Algorytm ten może być użyty do odparcia ataku z „podziałem potwierdzeń ACK", opisanym w podrozdziale 16.12, w którym wykorzystuje się dużą ilość małych potwierdzeń ACK w próbie spowodowania, aby nadawczy protokół TCP wysyłał dane szybciej niż normalnie. Linux do ustalenia, czy algorytm ABC jest włączony (domyślnie nie jest), używa zmiennej konfiguracyjnej systemu `net.ipv4.tcp_abc` przyjmującej wartości logiczne. W nowszych wersjach systemu Windows algorytm ABC jest domyślnie włączony.

Tak więc, po potwierdzeniu przez ACK jednego segmentu, wartość *cwnd* jest zazwyczaj zwiększana do 2 i wysyłane są dwa segmenty. Jeżeli każdy z nich powoduje zwrócenie nowego dobrego potwierdzenia ACK, 2 zwiększa się do 4, 4 do 8 itd. Ogólnie rzecz biorąc, przy założeniu braku strat i potwierdzania przez ACK każdego pakietu, wartość W po k obustronnych wymianach wynosi $W = 2^k$. Ujmując to z odwrotnej strony, możemy powiedzieć, że osiągnięcie okna operacyjnego o rozmiarze W wymaga $k = \log_2 W$ czasów RTT. Ten wzrost okna wydaje się całkiem szybki (zwiększa się ono zgodnie z funkcją wykładniczą), ale wciąż jest wolniejszy w stosunku do tego, co zrobiłby protokół TCP, jeśli by mu pozwolono wysłać od razu okno pakietów równe rozmiarem oknu proponowanemu przez odbiorcę. (Pamiętajmy, że wartość W nigdy nie może przekroczyć *awnd*).

Jeśli wyobrazimy sobie takie połączenie TCP, w którym proponowane okno odbiorcy jest bardzo duże (powiedzmy, nieskończenie duże), *cwnd* jest głównym czynnikiem zarządzającym szybkością wysyłania (pod warunkiem, że nadawca ma coś do wysłania). Jak widzieliśmy, wartość ta rośnie w tempie wykładniczym z każdym kolejnym czasem RTT połączenia. Tak więc, w końcu wartość *cwnd* (a zatem i W) może stać się tak duża, że odpowiadające jej okno wysłanych pakietów zaleje sieć (pamiętajmy, że przepływność w protokole TCP jest proporcjonalna do W/RTT). Kiedy zdarzy się coś takiego, wartość *cwnd* zostaje znacznie zmniejszona (do połowy poprzedniej wartości). W dodatku jest to moment, w którym protokół TCP przełącza swoje działanie z powolnego startu na unikanie przeciążenia. Moment przełączenia jest określony relacją między wartością *cwnd* i wartością zwaną **progiem powolnego startu** (*slow start threshold* lub *ssthresh*).

Na rysunku 16.2 (w lewej części) pokazujemy działanie powolnego startu. Czas jest przedstawiony w jednostkach równych czasowi RTT połączenia. Przy założeniu, że połączenie rozpoczyna działanie od wysłania jednego pakietu (od góry), zwracane jest jedno potwierdzenie ACK, które pozwala na wysłanie dwóch pakietów w drugim czasie RTT. Pakiety te powodują zwrócenie dwóch potwierdzeń ACK. Protokół nadawczy TCP zwiększa wartość *cwnd* o jeden segment dla każdego zwróconego ACK i taki proces jest kontynuowany. Wykładniczy wzrost wartości *cwnd* w funkcji czasu jest zilustrowany po prawej stronie rysunku. Druga linia wykresu pokazuje, jak wzrasta *cwnd*, kiedy potwierdzany jest co drugi pakiet, co występuje często, gdy używane są opóźnione potwierdzenia ACK. W tym przypadku wzrost rozmiaru okna jest nadal wykładniczy, ale nie tak szybki.



Rysunek 16.2. *Działanie klasycznego algorytmu powolnego startu. W prostym przypadku, gdy potwierdzenia ACK nie są opóźniane, każde przychodzące dobre potwierdzenie ACK pozwala nadawcy wprowadzić do sieci dwa nowe pakiety (po lewej stronie). Prowadzi to do wykładniczego wzrostu rozmiaru okna nadawcy w funkcji czasu (górny wykres po prawej stronie rysunku). Kiedy potwierdzenia ACK są opóźniane, tak jak wtedy, gdy potwierdzenie ACK jest generowane dla co drugiego pakietu, wzrost jest nadal wykładniczy, ale wolniejszy (dolny wykres po prawej stronie rysunku)*

Z tego powodu niektóre implementacje protokołu TCP stosują opóźnianie potwierdzeń dopiero wtedy, gdy połączenie zakończy proces powolnego startu. W systemie Linux nazywa się to trybem szybkich potwierdzeń (*quick acknowledgements*) lub krótko trybem *quickack* i stało się częścią podstawowego stosu protokołów TCP/IP od wersji 2.4.4 jądra systemu.

16.2.2. Unikanie przeciążenia

Opisany poprzednio algorytm powolnego startu jest używany przy inicjowaniu przepływu danych przez połączenie lub po zdarzeniu utraty pakietu zasygnalizowanym przez przeterminowanie. Zwiększa on dość szybko wartość *cwnd* i pomaga ustanowić wartość progu *ssthresh*. Kiedy te cele zostaną osiągnięte, nadal istnieje możliwość, że zwiększy się pojemność sieci dostępna dla połączenia. Jeśli taka dodatkowa pojemność zostałaby niezwłocznie wykorzystana przez duże, nagłe wzrosty ruchu sieciowego, inne połączenia TCP, których pakiety dzielą te same kolejki w routerach, doświadczyłyby prawdopodobnie znacznej utraty pakietów, co doprowadziłoby do ogólnej niestabilności w sieci, ponieważ doszłoby do sytuacji, w której wiele połączeń jednocześnie doświadcza utraty pakietów i reaguje na nią retransmisjami.

By podjąć próby znalezienia dodatkowej pojemności, która może stać się dostępna, ale nie robić tego w zbyt agresywny sposób, protokół TCP implementuje algorytm unikania przeciążeń (*congestion avoidance*), który poszukuje dodatkowej pojemności przez zwiększanie wartości *cwnd* o mniej więcej jeden segment dla każdej porcji danych o wielkości okna, która została z sukcesem przetransmitowana od nadawcy do odbiorcy. Ten algorytm generuje dużo wolniejsze tempo wzrostu niż powolny start: w przybliżeniu *cwnd*

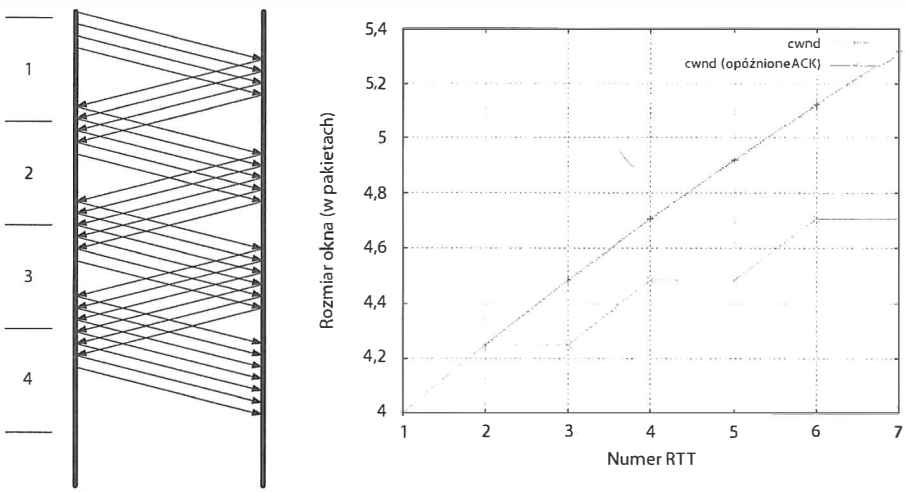
wzrasta liniowo w zależności od czasu, w przeciwieństwie do wykładniczego wzrostu w algorytmie powolnego startu. Ujmijmy to bardziej precyzyjnie: wartość *cwnd* jest aktualizowana po każdym odebranym, niezduplikowanym potwierdzeniu ACK w następujący sposób:

$$\text{cwnd}_{t+1} = \text{cwnd}_t + \text{SMSS} \cdot \text{SMSS} / \text{cwnd}_t$$

Załóżmy w powyższej zależności, że wysłano do sieci $cwnd_0 = k \cdot SMSS$ bajtów zawartych w k segmentach. Po przyjściu pierwszego potwierdzenia ACK wartość *cwnd* jest powiększona o czynnik $(1/k)$:

$$\text{cwnd}_1 = \text{cwnd}_0 + \text{SMSS} \cdot \text{SMSS} / \text{cwnd}_0 = k \cdot \text{SMSS} + \text{SMSS} \cdot (\text{SMSS} / (k \cdot \text{SMSS})) =$$
$$k \cdot \text{SMSS} + (1/k) \cdot \text{SMSS} = (k + (1/k)) \cdot \text{SMSS} = \text{cwnd}_0 + (1/k) \cdot \text{SMSS}$$

Ponieważ wartość *cwnd* wzrasta nieznacznie z każdym przyjściem nowego potwierdzenia ACK, a wartość ta występuje w mianowniku wyrażenia w pierwszym z powyższych równań, ogólny wzrost *cwnd* jest lekko podliniowy. Mimo to, na ogół przyjmujemy, że algorytm unikania przeciążenia generuje liniowy wzrost okna w odniesieniu do czasu (patrz rysunek 16.3), podczas gdy algorytm szybkiego startu zwiększa je wykładniczo względem czasu (patrz rysunek 16.2). Przedstawiona funkcja jest również nazywana **wzrostem addytywnym** (*additive increase*), ponieważ określona wartość (w tym przypadku mniej więcej jeden pakiet) jest dodawana do *cwnd* dla każdej pomyślnie odebranej porcji danych równej rozmiarowi okna.



Rysunek 16.3. *Działanie algorytmu unikania przeciążenia. W prostym przypadku, kiedy potwierdzenia ACK nie są opóźniane, każde przychodzące dobre potwierdzenie ACK pozwala nadawcy na dodatkowe wprowadzenie do sieci ułamka nowego pakietu równego w przybliżeniu 1/W. Prowadzi to do mniej więcej liniowego wzrostu rozmiaru okna nadawcy jako funkcji czasu (górny wykres po prawej stronie). Kiedy potwierdzenia ACK są opóźniane, tak jak wtedy, gdy potwierdzenie ACK jest generowane dla co drugiego pakietu, wzrost jest nadal w przybliżeniu liniowy, ale wolniejszy (dolny wykres po prawej stronie rysunku)*

Rysunek 16.3 (lewa część) ilustruje działanie algorytmu unikania przeciążenia. Podobnie jak poprzednio, czas jest wyrażony w jednostkach równych czasowi RTT połączenia. Przy założeniu, że połączenie przesyła cztery pakiety (pokazane na samej górze rysunku), zwracane są cztery potwierdzenia ACK, pozwalające na niewielkie zwiększenie wartości *cwnd*. Przed upływem drugiego okresu RTT przyrost tej wartości jest wystarczająco duży, aby po pokonaniu granicy zaokrąglenia w dół do liczby całkowitej spowodować wzrost *cwnd* o pojedynczą wartość SMSS i tym samym umożliwić wysłanie jednego dodatkowego pakietu. Wzrost wartości *cwnd* jako prawie liniowa funkcja czasu jest zilustrowany po prawej stronie rysunku na wykresie liniowym. Druga linia z prawej strony pokazuje, jak rośnie *cwnd*, kiedy potwierdzany jest co drugi pakiet, co symuluje użycie opóźnionych potwierdzeń ACK. W tym przypadku wzrost jest nadal prawie liniowy, ale nie tak szybki.

Algorytm unikania przeciążenia zakłada, że utrata pakietów powodowana przez błędy bitów jest bardzo mała (dużo mniejsza niż 1%), i dlatego utrata pakietu sygnalizuje wystąpienie przeciążenia gdzieś w sieci, między źródłem a punktem docelowym. Jeśli to założenie okazuje się fałszywe, co zdarza się czasem w przypadku sieci bezprzewodowych, protokół TCP spowalnia działanie nawet w sytuacji, gdy nie ma żadnego przeciążenia. W dodatku może być potrzebnych wiele czasów RTT, zanim wartość *cwnd* stanie się duża, co jest wymagane do efektywnego wykorzystania sieci o dużej pojemności. Rozwiązywanie tych problemów dotyczących protokołu TCP jest popularnym obszarem badań, a niektóre z różnorodnych podejść omówimy później.

16.2.3. Wybór między algorytmami powolnego startu i unikania przeciążenia

W normalnych sytuacjach połączenie TCP zawsze wykonuje albo powolny start, albo procedurę unikania przeciążenia, ale nigdy nie wykonuje obu algorytmów jednocześnie. Teraz zastanowimy się, co decyduje, jakiego algorytmu używa protokół TCP w określonym momencie czasu? Wiemy już, że powolny start jest stosowany wówczas, kiedy tworzone jest nowe połączenie lub kiedy dochodzi do retransmisji na podstawie przeterminowania. Teraz napiszemy, co steruje wyborem między powolnym startem a unikaniem przeciążenia.

Wcześniej wspomnieliśmy o progu *ssthresh*. Próg jest graniczną wartością *cwnd*, która rozstrzyga, który algorytm jest aktualnie stosowany, powolny start czy unikanie przeciążenia. Kiedy *cwnd* < *ssthresh*, używany jest algorytm powolnego startu, a kiedy *cwnd* > *ssthresh*, wykorzystywany jest algorytm unikania przeciążenia. Kiedy te dwie wartości są równe, może być stosowany dowolny z algorytmów. Najważniejszą różnicę między powolnym startem i unikaniem przeciążenia, jak widzieliśmy, stanowi sposób, w jaki każdy z algorytmów modyfikuje wartość *cwnd*, gdy przychodzą nowe potwierdzenia ACK. Tym, co sprawia, że protokół TCP jest nieco zawiły, a zarazem interesujący, jest fakt, że wartość progu *ssthresh* nie jest stała, ale zmienia się w czasie. Głównym celem progu *ssthresh* jest zapamiętanie ostatniego, „najlepszego" oszacowania okna operacyjnego, funkcjonującego bez utraty danych. Mówiąc inaczej, zawiera on dolną granicę najlepszego oszacowania optymalnego rozmiaru okna przez protokół TCP.

Wartość początkowa progu *ssthresh* może być ustawiona dowolnie wysoko (np. może być równa wartości *awnd* lub większa), co sprawia, że protokół TCP zawsze rozpoczyna działanie od powolnego startu. Kiedy dochodzi do retransmisji spowodowanej przez wyczerpanie się limitu czasu oczekiwania na retransmisję albo przez wykonanie algorytmu szybkiej retransmisji, próg *ssthresh* jest aktualizowany w następujący sposób:

$$ssthresh = \max(rozmiar\ przelotu / 2,\ 2 \cdot SMSS) \qquad [1]$$

W najnowszym stosie protokołów TCP/IP Microsoftu („Next Generation") równanie to jest podobno zamienione na nieco bardziej zachowawczą zależność: *ssthresh* = max(min(*cwnd*, *awnd*)/2, 2·*SMSS*) (patrz [NB08]).

Tu widzimy, że jeśli wymagana jest retransmisja, protokół TCP przyjmuje, iż okno operacyjne musiało być za duże jak na możliwości sieci. Zmniejszeniu oszacowania optymalnego rozmiaru okna towarzyszy zmiana progu *ssthresh* na mniej więcej połowę aktualnego rozmiaru okna (ale nigdy poniżej podwójnej wartości parametru SMSS). To zwykle skutkuje obniżeniem wartości *ssthresh*, ale może również spowodować zwiększenie wartości *ssthresh*. Jeśli zbadamy procedurę unikania przeciążenia używaną przez protokół TCP, przypomnimy sobie, że w sytuacji, gdy pomyślnie przetransmitowano dane w ilości równej całemu oknu, wolno zwiększyć wartość *cwnd* o ok. 1 SMSS. Tak więc, jeśli wartość *cwnd* rosła przez znaczną ilość czasu, ustalenie wartości progu *ssthresh* na połowę rozmiaru przelotu może spowodować jej zwiększenie. Tak się dzieje, kiedy protokół odkryje więcej użytecznego pasma. Współzależność między *ssthresh* i *cwnd* w połączeniu z działaniem powolnego startu i unikania przeciążenia nadaje charakterystyczną specyfikę działaniu protokołu TCP w obliczu przeciążenia. Teraz zbadamy pełne, połączone algorytmy.

16.2.4. Algorytmy Tahoe, Reno i szybkie odtwarzanie

Dotąd omawiane algorytmy powolnego startu i unikania przeciążenia są pierwszymi algorytmami kontroli przeciążenia stosowanymi w protokole TCP. Zostały wprowadzone w późnych latach 80. ubiegłego wieku razem z wydaniem 4.2 odmiany systemu UNIX powstałej na Uniwersytecie Kalifornijskim w Berkeley, która nosi nazwę *Berkeley Software Distribution* lub BSD UNIX. W ten sposób zapoczątkowano konwencję tworzenia nazw różnych wersji protokołu TCP od nazw miast w USA, szczególnie tych, gdzie dozwolone jest uprawianie hazardu.

Wydanie 4.2 systemu BSD (noszące nazwę Tahoe) zawierało wersję protokołu TCP, która rozpoczynała połączenia od pracy w trybie powolnego startu, a jeśli dochodziło do utraty pakietu wykrytej przez przeterminowanie albo przez procedurę szybkiej retransmisji, algorytm powolnego startu był inicjowany na nowo. Protokół TCP Tahoe został zaimplementowany w taki sposób, że po prostu zmniejszał parametr *cwnd* do jego początkowej wartości (1 SMSS w tamtym czasie) przy każdej utracie pakietu, wymuszając przejście połączenia do trybu powolnego startu, dopóki wartość *cwnd* nie osiągnęła progu *ssthresh*.

Jednym z problemów związanych z tym podejściem jest to, że w przypadku ścieżek z dużym iloczynem BDP może ono powodować znaczny stopień niewykorzystania dostępnego pasma przez połączenie w czasie, gdy protokół nadawczy TCP przechodzi przez procedurę

powolnego startu, żeby wrócić do punktu, w którym działał przed utratą pakietu. By rozwiązać ten problem, została ponownie rozważona reinicjacja powolnego startu przy każdej utracie pakietu. Ostatecznie przyjęto, że jeśli utrata pakietu zostaje wykryta przez zduplikowane potwierdzenia ACK (wywołujące szybką retransmisję), parametr *cwnd* jest resetowany do ostatniej wartości progu *ssthresh* zamiast do wartości tylko 1 SMSS. (Algorytm szybkiego startu jest nadal inicjowany w przypadku przeterminowania, co na ogół ma miejsce dla większości wariantów protokołu TCP). Podejście to umożliwia protokołowi TCP spowolnienie działania do połowy jego poprzedniej szybkości bez powracania do trybu powolnego startu.

Prowadząc dalsze badania kwestii ścieżek z dużym iloczynem BDP i rozważając ponownie wspomnianą wcześniej zasadę zachowania pakietów, zaobserwowano, że jakiekolwiek potwierdzenia ACK, które zostały odebrane, nawet w trakcie procesu odtwarzania po utracie danych, nadal stanowią okazję do wprowadzenia do sieci nowych pakietów. Ta obserwacja stała się podstawą procedury **szybkiego odtwarzania** (*fast recovery*), która została rozpowszechniona wraz z popularną wersją 4.3 BSD Reno systemu BSD UNIX. Procedura szybkiego odtwarzania pozwala wartości *cwnd* (tymczasowo) rosnąć o 1 SMSS dla każdego ACK odebranego w trakcie odtwarzania. Dlatego okno przeciążenia znajduje się przez pewien okres czasu w stanie „inflacji" (*is inflated*), pozwalając na wysłanie dodatkowego nowego pakietu dla każdego otrzymanego potwierdzenia ACK, dopóki nie pojawi się dobre potwierdzenie ACK. Każde niezduplikowane („dobre") potwierdzenie ACK powoduje wyjście protokołu TCP z fazy odtwarzania i zmniejsza okno przeciążenia do jego poprzedniej, normalnej (nieinflacyjnej) wartości. Protokół TCP Reno stał się bardzo popularny i w końcu został podstawą tego, co całkiem rozsądnie możemy nazwać „standardowym protokołem TCP".

16.2.5. Standardowy protokół TCP

Chociaż kwestia, co stanowi „standardowy" protokół TCP, jest przedmiotem pewnej dyskusji, algorytmy omówione przez nas do tej pory są głównymi procedurami identyfikowanymi ze standardowym działaniem protokołu TCP. Algorytmy powolnego startu i unikania przeciążenia są zwykle implementowane razem, a podstawy ich działania w ujęciu całościowym zostały przedstawione w dokumencie [RFC5681]. Wymieniona specyfikacja nie wymaga używania dokładnie tych algorytmów, ale nakłada wymaganie, aby żadna implementacja protokołu TCP nie przekraczała dopuszczonego przez te algorytmy stopnia agresywności.

Podsumowując łączny algorytm z dokumentu [RFC5681], stwierdzamy, że protokół TCP rozpoczyna połączenie procedurą powolnego startu (*cwnd* = *IW*) z dużą wartością progu *ssthresh*, na ogół przynajmniej równą wartości *awnd*. Po odebraniu dobrego potwierdzenia ACK (takiego, które potwierdza nowe dane) protokół TCP aktualizuje wartość *cwnd* w następujący sposób:

$$cwnd += SMSS \quad (\text{if } cwnd < ssthresh) \qquad \text{Wolny start}$$

$$cwnd += SMSS \cdot SMSS / cwnd \quad (\text{if } cwnd > ssthresh) \qquad \text{Unikanie przeciążenia}$$

Jeśli uruchamiana jest szybka retransmisja z powodu odebrania trzeciego zduplikowanego potwierdzenia ACK (lub innego sygnału, jeśli nie jest używany konwencjonalny sposób inicjowania szybkiej retransmisji), wykonywane są następujące działania.

1. Próg *ssthresh* jest aktualizowany i otrzymuje wartość nie większą niż podana w równaniu [1].
2. Wykonywany jest algorytm szybkiej retransmisji, a *cwnd* otrzymuje wartość (*ssthresh*+3·*SMSS*).
3. Parametr *cwnd* jest tymczasowo zwiększany o wartość *SMSS* dla każdego odebranego potwierdzenia ACK.
4. Kiedy zostaje odebrane dobre potwierdzenie ACK, *cwnd* ponownie otrzymuje wartość *ssthresh*.

Działania przedstawione w krokach 2. i 3. tworzą procedurę szybkiego odtwarzania (*fast recovery*). Krok 2. najpierw koryguje okno *cwnd*, co zwykle powoduje jego zmniejszenie do połowy poprzedniej wartości, a następnie wprowadza je tymczasowo w stan inflacji, z uwzględnieniem faktu, że odebranie każdego zduplikowanego potwierdzenia ACK wskazuje, że jakiś pakiet opuścił sieć (co pozwala na wprowadzenie następnego pakietu). W tym kroku ma także miejsce multiplikatywne zmniejszenie (*multiplicative decrease*), jako że okno *cwnd* jest zazwyczaj mnożone przez pewną liczbę (w tym przypadku 0,5) w celu utworzenia jego nowej wartości. Krok 3. kontynuuje proces inflacji, pozwalając nadawcy na wysłanie dodatkowych pakietów (przy założeniu, że nie jest przekroczona wartość *awnd*). W kroku 4. zakłada się, że protokół TCP zakończył odtwarzanie, więc tymczasowa inflacja zostaje zlikwidowana (dlatego 4. krok bywa czasem nazywany „deflacją").

Algorytm powolnego startu jest zawsze używany w dwóch przypadkach: kiedy rozpoczynane jest nowe połączenie i kiedy upływa czas oczekiwania przed wykonaniem retransmisji. Może także zostać uruchomiony, kiedy nadawca pozostawał przez stosunkowo długi czas w stanie bezczynności lub zaistnieje jakiś inny powód, by podejrzewać, że *cwnd* może nie odzwierciedlać dokładnie aktualnego stanu obciążenia sieci (patrz punkt 16.3.5). W takim przypadku początkowa wartość *cwnd* zostaje ustawiona zgodnie z **oknem restartu** (RW, *Restart Window*). Według zalecenia zawartego w dokumencie [RFC5681] $RW = \min(IW, cwnd)$. W innych sytuacjach, w których uruchamiany jest powolny start, *cwnd* otrzymuje wartość *IW*.

16.3. Ewolucja algorytmów standardowych

Klasyczne i standardowe algorytmy TCP ogromnie zaważyły na funkcjonowaniu protokołu TCP; w zasadzie rozwiązały poważny problem zapaści na skutek przeciążenia, występujący w Internecie.

Problem występującej w Internecie zapaści na skutek przeciążenia stanowił poważne zmartwienie w latach 1986–1988. W październiku 1986 r. zauważono, że sieć szkieletowa NSFNET, ważny składnik wczesnego Internetu, działała z efektywną przepustowością ok. 1000 razy mniejszą od tej, z którą powinna działać (nazwano to krachem NSFNET-u, *NSFNET meltdown*). Główną przyczyną problemu było niekontrolowane nasilenie agresywnych retransmisji w okresach utraty pakietów. Ten sposób zachowania protokołu wprowadził sieć w stan ustawicznego przeciążenia, w którym występowała masowa utrata pakietów (powodująca jeszcze więcej retransmisji) i przepływność była niska. Zastosowanie klasycznych algorytmów kontroli przeciążenia skutecznie wyeliminowało ten problem.

Pozostało jednak kilka obszarów do poprawienia. Ze względu na popularność protokołu TCP wkładano coraz większy wysiłek w zapewnienie dobrego działania protokołu TCP w coraz bardziej zróżnicowanych warunkach. Teraz wspomnimy o niektórych ulepszeniach, które znalazły zastosowanie w wielu współczesnych implementacjach TCP.

16.3.1. NewReno

Jeden z problemów dotyczących algorytmu szybkiego odtwarzania polega na tym, że kiedy dochodzi do utraty wielu pakietów z pojedynczego okna danych i zostanie odzyskany (tzn. pomyślnie dostarczony i potwierdzony przez ACK) jeden z pakietów, może zostać odebrane przez nadawcę dobre potwierdzenie ACK, które spowoduje, że tymczasowa inflacja okna występująca w trakcie szybkiego odtwarzania zostanie zlikwidowana, zanim wszystkie utracone pakiety zostaną przesłane ponownie. Potwierdzenia ACK, które uruchamiają takie działanie protokołu, są nazywane **częściowymi potwierdzeniami ACK** (*partial ACKs*). Protokół TCP Reno, reagując na częściowe potwierdzenia ACK przez zmniejszenie swojego powiększonego przez inflację okna przeciążenia może wejść w stan bezczynności, dopóki nie pojawi się sygnał od licznika czasu oczekiwania przed retransmisją. Aby zrozumieć, dlaczego tak się dzieje, przypomnijmy sobie, że (nieużywający opcji SACK) protokół TCP uruchamia swą procedurę szybkiej retransmisji, polegając na sygnale o trzech (lub innej liczbie, określonej wartością progu *dupthresh*) zduplikowanych potwierdzeniach ACK. Jeśli nie ma wystarczającej ilości pakietów w sieci, nie jest możliwe uruchomienie tej procedury po utracie pakietu, co w końcu prowadzi do upłynięcia limitu czasu oczekiwania przed retransmisją i wywołania procedury powolnego startu, która wpływa drastycznie na wydajność protokołu TCP w aspekcie przepływności.

W celu rozwiązanie tego problemu, występującego w TCP w wersji Reno, została opracowana modyfikacja o nazwie **NewReno** (patrz [RFC3782]). Procedura ta modyfikuje algorytm szybkiego odtwarzania, śledząc najwyższy numer sekwencyjny z ostatnio transmitowanego okna danych (tzn. punkt odtwarzania, z którym po raz pierwszy zetknęliśmy się w rozdziale 14.). Dopiero wtedy, gdy zostanie odebrane potwierdzenie z numerem ACK przynajmniej równym punktowi odtwarzania, jest likwidowana inflacja okna właściwa dla szybkiego odtwarzania. Pozwala to protokołowi TCP na kontynuację wysyłania jednego segmentu po każdym, odebranym w czasie odtwarzania, potwierdzeniu ACK i zmniejsza częstość występowania przekroczeń limitu czasu oczekiwania przed retransmisją, szczególnie wtedy, gdy następuje utrata wielu pakietów w pojedynczym oknie danych. NewReno jest popularną odmianą współczesnych implementacji protokołu TCP — nie występują w nim problemy charakterystyczne dla oryginalnego algorytmu szybkiego odtwarzania i jest znacznie mniej skomplikowany w implementacji niż opcje SACK. Jednakże przy wykorzystaniu opcji SACK protokół TCP może osiągnąć lepszą wydajność niż NewReno, kiedy dochodzi do utraty wielu pakietów w pojedynczym oknie danych, ale realizacja tego celu wymaga poświęcenia starannej uwagi procedurom, które omówimy w następnej kolejności.

16.3.2. Kontrola przeciążenia w TCP z użyciem opcji SACK

Dzięki wprowadzeniu w protokole TCP opcji SACK i selektywnego powtórzenia nadawca ma możliwość podejmowania lepszych decyzji o tym, jakie segmenty należy wysłać, aby wypełnić luki występujące u odbiorcy (patrz rozdział 14.). W trakcie wypełniania luk

u odbiorcy nadawca na ogół wysyła każdy z brakujących segmentów po kolei, dopóki wszystkie retransmisje utraconych segmentów nie zostaną skutecznie odebrane. Procedura ta różni się od wcześniej wspomnianej, podstawowej procedury szybkiej retransmisji i szybkiego odtwarzania w subtelny sposób.

W przypadku szybkiej retransmisji i szybkiego odtwarzania, w sytuacji kiedy dochodzi do utraty pakietu, nadawczy protokół TCP transmituje tylko segment, który uważa za utracony, i może wysłać nowe dane, jeśli pozwala na to okno *W*. Ponieważ okno jest powiększane (zachodzi proces inflacji) w czasie odtwarzania po każdym przychodzącym potwierdzeniu ACK, protokół TCP, mając do dyspozycji większe okna, ma zazwyczaj możliwość wysłania pewnych dodatkowych danych po wykonaniu swojej retransmisji. W protokole TCP używającym opcji SACK nadawca może być informowany o wielu brakujących segmentach i teoretycznie mógłby je wysłać wszystkie natychmiast, ponieważ wszystkie mieściłyby się w prawidłowym oknie. Jednak to mogłoby się wiązać z wysłaniem zbyt dużej ilości danych do sieci i w ten sposób zagrozić kontroli przeciążenia. W przypadku użycia protokołu TCP z opcjami SACK pojawia się następująca kwestia: użycie jedynie wartości *cwnd*, jako ograniczenia dla okna przesuwnego nadawcy, w celu wskazania, ile pakietów (i które) należy wysłać w okresach odtwarzania, nie jest wystarczające. Natomiast decyzja, **które** pakiety należy wysłać, musi zostać oddzielona od wyboru momentu, **kiedy** należy je wysłać. Mówiąc inaczej, protokół TCP używający opcji SACK podkreśla potrzebę oddzielenia obsługi przeciążenia od wyboru i mechanizmu retransmisji pakietów. Konwencjonalne (nieużywające opcji SACK) protokoły TCP mieszają ze sobą te zagadnienia.

Jednym ze sposobów tego rozdzielenia jest sprawienie, aby protokół TCP śledził ilość danych, które wprowadził do sieci, odrębnie od utrzymywania okna. W dokumencie [RFC3517] użyty w tym celu mechanizm nosi nazwę zmiennej *pipe* i zawiera oszacowanie rozmiaru przelotu. Co istotne, zmienna *pipe* zlicza bajty (lub pakiety, zależnie od implementacji) **transmisji i retransmisji**, pod warunkiem że nie wiadomo o nich, że zostały utracone. Przy założeniu, że wartość *awnd* jest duża, protokołowi TCP używającemu opcji SACK wolno wysłać segment za każdym razem, gdy prawdziwa jest następująca relacja: $cwnd - pipe \geq SMSS$. Innymi słowy, okno *cwnd* jest nadal używane do określenia limitu ilości danych, które mogą znajdować się w sieci, ale szacunkowa ilość danych w sieci jest rozliczana odrębnie od samego okna. To, jak przedstawia się porównanie protokołu TCP używającego opcji SACK i korzystającego z omówionego wyżej podejścia do kontroli przeciążenia z konwencjonalnym protokołem TCP, zostało szczegółowo zbadane przez przeprowadzenie szeregu symulacji opisanych w artykule [FF96].

16.3.3. Potwierdzenie generowane w przód (FACK) i zmniejszanie szybkości transmisji o połowę

W wariantach protokołu TCP opartych na Reno (w tym NewReno) typowe działanie polega na tym, że kiedy okno *cwnd* zostaje zmniejszone po szybkiej retransmisji, muszą zostać odebrane potwierdzenia ACK dla co najmniej połowy znajdujących się w sieci danych bieżącego okna, zanim nadawczy protokół będzie mógł kontynuować transmitowanie. Jest to oczekiwana konsekwencja zmniejszenia okna przeciążenia o połowę natychmiast po wykryciu utraty danych. To powoduje, że nadawczy protokół TCP czeka przez ok. połowę czasu RTT, a potem wysyła wszystkie nowe dane w trakcie

drugiej połowy tego samego czasu RTT, co powoduje bardziej impulsowe działanie protokołu, niż to jest naprawdę konieczne.

W usiłowaniu uniknięcia początkowej przerwy w transmisjach po utracie danych, bez naruszenia konwencji polegającej na wyjściu z procesu odtwarzania z oknem przeciążenia ustawionym na połowę swojego rozmiaru na wejściu, został przedstawiony w pracy [MM96] mechanizm **potwierdzenia generowanego w przód** (FACK, *Forward Acknowledgement*). Składa się on z dwóch algorytmów noszących angielskie nazwy *overdamping* i *rampdown* (odpowiednio: silne tłumienie, stłumienie). Od czasu swojej pierwszej propozycji autorzy uaktualnili metodę, tworząc ujednolicony i ulepszony algorytm, który nazwali ***rate halving*** (zmniejszanie szybkości transmisji o połowę), opierając się na wcześniejszej pracy Hoe'a [H96]. Aby osiągnąć pewność, że algorytm działa tak efektywnie, jak to tylko możliwe, dalej modyfikują jego działanie przez dodanie parametrów ograniczających, co daje w efekcie kompletny algorytm noszący nazwę *Rate Halving with Bounding Parameters* (RHBP; *rate halving* z parametrami ograniczającymi — patrz [PSCRH]).

Podstawowe działanie algorytmu RHBP pozwala protokołowi TCP nadawcy na wysłanie jednego pakietu dla każdych dwóch zduplikowanych potwierdzeń ACK odebranych w ciągu jednego czasu RTT. To sprawia, że wykonujący proces odtwarzania protokół TCP wysyła odpowiednią ilość danych przed zakończeniem okresu odtwarzania, ale reguluje tempo wysyłania danych i rozkłada transmisje równomiernie, nie skupiając ich w drugiej połowie okresu RTT. Unikanie zbijania transmisji w grupy, czyli impulsowości ruchu, jest korzystne, ponieważ zgrupowane sekwencje transmisji mają tendencję do utrzymywania się przez wiele okresów RTT, co obciąża nadmiernie bufory routerów.

W celu utrzymywania dokładnego oszacowania rozmiaru przelotu algorytm RHBP wykorzystuje informacje zawarte w opcjach SACK do ustalenia wartości FACK: najwyższego numeru sekwencyjnego, o którym wiadomo, że dotarł do odbiorcy, plus 1. Obliczenie różnicy między numerem sekwencyjnym, który właśnie ma być wysłany przez nadawcę (parametr SND.NXT na rysunku 15.9), a wartością FACK daje oszacowanie rozmiaru przelotu, z wyłączeniem retransmisji.

W algorytmie RHBP dokonano rozróżnienia między **interwałem regulacji** (*adjustment interval*; okres, kiedy ma miejsce modyfikacja okna *cwnd*) a **interwałem naprawy** (*repair interval*; okres, w którym są retransmitowane pewne segmenty). Wejście w interwał regulacji następuje natychmiast, gdy pojawi się utrata danych lub wskaźnik przeciążenia. Ostateczna wartość *cwnd*, w momencie zakończenia interwału, jest równa połowie prawidłowo dostarczonej części okna danych znajdujących się w sieci w chwili wykrycia symptomu przeciążenia. Nadawcy używającemu algorytmu RHBP wolno transmitować dane, jeśli spełniona jest następująca zależność:

$$(SND.NXT - fack + dane\_retran + dp) < cwnd$$

Powyższe wyrażenie ujmuje rozmiar przelotu z uwzględnieniem retransmisji i zapewnia, że wprowadzenie następnego pakietu o długości *dp* nie spowoduje przekroczenia wartości *cwnd*. Pod warunkiem, że wszystkie dane poprzedzające numer FACK faktycznie nie znajdują się już w sieci (tzn. zostały utracone albo przechowane przez odbiorcę), spełnienie powyższej zależności sprawia, że nadawca używający opcji SACK jest właściwie kontrolowany przez okno *cwnd*. Jednak formuła ta może okazać się zbyt agresywna,

w sytuacji gdy zmieniona została kolejność pakietów w sieci, ponieważ wtedy luki wskazywane przez opcję SACK nie oznaczają utraconych danych.

W systemie Linux zarówno potwierdzenia FACK, jak i algorytm *rate halving* są zaimplementowane i domyślnie włączone. Mechanizm FACK jest aktywowany tylko wtedy, kiedy włączone jest użycie opcji SACK i konfiguracyjna zmienna logiczna `net.ipv4.tcp_fack` ma wartość 1. Kiedy w sieci zostaje wykryta zmiana kolejności pakietów, bardziej agresywne działanie potwierdzeń FACK zostaje wyłączone.

Metoda *rate halving* jest jednym z kilku sposobów regulacji rytmu działania procedury nadawczej protokołu TCP w celu uniknięcia lub ograniczenia impulsowości. Chociaż oferuje ona szereg korzyści, generuje także kilka problemów. W pracy [ASA00] autorzy analizują regulowanie rytmu działania protokołu TCP z pewną szczegółowością przy wykorzystaniu symulacji i dochodzą do wniosku, że daje ono gorszą wydajność niż TCP Reno. Co więcej, wiadomo, że protokół TCP korzystający z metody *rate halving* wykazuje słabą wydajność, kiedy połączenie może zostać ograniczone przez proponowane okno odbiorcy (patrz [MM05]).

16.3.4. Algorytm ograniczonej transmisji

W dokumencie [RFC3042] autorzy proponują algorytm **ograniczonej transmisji** (*limited transmit*), małą modyfikację protokołu TCP, zaprojektowaną w celu poprawienia wydajności protokołu w sytuacji, gdy użyteczne okno jest małe. Pamiętamy z naszego doświadczenia dotyczącego protokołu TCP Reno, że kiedy używane jest małe okno, może nie być wystarczającej ilości pakietów w sieci, aby uruchomić algorytmy szybkiej retransmisji i szybkiego odtwarzania, kiedy dochodzi do utraty pakietu, jako że te algorytmy zwykle wymagają odnotowania trzech zduplikowanych potwierdzeń ACK przed rozpoczęciem działania.

Protokół TCP posiadający niewysłane dane i używający algorytmu ograniczonej transmisji może wysłać nowy pakiet dla każdej pary kolejnych zduplikowanych potwierdzeń ACK, które odbierze. Ten sposób działania pomaga utrzymać przynajmniej minimalną liczbę pakietów w sieci — wystarczającą do tego, aby po wystąpieniu utraty pakietu mogła być uruchomiona szybka retransmisja. Jest to korzystne dla protokołu TCP, ponieważ odczekiwanie okresu RTO (który może trwać stosunkową dużą ilość czasu — kilkaset milisekund) może znacznie pogorszyć wydajność w zakresie przepływności. Od opublikowania dokumentu [RFC5681] procedura ograniczonej transmisji jest zalecanym sposobem działania protokołu TCP. Zauważmy, że algorytm *rate halving* jest jedną z form algorytmu ograniczonej transmisji.

16.3.5. Walidacja okna przeciążenia (CWV)

Jedna z kwestii związanych z obsługą przeciążenia w protokole TCP powstaje, kiedy nadawca zatrzymuje wysyłanie danych na pewien okres czasu, dlatego że nie ma już więcej danych do wysłania albo nie może wysyłać danych, kiedy chce, z jakiegoś innego powodu. Jeśli wszystko przebiega dobrze, nadawca nigdy nie pauzuje, kontynuując wysyłanie danych i odbieranie potwierdzeń ACK od drugiej strony połączenia. Ta ciągłość

otrzymywania informacji zwrotnych umożliwia nadawcy utrzymywanie w miarę aktualnego (w obrębie pojedynczego czasu RTT) oszacowania właściwych wartości okna *cwnd* i progu *ssthresh*.

Gdy nadawczy protokół TCP wysyła dane już od pewnego czasu, jego okno *cwnd* mogło osiągnąć znaczny rozmiar. Jeśli potem nadawca zaprzestaje na pewien czas wysyłania, ale później wznawia je, duże okno *cwnd* umożliwia mu bezzwłoczne wprowadzenie do sieci nazbyt dużej liczby pakietów (czyli może wystąpić nagły, przejściowy wzrost szybkości transmisji). Co więcej, jeśli przerwa w transmisjach była wystarczająco duża, ostatnia wartość *cwnd* może już nie być odpowiednia dla ścieżki i stanu obciążenia sieci.

W dokumencie [RFC2861] autorzy proponują eksperymentalny mechanizm nazwany **walidacją okna przeciążenia** (CWV, *Congestion Window Validation*). W zasadzie bieżąca wartość okna *cwnd* nadawcy zmniejsza się stopniowo w ciągu okresu nieużywania, a próg *ssthresh* zachowuje jej „pamięć" sprzed rozpoczęcia procesu zmniejszania. Dla lepszego zrozumienia omawianego mechanizmu wprowadzono rozróżnienie między nadawcą **bezczynnym** (*idle*) i nadawcą **ograniczonym przez aplikację** (*application-limited*). Nadawca bezczynny przestał generować dane, które chce wysłać do sieci; potwierdzenia ACK dla wszystkich danych, które dotychczas wysłał, zostały już odebrane. Tak więc połączenie jest całkowicie nieaktywne — nie przepływają żadne dane ani potwierdzenia ACK, z wyjątkiem sporadycznych aktualizacji okna (patrz rozdział 15.). Nadawca ograniczony przez aplikację ma jeszcze dane do wysłania, ale z jakiegoś powodu nie może tego uczynić. Może to wynikać z tego, że komputer nadawcy jest zajęty wykonywaniem innych zadań albo jakiś mechanizm lub protokół niższej warstwy uniemożliwia wysyłanie danych. Przypadek ten powoduje niepełne wykorzystanie dozwolonego okna przeciążenia, ale połączenie nie jest całkowicie nieaktywne; mogą wciąż powracać potwierdzenia ACK dla poprzednio wysłanych danych.

Algorytm CWV działa następująco: zawsze, kiedy ma być wysłany nowy pakiet, mierzony jest czas, jaki upłynął od ostatniego zdarzenia wysłania danych, w celu ustalenia, czy przekracza wartość jednego czasu RTO. Jeśli tak jest, to:

- próg *ssthresh* jest modyfikowany, ale nie zmniejszany — otrzymuje wartość $\max(ssthresh, (3/4) \cdot cwnd)$,
- okno *cwnd* jest zmniejszane o połowę dla każdego czasu RTT w okresie bezczynności, ale zawsze ma wartość co najmniej równą 1 SMSS.

Dla okresów ograniczenia przez aplikację, które nie są okresami bezczynności, stosowany jest podobny sposób działania, który przedstawia się następująco.

- Ilość danych aktualnie używanego okna jest zapamiętywana w zmiennej *W_used*.
- Próg *ssthresh* jest modyfikowany, ale nie zmniejszany — otrzymuje wartość $\max(ssthresh, (3/4) \cdot cwnd)$.
- Okno *cwnd* otrzymuje wartość równą średniej z *cwnd* i *W_used*.

Obie z powyżej opisanych zmian zmniejszają wartość okna *cwnd*, „zapamiętując" ją przedtem w ustawieniu progu *ssthresh*. Pojedyncze wystąpienie pierwszego przypadku może w sposób dramatyczny wpłynąć na wartość *cwnd*, jeśli aplikacja pozostaje bezczynna przez długi okres czasu. Ten sposób obsługi okna przeciążenia może w pewnych

warunkach prowadzić do lepszej wydajności. Jak podają autorzy algorytmu, redukcja nagłego, chwilowego wzrostu ilości pakietów, który może pojawić się po okresie bezczynności, łagodzi napór na potencjalnie ograniczoną przestrzeń buforów w routerach, co prowadzi ostatecznie do mniejszej liczby utraconych pakietów. Zauważmy, że ponieważ okno *cwnd* ulega zmniejszeniu, a próg *ssthresh* nie, typową konsekwencją zastosowania tego algorytmu jest, po wystarczająco długiej przerwie w aktywności nadawcy, przeniesienie go do fazy powolnego startu. Algorytm CWV jest domyślnie włączony w implementacjach protokołu TCP systemu Linux.

16.4. Obsługa zbędnych retransmisji — algorytm odpowiedzi Eifel

Jak widzieliśmy w rozdziale 15., kiedy protokół TCP trafia na duży, nagły skok opóźnienia, może to spowodować wystąpienie przeterminowania prowadzącego do retransmisji nawet wtedy, gdy nie doszło do utraty pakietu. Takie zbędne retransmisje pojawiają się w szeregu okoliczności związanych ze zmianami w znajdującej się poniżej warstwie łącza danych (okoliczności takich jak przenoszenie połączeń w systemie telefonii komórkowej) lub nagłym wystąpieniem poważnego przeciążenia przyczyniającego się do dużego wzrostu czasu RTT. Kiedy to się dzieje, protokół TCP koryguje wartość progu *ssthresh* i wchodzi w procedurę powolnego startu, przypisując oknu *cwnd* wartość *IW*. Jeśli nie zostały zgubione żadne pakiety, potwierdzenia ACK przychodzące po przeterminowaniu powodują stosunkowo szybki wzrost okna *cwnd*, ale protokół TCP nadal wykonuje niepotrzebne retransmisje; nie wykorzystuje w pełni dostępnej pojemności, dopóki wartości *cwnd* i *ssthresh* nie ustalą się ponownie.

W celu uniknięcia problemów z wydajnością związanych ze zbędnymi retransmisjami zaproponowano kilka metod ich wykrywania. Niektóre z nich (np. DSACK, Eifel, F-RTO) omówiliśmy w rozdziale 14. Każda z tych metod, a być może inne, które mogą zostać opracowane, może być połączona z algorytmem odpowiedzi, wykorzystywanym do „wycofania" zmian, jakie protokół TCP wprowadza po wystąpieniu przeterminowania w swoich zmiennych kontroli przeciążenia. Jednym z popularnych (tzn. należących do ścieżki standardów IETF) algorytmów odpowiedzi jest **algorytm odpowiedzi Eifel** (patrz [RFC4015]).

Projekt Eifel zawiera zarówno algorytm wykrywania, jak i algorytm odpowiedzi, które są logicznie rozłączne. Każda implementacja protokołu TCP używająca algorytmu odpowiedzi Eifel jest zobowiązana do użycia jednego z algorytmów wykrywania zdefiniowanych w standardowym lub eksperymentalnym dokumencie RFC (tzn. algorytmu, który jest udokumentowany).

Algorytm odpowiedzi Eifel ma na celu obsługę licznika czasu retransmisji oraz stanu kontroli przeciążenia po upłynięciu czasu wskazywanego przez licznik czasu retransmisji. W tym miejscu omówimy tylko fragmenty algorytmu odpowiedzi dotyczące kwestii przeciążenia. Algorytm jest uruchamiany po wykonaniu pierwszej retransmisji na podstawie przeterminowania. Jego celem jest wycofanie zmiany progu *ssthresh* w sytuacji, gdy retransmisja została uznana za zbędną. W każdym przypadku, zanim wartość progu *ssthresh* zostanie zmodyfikowana w wyniku wystąpienia RTO (*retransmission timeout*

— przeterminowanie powodujące retransmisję), jest zapamiętywana w specjalnej zmiennej zgodnie z następującym wzorem: *pipe_prev* = min(*rozmiar przelotu*, *ssthresh*). Po wykonaniu tej czynności wywoływany jest algorytm wykrywania, podobny do jednego z wcześniej wspomnianych, w celu ustalenia, czy przeterminowanie RTO nie jest fałszywe (czyli prowadzące do zbędnej retransmisji). Jeśli jest fałszywe, kiedy przychodzi potwierdzenie ACK po retransmisji, wykonywane są następujące kroki.

1. Jeśli odebrane, dobre potwierdzenie ACK zawiera ustawioną flagę ECN-Echo, zakończ działanie (patrz podrozdział 16.11).
2. *cwnd* = *rozmiar przelotu* + min(*bajty_potw*, *IW*) (przy założeniu, że okno *cwnd* jest mierzone w bajtach).
3. *ssthresh* = *pipe_prev*.

Zmienna *pipe_prev* zostaje ustawiona, zanim próg *ssthresh* zostanie zmieniony w zwykły sposób. Przechowuje ona pierwotną wartość progu *ssthresh*, która może zostać przywrócona w kroku 3., jeśli będzie to konieczne. Krok 1. dotyczy przypadku, w którym przychodzące potwierdzenie ACK przekazuje flagę ECN. (Flagę ECN omówimy dokładniej w podrozdziale 16.11). Kiedy ma to miejsce, wycofanie redukcji progu *ssthresh* jest uważane za niebezpieczne, więc algorytm kończy swoje działanie. Kroki 2. i 3. stanowią tę ważną część algorytmu (w odniesieniu do okna *cwnd*). Krok 2. odtwarza okno *cwnd* do wartości, przy której będzie możliwe wprowadzenie do sieci pewnej ilości dodatkowego ruchu, ale nie więcej niż *IW* bajtów nowych danych. Uważa się, że wartość *IW* określa bezpieczną ilość danych, którą można wprowadzić do ścieżki sieciowej z nieznanym stanem obciążenia. Krok 3. przywraca próg *ssthresh* do wartości sprzed wystąpienia przeterminowania RTO, co kończy operację wycofywania zmian parametrów kontroli przeciążenia.

16.5. Rozszerzony przykład

Teraz zajmiemy się rozszerzonym przykładem w celu zademonstrowania większości procedur opisanych w poprzednich podrozdziałach. Posługując się programem sock, realizujemy przesłanie przez linię DSL ok. 2,5 MB danych od nadawcy, którym jest system Linux 2.6, do systemu FreeBSD 5.4 występującego w roli odbiorcy. Linia DSL ma ograniczoną przepustowość w kierunku nadawania do ok. 300 kb/s. Odbiorca FreeBSD jest dołączony do łącza o dużej szerokości pasma. Minimalny czas RTT między nadawcą a odbiorcą wynosi 15,9 ms, a ścieżka zawiera 17 przeskoków. Systemy są tak skonfigurowane, aby używać podstawowych algorytmów (tzn. powolnego startu i unikania przeciążenia) w trakcie większości swoich działań. W ten sposób unikamy wielu szczegółów specyficznych dla systemów operacyjnych. (Niektóre z nich omówimy później). W celu zainicjowania naszego eksperymentu uruchamiamy w systemie odbiorcy następujące polecenie:

```
FreeBSD% sock -i -r 32768 -R 233016 -s 6666
```

Polecenie to ustala, że program sock będzie używał dość dużego bufora odbiorczego gniazda (228 kB) i będzie powodował wykonywanie dość dużych odczytów przez aplikację (32 kB). Przy używanej w przykładzie ścieżce jest to odpowiedni rozmiar bufora dla

odbiorcy. W systemie nadawcy uruchamiamy program sock w trybie wysyłania danych w sposób następujący:

```
Linux% sock -n20 -i -w 131072 -S 262144 128.32.37.219 6666
```

Polecenie to wybiera duży bufor nadawczy i wysyła 20·131 072 bajtów (2,5 MB) danych. Ślad pokazujący przesyłane pakiety jest przechwycony przy użyciu programu tcpdump po stronie nadawcy. Polecenie użyte do przechwycenia tego śladu wygląda następująco:

```
Linux# tcpdump -s 128 -w sack-to-free-12.td port 6666
```

Zapewnia to przechwycenie przynajmniej 128 bajtów każdego pakietu, wystarczająco dużo do uzyskania całej interesującej informacji zawartej w nagłówkach protokołów TCP i IP. Po zebraniu informacji zawartych w śladzie możemy użyć narzędzia tcptrace (patrz [TCPTRACE]) w celu uzyskania szeregu użytecznych, zbiorczych statystyk dotyczących połączenia:

```
Linux% tcptrace -Wl sack-to-free-12.td
```

Powyższe polecenie nakazuje programowi dostarczenie informacji o oknie przeciążenia, wyprowadzając je przy użyciu długiego (opisowego) formatu. Generuje to następujące dane wyjściowe:

```
1 arg remaining, starting with 'sack-to-free-12.td'
Ostermann's tcptrace -- version 6.6.7 -- Thu Nov 4, 2004

3175 packets seen, 3175 TCP packets traced
elapsed wallclock time: 0:00:00.167213, 18987 pkts/sec analyzed
trace file elapsed time: 0:01:40.475872
TCP connection info:
1 TCP connection traced:
TCP connection 1:
      host a:        adsl-63-203-72-138.dsl.snfc21.pacbell.net:1059
      host b:        dwight.CS.Berkeley.EDU:6666
      complete conn: yes
      first packet:  Wed Sep 28 22:15:29.956897 2005
      last packet:   Wed Sep 28 22:17:10.432769 2005
      elapsed time:  0:01:40.475872
      total packets: 3175
      filename:      sack-to-free-12.td
a->b:                                 b->a:
total packets:            1903        total packets:            1272
ack pkts sent:            1902        ack pkts sent:            1272
pure acks sent:              2        pure acks sent:           1270
sack pkts sent:              0        sack pkts sent:             79
dsack pkts sent:             0        dsack pkts sent:             0
max sack blks/ack:           0        max sack blks/ack:           2
unique bytes sent:     2621440        unique bytes sent:           0
actual data pkts:         1900        actual data pkts:            0
actual data bytes:     2659240        actual data bytes:           0
rexmt data pkts:            27        rexmt data pkts:             0
rexmt data bytes:        37800        rexmt data bytes:            0
zwnd probe pkts:             0        zwnd probe pkts:             0
zwnd probe bytes:            0        zwnd probe bytes:            0
outoforder pkts:             0        outoforder pkts:             0
```

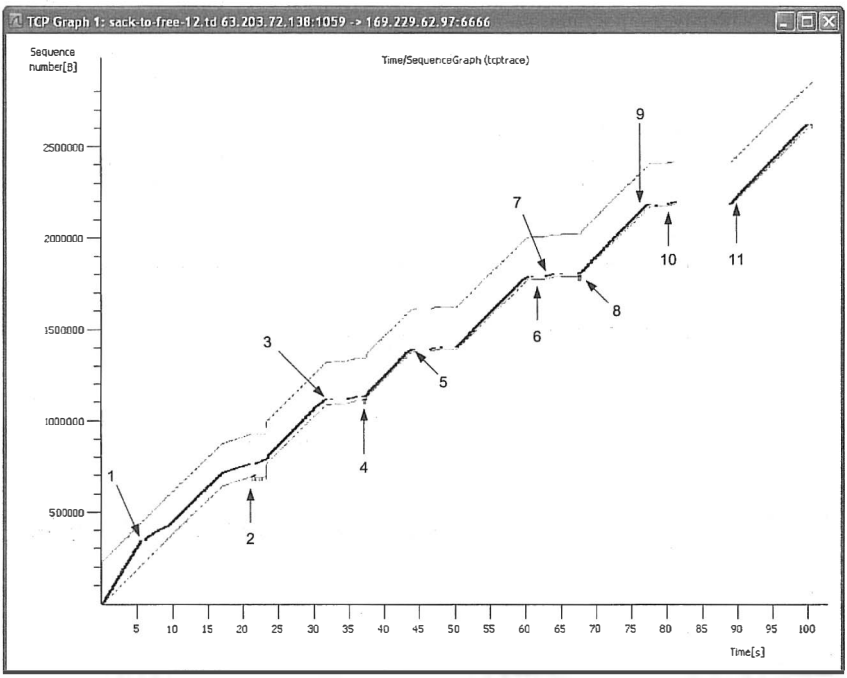
```
pushed data pkts:          44          pushed data pkts:           0
SYN/FIN pkts sent:         1/1         SYN/FIN pkts sent:        1/1
req 1323 ws/ts:            Y/Y         req 1323 ws/ts:           Y/Y
adv wind scale:            2           adv wind scale:           2
req sack:                  Y           req sack:                 Y
sacks sent:                0           sacks sent:              79
urgent data pkts:          0 pkts      urgent data pkts:         0 pkts
urgent data bytes:         0 bytes     urgent data bytes:        0 bytes
mss requested:          1412 bytes     mss requested:         1460 bytes
max segm size:          1400 bytes     max segm size:            0 bytes
min segm size:           640 bytes     min segm size:            0 bytes
avg segm size:          1399 bytes     avg segm size:            0 bytes
max win adv:            5808 bytes     max win adv:         233016 bytes
min win adv:            5808 bytes     min win adv:         170016 bytes
zero win adv:              0 times     zero win adv:             0 times
avg win adv:            5808 bytes     avg win adv:         232268 bytes
max owin:             137201 bytes     max owin:                 1 bytes
min non-zero owin:         1 bytes     min non-zero owin:        1 bytes
avg owin:              37594 bytes     avg owin:                 1 bytes
wavg owin:             33285 bytes     wavg owin:                0 bytes
initial window:         2800 bytes     initial window:           0 bytes
initial window:            2 pkts      initial window:           0 pkts
ttl stream length:   2621440 bytes     ttl stream length:        0 bytes
missed data:               0 bytes     missed data:              0 bytes
truncated data:      2556640 bytes     truncated data:           0 bytes
truncated packets:      1900 pkts      truncated packets:        0 pkts
data xmit time:        99.631 secs     data xmit time:       0.000 secs
idletime max:         7778.8 ms        idletime max:        7930.4 ms
throughput:            26090 Bps       throughput:               0 Bps
```

Dzięki temu pożytecznemu narzędziu możemy dowiedzieć się całkiem sporo o połączeniu. Interesuje nas głównie lewa część danych wyjściowych (`a->b`). Przede wszystkim widzimy, że w kierunku `a->b` były przesłane 1903 pakiety, a 1902 z nich były potwierdzeniami ACK. Tego się można było spodziewać, jako że pierwszym wysłanym pakietem jest zwykle SYN — jedyny pakiet bez włączonej flagi ACK. Określenie Pure ACKs (czyste potwierdzenia ACK) odnosi się do pakietów niezawierających żadnych danych. Nadawca tworzy jeden z nich w początkowej fazie połączenia, kiedy dostarcza potwierdzenia ACK dla otrzymanego od drugiej strony pakietu SYN+ACK, a drugi pakiet stanowi końcowe potwierdzenie ACK, wysłane na etapie zamykania połączenia, a więc jest również czymś spodziewanym. Z drugiej kolumny listingu (kierunek `b->a`) dowiadujemy się, że odbiorca wysłał 1272 pakiety, z których wszystkie są potwierdzeniami ACK. Spośród nich 1270 było czystymi potwierdzeniami ACK. Zauważamy również, że wysłano 79 pakietów SACK (tzn. potwierdzeń ACK zawierających opcję SACK). Jedyne dwa potwierdzenia ACK, które nie są czyste, to segmenty SYN+ACK i FIN+ACK wysłane odpowiednio na początku i na końcu połączenia.

Następne pięć wartości pokazuje udział danych, które były retransmitowane. Jak widzimy, zostało przesłanych 2 621 440 unikalnych (tzn. nieretransmitowanych) bajtów, ale w sumie przesłano 2 659 240 bajtów, co oznacza, że jakieś 2 659 240 – 2 621 440 = 37 800 bajtów musiało zostać przesłanych więcej niż jeden raz. Potwierdzają ten fakt dwa następne pola, bo wskazują, że te retransmitowane bajty były zawarte w 27 retransmitowanych pakietach, co daje przeciętny rozmiar retransmitowanego segmentu wielkości 1399 bajtów. Ponieważ w tym połączeniu wykonano transfer 2 659 240 bajtów w czasie 100,476 s, jego średnia przepustowość wyniosła 26 466 bajtów na sekundę (ok. 212 kb/s).

Jego przepustowość skuteczna, czyli ilość nieretransmitowanych danych przesłanych w jednostce czasu, wyniosła 2 621 440 / 100,476 = 26 090 B/s, tj. ok. 209 kb/s. Jak zobaczymy, połączenie to doznaje szeregu znaczących zakłóceń naruszających jego normalne działanie. Skorzystamy z możliwości analitycznych programu Wireshark i przeprowadzimy swoją własną analizę zachowania protokołu TCP po wystąpieniu tego rodzaju zdarzeń.

W celu uzyskania wizualnego przedstawienia śladu możemy użyć funkcji z menu *Statistics* programu Wireshark: *Statistics/TCP Stream Graph/Time-Sequence Graph (tcptrace)*, aby otrzymać obraz pokazany na rysunku 16.4 (uzupełniony strzałkami ułatwiającymi późniejszą analizę).



Rysunek 16.4. *Wygenerowany przez program Wireshark ślad przesłania na serwer pliku wielkości 2,5 MB przez nadawcę używającego protokołu TCP w systemie Linux 2.6.10, korzystającego z linii DSL o przepustowości ograniczonej do ok. 300 kb/s. Ciemna linia wykresu reprezentuje wysłane numery sekwencyjne. Górna linia pokazuje najwyższy numer sekwencyjny proponowany przez odbiorcę (prawą krawędź okna odbiorcy), a dolna linia przedstawia najwyższy numer sekwencyjny, którego potwierdzenie przez odbiorcę zostało już odebrane przez nadawcę. 11 zdarzeń oznaczonych etykietami przedstawia momenty, w których nastąpiła modyfikacja okna przeciążenia*

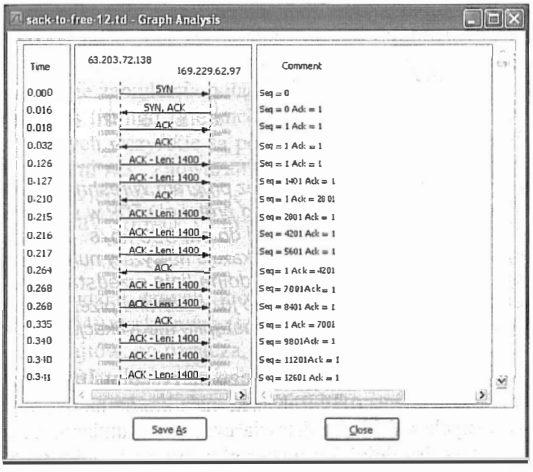
Oś Y wykresu pokazanego na rysunku 16.4 przedstawia względne wartości numeru sekwencyjnego TCP. Każda mała kreska na osi odpowiada 100 000 numerów sekwencyjnych. Oś X przedstawia czas w sekundach. Ciemna ciągła linia zawiera wiele mniejszych odcinków w kształcie litery I, z których każdy reprezentuje zakres numerów sekwencyjnych zawartych w pojedynczym segmencie TCP.

Wysokość pojedynczego znaku I pokazuje rozmiar ładunku danych użytkownika wyrażony w bajtach. Nachylenie „linii" utworzonej przez te I-kształtne znaczki pokazuje szybkość transmisji danych osiągniętą przez połączenie. Każde przesunięcie w prawo i w dół wskazuje retransmisję. Nachylenie linii w danym zakresie czasu wyznacza przeciętną przepływność osiągniętą w tym przedziale czasowym. Jak widzimy, najwyższy numer sekwencyjny wysłanych danych miał wartość ok. 2600000 osiągniętą w czasie 100, co pozwala z grubsza określić przeciętną wartość skutecznej przepustowości (*goodput*) na 26 000 bajtów na sekundę, a więc całkiem blisko numerycznej wartości wyświetlonej poprzednio przez program `tcptrace`.

Górna linia wykresu pokazuje największy numer sekwencyjny, który odbiorca jest gotów przyjąć do tego momentu (najwyższą wartość mieszczącą się w proponowanym oknie). Jak widzimy, na początku szeregów czasowych linia ta wskazuje wartość ok. 250000, przy rzeczywistej wartości wynoszącej 233016, jak pokazują dane wyświetlone przez program `tcptrace` w kolumnie `b->a`. Dolna linia przedstawia najwyższy numer ACK odebrany, jak dotąd, przez nadawcę. Jak wynika z naszej poprzedniej analizy, protokół TCP szuka dodatkowego pasma w trakcie działania przez zwiększanie swojego okna przeciążenia. Widzimy to działanie na naszym wykresie, kiedy ciemna linia odsuwa się w miarę upływu czasu od dolnej linii w kierunku górnej linii. Jeżeli górna linia nigdy nie zostaje osiągnięta, to wówczas albo nadawca, albo użytkowa pojemność sieci stanowią czynnik ograniczający dla przepływności połączenia. Jeśli górna linia jest zawsze osiągana, prawdopodobnym czynnikiem ograniczającym jest okno odbiorcy.

16.5.1. Działanie procedury powolnego startu

Rozpoczynamy naszą analizę, obserwując działanie wcześniej opisanego algorytmu powolnego startu. W programie Wireshark wybieramy pierwszy pakiet śladu, a następnie używamy funkcji *Statistics/Flow graph*, aby zilustrować pakiety wymieniane na początku połączenia (patrz rysunek 16.5).



Rysunek 16.5. *Analiza wykonana przez program Wireshark pokazuje numery sekwencyjne oraz numery ACK wymienianych pakietów, kiedy połączenie jest ustanawiane. Każde potwierdzenie ACK odebrane przez nadawcę uwalnia dwa lub trzy pakiety. Ta charakterystyka działania jest typowa dla nadawcy w fazie powolnego startu*

Na rysunku widzimy początkową wymianę segmentów SYN i SYN+ACK. Potwierdzenie ACK w czasie 0,032 jest aktualizacją okna (patrz rozdział 15.). Pierwsze dwa pakiety z danymi zostają wysłane w czasach 0,126 i 0,127. Potwierdzenie ACK w czasie 0,210 nie dotyczy pojedynczego pakietu. Zawarty w nim numer ACK ma wartość 2801 i w ten sposób potwierdza obydwa przedtem wysłane pakiety z danymi, dzięki kumulatywnemu charakterowi potwierdzeń ACK w protokole TCP. Jest to przykład opóźnionych potwierdzeń ACK, które są często generowane dla co drugiego pakietu (lub częściej, zgodnie z zaleceniem dokumentu [RFC5681]). Jak zobaczymy, ten konkretny odbiorca (system FreeBSD 5.4) przełącza swój sposób działania między potwierdzaniem pojedynczego pakietu i dwóch pakietów. Oznacza to, że zwracane są średnio dwa potwierdzenia ACK dla każdych trzech wysłanych pakietów z danymi (przy założeniu braku błędów i retransmisji). Opóźnione potwierdzenia ACK i aktualizacje okna omawialiśmy w rozdziale 15.

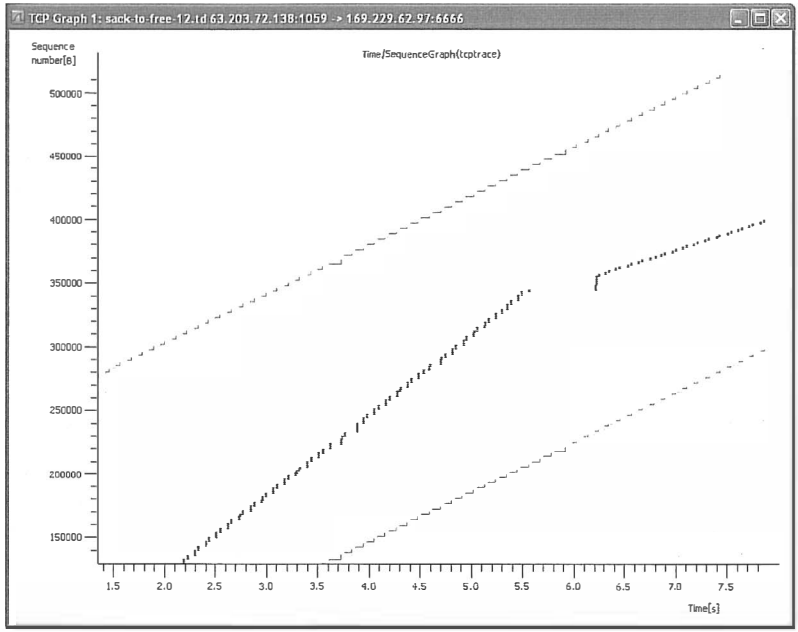
Przychodzące potwierdzenie ACK, które obejmuje dwa pakiety, umożliwia przesunięcie okna przesuwnego nadawcy o dwa pakiety do przodu i w ten sposób pozwala na wysłanie dwóch dodatkowych pakietów do sieci. Ponieważ jednak to połączenie dopiero rozpoczyna swoje działanie i wciąż wykonuje procedurę powolnego startu, przyjście dobrego potwierdzenia ACK sprawia, że nadawca zwiększa swoje okno przeciążenia o jeden pakiet (ten protokół TCP systemu Linux zarządza swoim oknem przeciążenia, używając pakietów jako jednostek). W tym przypadku wartość okna *cwnd* wzrasta z 2 do 3. Ma to ten skutek, że mogą być wysłane w sumie **trzy** pakiety w wyniku przyjścia potwierdzenia ACK. Zostają one wysłane w czasach 0,215, 0,216 i 0,217.

Potwierdzenie ACK przychodzące w czasie 0,264 potwierdza pojedynczy pakiet i pokazuje, że następnym numerem sekwencyjnym oczekiwanym przez odbiorcę jest 4201. Jednak ten pakiet i pakiet następny, z numerem sekwencyjnym 5601, zostały już wysłane i wciąż (z punktu widzenia nadawcy) znajdują się w sieci. Tak więc przyjście potwierdzenia ACK pozwala na zwiększenie okna *cwnd* z 3 do 4, ale ponieważ dwa już wysłane pakiety nie zostały jeszcze potwierdzone, mogą być wysłane tylko dwa dodatkowe pakiety (jeden, ponieważ potwierdzenie ACK przesunęło okno do przodu, drugi, ponieważ odebranie dobrego potwierdzenia ACK pozwoliło na zwiększenie okna *cwnd* o jeden pakiet). Te pakiety zostają wysłane w czasach 0,268 i 0,268 (w odstępie mniejszym niż $^1/_{1000}$ s).

Ten sposób działania w czasie rozpoczynania połączenia jest typowy dla nadawcy wykonującego powolny start i odbiorcy stosującego opóźnione potwierdzenia ACK. Proces nadal działa w tym trybie (każde potwierdzenie ACK uwalnia dwa lub trzy pakiety), dopóki nie zdarzy się coś interesującego w czasie mniej więcej 5,6. Zbadamy to teraz dokładniej.

16.5.2. Przerwa w działaniu nadawcy i lokalne przeciążenie (zdarzenie 1.)

Patrząc na rysunek 16.4, odkrywamy, że po wysłaniu segmentu w czasie 5,512 występuje przerwa do momentu wysłania następnego segmentu z danymi w czasie 6,162. Można to lepiej zobaczyć przy użyciu graficznego mechanizmu powiększania programu Wireshark, co pokazujemy na rysunku 16.6.



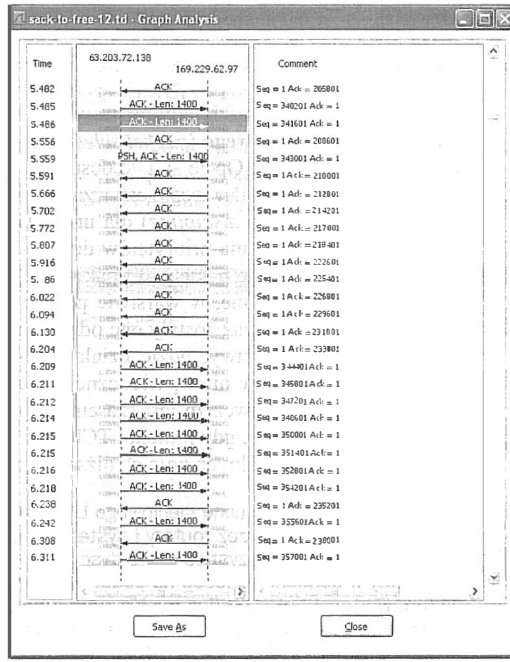
Rysunek 16.6. *Po rozpoczęciu działania przy użyciu procedury powolnego startu następuje w czasie 5,512 przerwa w aktywności połączenia trwająca ok. 512 ms, po czym połączenie wznawia transmisję, wysyłając nagle szybką sekwencję pakietów*

Na tym rysunku widzimy, że nadawca zatrzymał wysyłanie danych i chociaż nic nie wskazuje na obecność retransmitowanego pakietu, szybkość transferu danych okazuje się być po przerwie mniejsza. Dlaczego tak się dzieje? Możemy zbadać to dokładniej, używając raz jeszcze funkcji *Flow graph* (patrz rysunek 16.7).

Nadawczy protokół TCP ewidentnie zaprzestał wysyłania danych w czasie 5,559. Potwierdza to fakt, że ostatni przetransmitowany segment danych przed przerwą ma włączoną flagę PSH, co zwykle wskazuje, że bufor nadawczy został opróżniony. Może być kilka powodów zaistniałej sytuacji, łącznie z możliwością, że system hosta jest zajęty innymi zadaniami, co uniemożliwia aplikacji wysyłającej dane zainicjowanie kolejnej operacji zapisu danych do sieci.

Możemy zauważyć, że omawiana przerwa nie jest początkiem okresu odtwarzania z wykorzystaniem retransmisji, a mimo to nachylenie linii wykresu zmniejsza się po tej przerwie, pokazując zmniejszoną szybkość wysyłania danych. Zbadajmy to zachowanie połączenia dokładniej, aby uzyskać odpowiedź, dlaczego tak się dzieje.

Ostatni numer sekwencyjny wysłany przed przerwą jest równy 343001 + 1400 – 1 = 344 400 i nigdy przedtem nie był wysyłany, więc nie jest efektem retransmisji. Po wysłaniu segmentu w czasie 5,486 (podświetlonego) połączenie posiada swą największą ilość danych pozostających w sieci: 341 601 + 1400 – 205 801 = 137 200 bajtów (98 pakietów).



Rysunek 16.7. *Nadawca robi przerwę w transmisjach w czasie 5,559. W dodatku szybka sekwencja pakietów w czasie 6,209 jest ograniczona do ośmiu z powodu lokalnego przeciążenia. Niektóre implementacje protokołu TCP, podobnie jak używana w przykładzie, ograniczają szybkość wysyłania danych, aby uniknąć przeciążania kolejek na hoście nadawcy*

To oznacza, że wartość *cwnd* wynosi 98 pakietów. Przyjście potwierdzenia ACK w czasie 5,556 wskazuje, że dwa dodatkowe pakiety zostały otrzymane przez odbiorcę. Ostatni pakiet do wysłania przed przerwą zawiera numer sekwencyjny 344400, więc w sieci pozostaje 97 pakietów.

W czasie, gdy aplikacja pauzuje, przychodzi 11 potwierdzeń ACK (potwierdzających na przemian jeden lub dwa pełnowymiarowe segmenty, jak już wcześniej wspomniano). Ostatnie z nich wskazuje, że został odebrany numer sekwencyjny 233800, co oznacza, że teraz w sieci pozostaje 110 600 bajtów danych (79 pakietów). W tym momencie nadawca budzi się i kontynuuje transmisję. W wyniku otrzymania tego potwierdzenia ACK w czasie 6,204 nadawca powinien móc teraz wprowadzić 98 – 79 = 19 dodatkowych pakietów, ale udaje mu się wysłać tylko 8. Ostatni numer sekwencyjny, który jest w stanie wysłać w czasie 6,128, ma wartość 354201 + 1400 – 1 = 355600.

To, co się dzieje z protokołem TCP w tym momencie, nie wynika w sposób bezpośredni i oczywisty z informacji uwidocznionych w śladzie. Można było oczekiwać wysłania 19 pakietów, a wysłanych zostało tylko 8. Powód jest taki, że nadawca zapełnił lokalną (należącą do niższej warstwy stosu protokołów) kolejkę wskutek chwilowego wzrostu ilości niemal jednocześnie wysyłanych pakietów i kolejne pakiety nie mogły być już wysłane. Korzystając z podanego poniżej polecenia systemu Linux i wiedząc, że nasz transfer jest wykonywany za pośrednictwem interfejsu sieciowego ppp0, możemy spróbować sprawdzić, czy jakiś element niższej warstwy nie sprawił problemów protokołowi TCP:

```
Linux% tc -s -d qdisc show dev ppp0
qdisc pfifo_fast 0: bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
Sent 122569547 bytes 348574 pkts (dropped 2, overlimits 0 requeues 0)
```

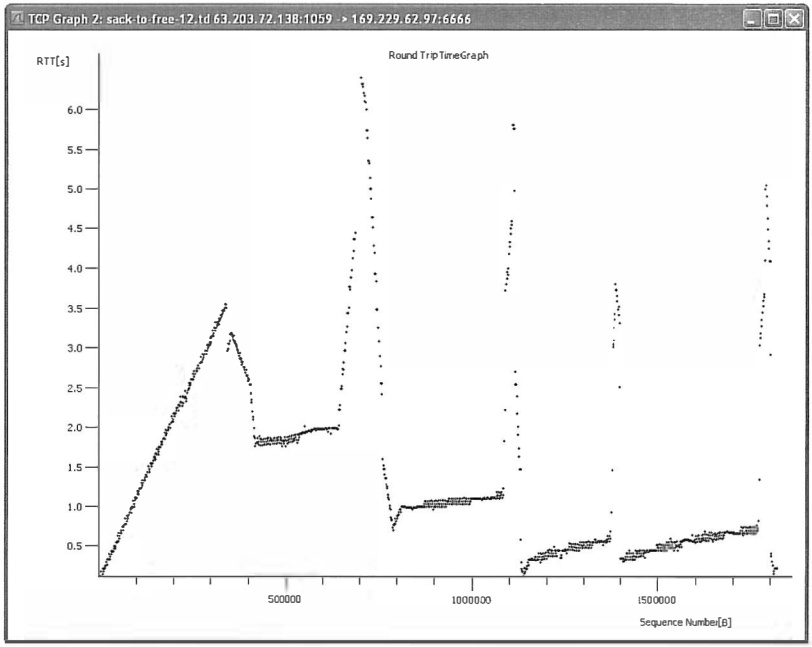
Program `tc` jest używany do administrowania **podsystemem szeregowania pakietów i kontroli ruchu sieciowego** (*packet scheduling and traffic control subsystem*) systemu Linux (patrz [LARTC]). Opcje `-s` i `-d` dostarczają szczegółowych danych statystycznych. Dyrektywa `qdisc show dev ppp0` oznacza, że powinna zostać wyświetlona **dyscyplina kolejkowania** (*queuing discipline*) dla urządzenia `ppp0`, która określa metodę buforowania pakietów i ustalania priorytetów decydujących o kolejności ich wysyłania. Zauważamy dwa odrzucone pakiety. Te pakiety zostały odrzucone nie w sieci, ale w komputerze wysyłającym dane, w warstwie protokołów znajdującej się poniżej protokołu TCP. Co więcej, ponieważ zostały one odrzucone w warstwie leżącej poniżej protokołu TCP, ale powyżej warstwy, gdzie działa mechanizm przechwytujący pakiety, próby transmisji tych pakietów nie są widoczne w śladzie. Odrzucanie transmitowanych pakietów w systemie nadawczym jest czasem nazywane **lokalnym przeciążeniem** (*local congestion*) i pojawia się, gdy protokół TCP generuje dane za szybko w stosunku do możliwości opróżniania lokalnych kolejek niższych warstw.

Podsystem kontroli ruchu sieciowego Linuksa oraz inne mechanizmy priorytetu lub QoS obsługiwane przez routery i systemy operacyjne (np. qWave API Microsoftu, patrz [WQOS]) wykorzystują różne dyscypliny kolejkowania, które mogą porządkować pakiety w odmienny sposób na podstawie zawartości określonych pól w pakietach (np. wartości pola DSCP w nagłówku IP lub numeru portu TCP). Nadanie priorytetu pewnym pakietom (np. pakietom zawierającym dane multimedialne, czystym potwierdzeniom ACK protokołu TCP) może poprawić jakość użytkowania aplikacji interaktywnych w sieciach, które obsługują priorytet. Duża część Internetu nie obsługuje takich priorytetów, ale są one obsługiwane w wielu sieciach LAN oraz w niektórych sieciach IP przedsiębiorstw.

Lokalne przeciążenie jest jednym z kilku powodów, dla których implementacja protokołu TCP w systemie Linux może znaleźć się w stanie **redukcji okna przeciążenia** (CWR, *Congestion Window Reducing*, patrz [SK02]). Rozpoczyna się on od przypisania progowi *ssthresh* wartości *cwnd*/2 oraz nadania oknu *cwnd* wartości równej min(*cwnd*, *rozmiar przelotu* + 1). W stanie CWR nadawca zmniejsza okno *cwnd* o jeden pakiet dla każdych dwóch odebranych potwierdzeń ACK, dopóki wartość *cwnd* nie osiągnie nowego progu *ssthresh* lub nie nastąpi wyjście ze stanu CWR z jakiegoś innego powodu, takiego jak zdarzenie utraty danych. Jest to w zasadzie wspomniany wcześniej algorytm *rate halving* (zmniejszanie szybkości transmisji o połowę). Jest on także wywoływany, kiedy nadawczy protokół TCP otrzymuje sygnał ECN-Echo w odebranym nagłówku TCP (patrz punkt 16.1.1).

Mając tę wiedzę, możemy teraz zrozumieć, co się wydarzyło. Kiedy protokół TCP kontynuuje po przerwie transmisję, jest w stanie wysłać tylko 8 pakietów. Żadne dodatkowe pakiety nie mogą być wysłane z powodu lokalnego przeciążenia, a protokół TCP zostaje przeniesiony do stanu CWR. Natychmiast próg *ssthresh* zostaje zmniejszony do 98/2 = 49 pakietów, a okno *cwnd* otrzymuje rozmiar 79 + 8 = 87 pakietów. Potem protokół pozostaje w stanie CWR, zmniejszając wartość *cwnd* o 1 dla każdych dwóch odebranych potwierdzeń ACK, co prowadzi do redukcji szybkości wysyłania danych, dopóki okno *cwnd* nie zmniejszy się do rozmiaru 66 pakietów w czasie 8,364.

Zmniejszenie szybkości transmisji można również zaobserwować w następujący sposób: patrząc na rysunek 16.6, zauważamy, że nachylenie linii wykresu przed czasem 5,5 wyznacza efektywną szybkość transmisji danych na poziomie ok. 500 kb/s. Jest to wartość wyższa od pojemności łącza w kierunku transferu danych, więc ta dodatkowa, pozorna pojemność jest wynikiem zapełnienia co najmniej jednej kolejki w ścieżce, co prowadzi do zwiększenia czasu RTT z powodu opóźnienia związanego z kolejkowaniem. Możemy użyć funkcji *Statistics/TCP Stream Graph/Round Trip Time Graph* do wizualizacji tego efektu (patrz rysunek 16.8).



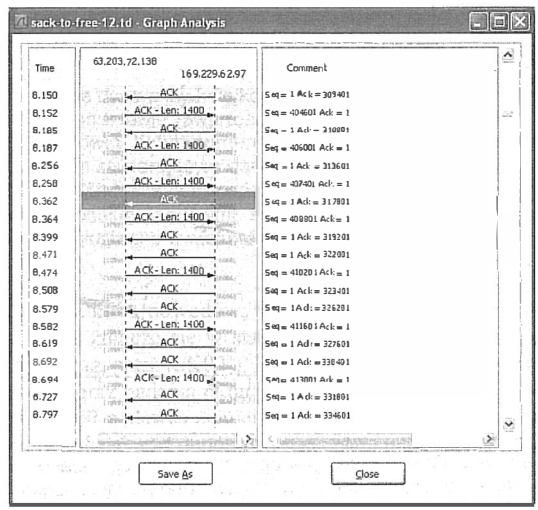
Rysunek 16.8. *Czas transmisji w obie strony według oszacowania nadawcy. Okresy wzrastania czasu RTT (gęste skupienia rosnących wartości) odpowiadają zapełnianiu się buforów z powodu nadwyżki szybkości wysyłania danych w stosunku do szybkości przekazywania danych w routerze znajdującym się w ścieżce. Malejące czasy RTT przedstawiają odwrotny efekt, wynikający ze spowolnienia transmisji przez nadawcę i opróżniania się kolejek*

Na tym rysunku oś Y przedstawia szacunkowy czas RTT wyrażony w sekundach, a oś X reprezentuje numery sekwencyjne. Możemy zauważyć, że w przybliżeniu od numeru 340000 czas RTT zaczyna się zmniejszać. Ten numer sekwencyjny dość dokładnie odpowiada opisanemu wcześniej, ostatniemu numerowi sekwencyjnemu wysłanemu przed przerwą w transmisjach (344400). Zmniejszanie się czasu RTT odpowiada faktowi, że nadawca spowalnia wysyłanie danych, a sieć staje się mniej obciążona (tzn. szybkość, z jaką dane odpływają z sieci, przewyższa szybkość napływu nowego ruchu). Powoduje to opróżnianie się kolejek wewnątrz routerów znajdujących się w sieci, co prowadzi do zmniejszenia się czasu oczekiwania i w konsekwencji do mniejszych wartości czasu RTT.

Redukcja szybkości wysyłania danych trwa nadal w czasie, gdy protokół TCP pozostaje w stanie CWR. Jeśli taki stan rzeczy utrzymywałby się wystarczająco długo, czas RTT zmniejszyłby się w końcu do swojego absolutnego minimum wynoszącego ok. 17 ms. Na ogół protokół TCP nie dopuszcza do tej sytuacji, ponieważ chce utrzymywać stan zapełnienia łącza (*keep the pipe full*), aby wykorzystać maksymalną, aktualnie dostępną pojemność sieci.

16.5.3. Przeciągnięte potwierdzenia ACK i odtwarzanie po lokalnym przeciążeniu

W czasie 8,364, po okresie stopniowego zmniejszania okna *cwnd* spowodowanego początkowo przez wejście TCP w stan CWR, protokół zaczyna w widoczny sposób zmniejszać okno szybciej. Jest to konsekwencja zmiany stosunku między wartością okna *cwnd* a ilością danych pozostających w sieci wskazaną przez potwierdzenie ACK w czasie 8,362 (podświetlone na rysunku 16.9).



Rysunek 16.9. *„Przeciągnięte" potwierdzenie ACK potwierdza numery sekwencyjne zawarte w trzech pakietach. Takie potwierdzenia ACK mogą spowodować impulsowe działanie nadawcy i mogą się pojawić, kiedy inne potwierdzenia ACK zostały zgubione w trakcie transmisji*

Potwierdzenie ACK w czasie 8,362 dotyczy numeru sekwencyjnego 317801, a poprzednio odebrane potwierdzenie ACK odnosi się do numeru sekwencyjnego 313601, co oznacza, że to nowe ACK potwierdza 317 801 – 313 601 = 4200 bajtów (trzy pakiety). Tego rodzaju potwierdzenie jest powszechnie nazywane **przeciągniętym potwierdzeniem ACK** (*stretch ACK*), co oznacza, że potwierdza ono większą ilość danych niż dwukrotny rozmiar największego, dotychczas wysłanego segmentu. Może być spowodowane szeregiem przyczyn, z których najprostszą jest utrata ACK. Zwykle trudno ustalić z całkowitą pewnością przyczynę pojawienia się przeciągniętego potwierdzenia ACK, ale dokładny powód nie jest zazwyczaj ważny. W naszym przykładzie możemy przyjąć, że wcześniejsze potwierdzenie ACK zostało zgubione, i kontynuować badanie zachowania nadawcy. Przyjście tego potwierdzenia powoduje spadek wielkości okna *cwnd* z 68 do 66.

Implementacja protokołu TCP w systemie Linux stara się korygować swoje oszacowanie liczby pakietów pozostających w sieci zawsze wtedy, kiedy odbiera potwierdzenie ACK. (Stara się także weryfikować okno przeciążenia zawsze wtedy, kiedy wysyła segmenty, zgodnie z opisanym wcześniej algorytmem walidacji okna przeciążenia, ale fakt ten nie ma tu znaczenia). Gdy TCP znajduje się w stanie CWR i dochodzi z jakiegoś powodu do zmniejszenia szacunkowej liczby pakietów pozostających w sieci, tak jak w naszym przypadku po odebraniu przeciągniętego potwierdzenia ACK, okno *cwnd* jest korygowane do wartości oszacowania plus 1. Zauważmy, że korekta ta jest dodatkiem do zwykłego zachowania protokołu w stanie CWR, który zmniejsza wartość *cwnd* o 1 dla każdej pary odebranych potwierdzeń ACK. Ogólnie mówiąc, okno *cwnd* jest zmniejszane albo o 1, albo o 0 dla każdego ACK, a następnie *cwnd* otrzymuje wartość min(*rozmiar przelotu* + 1, [być może zmniejszone] *cwnd*). Stan CWR nie przestaje funkcjonować, dopóki okno *cwnd* nie osiągnie progu *ssthresh* lub nie wystąpi jakieś inne zdarzenie, takie jak utrata danych i retransmisja.

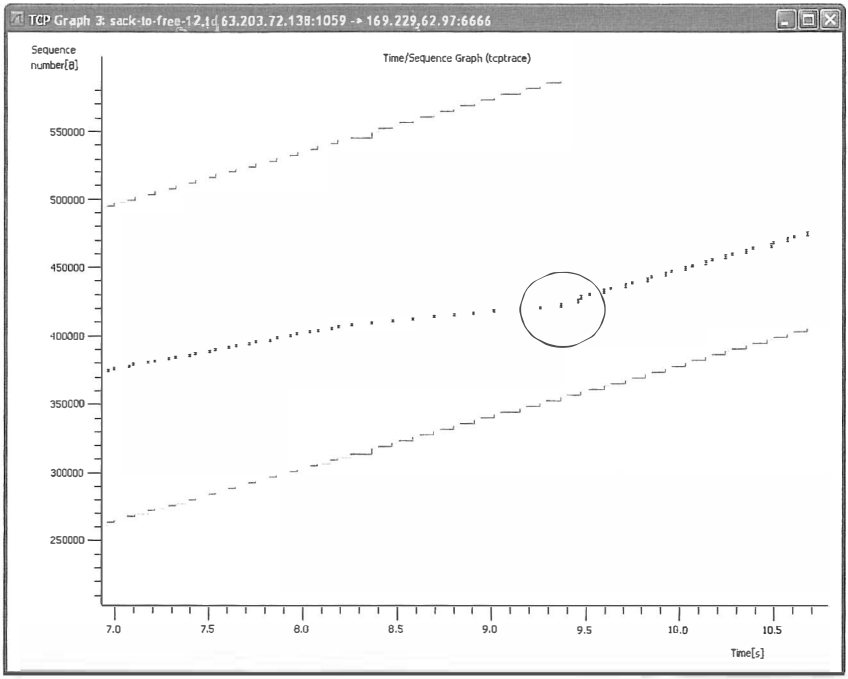
Przed odebraniem przeciągniętego potwierdzenia ACK, w czasie 8,258, pozostaje w sieci 407 401 + 1400 – 313 601 = 95 200 bajtów danych (68 pakietów). Po odebraniu przeciągniętego potwierdzenia ACK liczba pakietów pozostających w sieci zmniejsza się do 65, a okno *cwnd* otrzymuje wartość 66.

Ponieważ szacunkowy rozmiar przelotu i wartość *cwnd* są ściśle związane ze sobą w stanie CWR, a odbiorczy protokół TCP w naszym przykładzie opóźnia potwierdzenia ACK, wynikiem przyjścia pary potwierdzeń ACK jest zmniejszenie okna *cwnd* o 2 i uwolnienie jednego pakietu. Powód tego jest następujący: załóżmy, że przed przyjściem jakiegokolwiek potwierdzenia ACK wartość *cwnd* wynosi c_0, a szacunkowy rozmiar przelotu jest równy $f_0 = c_0$. Kiedy przychodzi pierwsze potwierdzenie ACK (tzn. dla jednego pakietu), $f_1 = f_0 - 1$, a okno *cwnd* zostaje zaktualizowane do wartości $c_1 = \min(c_0 - 1, f_1 + 1) = c_0 - 1$. Kiedy przychodzi drugie potwierdzenie ACK (dla dwóch pakietów, z powodu opóźnionych potwierdzeń ACK), $f_2 = f_1 - 2 = c_0 - 3$, a *cwnd* otrzymuje wartość $c_2 = \min(c_1, f_2 + 1) = \min(c_0 - 1, c_0 - 2) = c_0 - 2$. Ponieważ okno przeciążenia skurczyło się o 2 pakiety, ale w tym okresie zostały potwierdzone trzy pakiety, po odebraniu drugiego potwierdzenia ACK uwolniony zostaje pojedynczy pakiet.

Nadawca wychodzi ze stanu CWR w czasie 9,37, kiedy okno *cwnd* osiąga wartość progu *ssthresh* wynoszącą 49 pakietów. Teraz protokół TCP wraca do swojego normalnego sposobu działania i pracuje dalej, wykonując procedurę unikania przeciążenia (patrz rysunki 16.10 i 16.11).

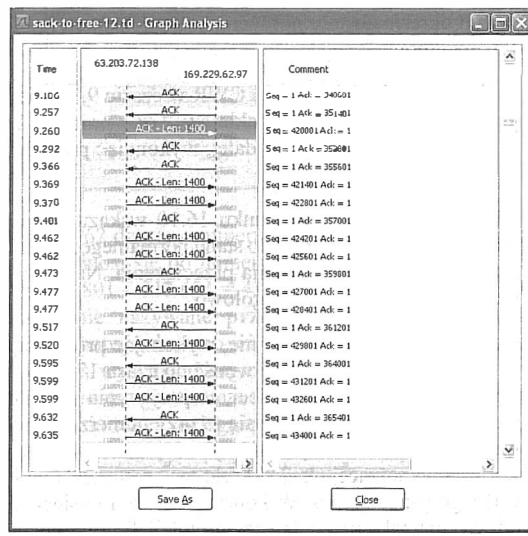
Zakreślone pakiety na rysunku 16.10 wskazują miejsce, gdzie nadawca z powrotem przechodzi ze stanu CWR do stanu normalnego, w którym kontrolę nad jego działaniem przejmuje algorytm unikania przeciążenia. Na rysunku 16.11 pokazujemy to zachowanie nadawcy bardziej szczegółowo.

Nadawca kontynuuje działanie, wykonując procedurę unikania przeciążenia i osiągając stosunkowo stabilną przepływność do czasu 17,232. W tym momencie zaczyna tworzyć się poważne przeciążenie sieci, co przyczynia się do dużego wzrostu czasu RTT. Zgodnie z rysunkiem 16.8 dzieje się to przy numerze sekwencyjnym 720000, gdzie czas RTT wzrasta do ok. 6,5 s — ponad trzykrotny wzrost w stosunku do swojej poprzedniej stabilnej wartości wynoszącej ok. 2 s. Efekt ten często występuje równocześnie z początkiem poważnego przeciążenia. W końcu przeciążenie sieci jest na tyle poważne, że powoduje odrzucenie pakietu. Nadawczy protokół TCP odpowiada swoją pierwszą retransmisją.



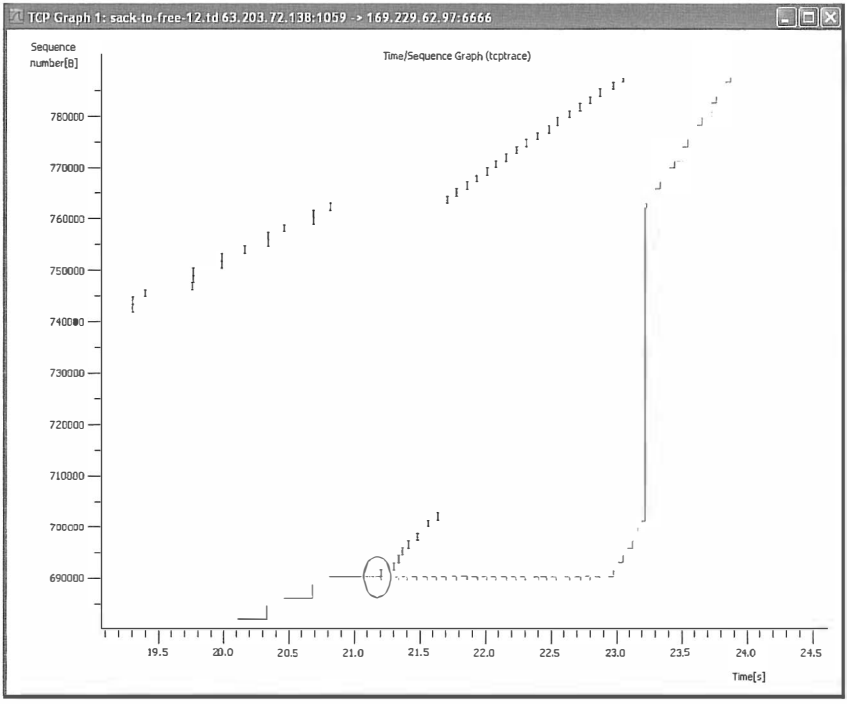
Rysunek 16.10. *Przed czasem 9,369 nadawca wraca do stanu normalnego i wysyła jeden pakiet lub dwa pakiety dla każdego otrzymanego potwierdzenia ACK*

Rysunek 16.11. *Protokół TCP zakończył proces odtwarzania i z powrotem znajduje się w stanie normalnym (czyli działa zgodnie z algorytmem unikania przeciążenia). Wysyła jeden pakiet lub dwa pakiety dla każdego odebranego potwierdzenia ACK*



16.5.4. Szybka retransmisja i odtwarzanie z wykorzystaniem opcji SACK (zdarzenie 2.)

W czasie 21,209, po dramatycznym wzroście mierzonego czasu RTT, obserwujemy pierwszą retransmisję. Możemy to dokładniej zobaczyć, korzystając z powiększenia, co pokazujemy na rysunku 16.12. Pierwsza retransmisja (zakreślona) dotyczy pakietu zaczynającego się od numeru sekwencyjnego 690201, zgodnego z najwyższym dotąd odebranym numerem ACK (także równym 690201). Została uruchomiona przez odebranie pojedynczego, zduplikowanego potwierdzenia ACK zawierającego blok SACK [698601, 700001]. Przypomnijmy sobie, że te liczby pokazują zakres numerów sekwencyjnych już otrzymanych przez odbiorcę. W tym przypadku jest to pojedynczy pakiet.



Rysunek 16.12. *Pierwsza retransmisja (zakreślona) ma miejsce w czasie 21,209. Bloki SACK służą do wskazania nadawcy, które pakiety powinien retransmitować. W sumie, między czasami 21,0 i 22,0 występuje osiem retransmisji*

W czasie 21,209, kiedy ma miejsce retransmisja, największy dotąd wysłany numer sekwencyjny jest równy 761 601 + 1400 – 1 = 763000, a wartość *cwnd* wynosi 52. Wraz z tą szybką retransmisją próg *ssthresh* zostaje zmniejszony z 49 do 26, a protokół wchodzi w stan odtwarzania. Ta implementacja protokołu TCP pozostaje w stanie odtwarzania, dopóki nie odbierze kumulatywnego potwierdzenia ACK dla punktu odtwarzania: numeru sekwencyjnego 763000 (lub wyższego). Dodatkowo okno *cwnd* zostaje zmniejszone do (*rozmiar przelotu* + 1) pakietów. Ponieważ prawdopodobnie nastąpiła utrata danych,

ustalenie rozmiaru przelotu nie jest takie proste. Jest ono wykonywane zgodnie z następującą zależnością:

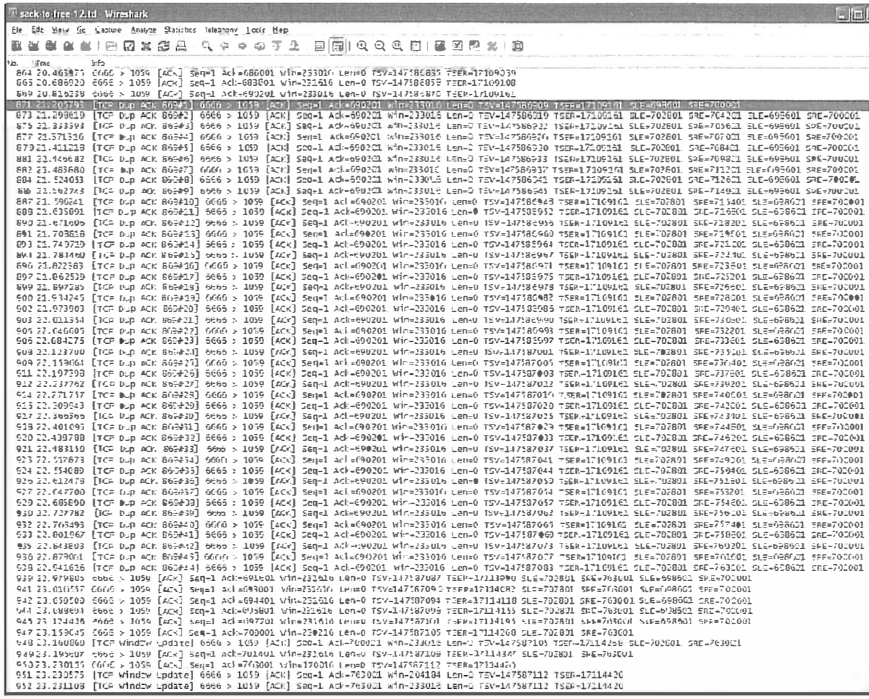
rozmiarpakietu= pakiety_pozostające_w_sieci+

+ pakiety_retransmitowane– pakiety_usunięte

Pierwszy termin po prawej stronie równości reprezentuje wszystkie pakiety wysłane jeden raz przez nadawcę i jeszcze niepotwierdzone przez normalne, kumulatywne pole *ACK* protokołu TCP. Drugi termin określa wszystkie pakiety, które zostały wysłane ponownie (i nie są potwierdzone), a ostatni termin oznacza wszystkie pakiety, których już nie ma w sieci, ale również nie zostały potwierdzone przez podstawowe, kumulatywne potwierdzenie ACK protokołu TCP. Wartość składnika *pakiety_usunięte* musi zostać oszacowana, ponieważ protokół TCP nie dysponuje niezawodnym sposobem bezpośredniego uzyskania tej informacji. Reprezentuje ona sumę wszystkich pakietów (odebranych poza kolejnością) przechowywanych przez odbiorcę i wszystkich pakietów, które zostały utracone w sieci. Przy użyciu opcji SACK można poznać liczbę pakietów przechowywanych u odbiorcy, ale liczba utraconych pakietów nadal musi zostać oszacowana.

Wartość składnika *pakiety_pozostające_w_sieci* wynosi w tym miejscu (763 001 – 690 201) / 1400 = 72 800 / 1400 = 52, a liczba pakietów przechowywanych u odbiorcy wynosi (700 001 – 698 601) / 1400 = 1400 / 1400 = 1 (obliczona na podstawie numerów sekwencyjnych zawartych w bloku SACK). W przypadku włączonego mechanizmu FACK, co ma tu miejsce ze względu na ustawienie domyślne, luki u odbiorcy wywnioskowane na podstawie informacji SACK są uważane za dane utracone. Tak więc w tym przypadku protokół TCP szacuje, że zostało utraconych 698 601 – 690 201 = 8400 bajtów (6 pakietów). Dlatego *rozmiar przelotu* jest równy 52 + 1 – (1 + 6) = 46 pakietów, a okno *cwnd* otrzymuje wartość 47. Znajdując się w stanie odtwarzania, protokół TCP zmniejsza okno *cwnd* o jeden pakiet dla każdych dwóch pakietów, które odbiera, podobnie do sposobu działania w stanie CWR. Po pierwszej retransmisji ma miejsce kolejne siedem retransmisji, a następnie transmisja nowych danych na podstawie informacji zawartych w opcjach SACK przekazywanych w każdym z potwierdzeń przychodzących między czasem 21,2 a czasem 21,7 (patrz rysunek 16.13).

Na tym rysunku dużo informacji normalnie wyświetlanych przez program Wireshark zostało usuniętych, aby wyraźniej było widać opcje SACK w każdym potwierdzeniu ACK. Patrząc na numery sekwencyjne zawarte w blokach SACK (SLE i SRE), widzimy, że przez większość czasu występują dwa aktywne bloki u odbiorcy: [698601, 700001], który zawiera jeden pakiet, i drugi [702801, 763001] (w swoim największym zakresie), który rośnie do 43 pakietów. W czasie odtwarzania ogólny algorytm *rate halving* mający zastosowanie zarówno w stanie CWR, jak i w stanie odtwarzania zmniejsza okno *cwnd* o przynajmniej jeden pakiet dla każdej pary odebranych potwierdzeń ACK. Ponieważ w tym przypadku każde odebrane potwierdzenie ACK potwierdza w efekcie jeden pakiet (przez zwiększenie rozmiaru bloku SACK o jeden pakiet), rozmiar przelotu zmniejsza się o 1, co pozwoliłoby na wysłanie kolejnego pakietu. Ponieważ jednak okno *cwnd* jest także zmniejszane o 1 dla co drugiego potwierdzenia ACK, potrzeba dwóch potwierdzeń ACK do uwolnienia nowego pakietu. Zauważmy, jak to się różni od przypadku związanego ze stanem CWR. W tamtym przypadku niektóre segmenty ACK dostarczały potwierdzenia dla dwóch pakietów, podczas gdy w omawianej sytuacji tylko jeden pakiet jest potwierdzany



Rysunek 16.13. *Odtwarzanie z wykorzystaniem opcji SACK po szybkiej retransmisji. Pakiet 871 zawiera pierwszą opcję SACK użytą w połączeniu. Kolejne potwierdzenia ACK, aż do pakietu 950 (wyłącznie), także zawierają informacje SACK*

(przez opcję SACK) dla każdego przychodzącego segmentu ACK. W ten sposób dla każdej z pokazanych na wykresie transmisji i retransmisji okno *cwnd* jest zmniejszane o 1 po odebraniu każdych dwóch potwierdzeń ACK. W sumie, w ciągu tego okresu odtwarzania okno *cwnd* kurczy się z 47 do 20 pakietów.

Większość segmentów ACK zawierających opcje SACK to zduplikowane potwierdzenia ACK dla numeru sekwencyjnego 690201 (jest ich 44), co pokazuje program Wireshark. Jest pięć dobrych potwierdzeń ACK, które zawierają bloki SACK: [702801, 763001] i [698601, 700001]. Dwa kolejne zawierają tylko jeden blok SACK: [702801, 763001]. Te dobre potwierdzenia ACK nie wyprowadzają nadawcy ze stanu odtwarzania, ponieważ ich numery ACK są wszystkie poniżej numeru sekwencyjnego punktu odtwarzania o wartości 763000; są więc one częściowymi potwierdzeniami ACK, które omawialiśmy wcześniej.

Protokół TCP kończy odtwarzanie związane z szybką retransmisją w czasie 23,301 w momencie przyjścia dobrego potwierdzenia ACK dla numeru sekwencyjnego (765801) większego od punktu odtwarzania. W tym momencie okno *cwnd* ma wartość 20, a wartość progu *ssthresh* wynosi 26, co oznacza, że protokół TCP jest w fazie powolnego startu. Do czasu 23,659, po kilku „podróżach w obie strony", okno *cwnd* osiąga wartość 27, a protokół TCP znajduje się w stanie normalnego działania, pod kontrolą algorytmu unikania przeciążenia. To kończy pierwszy w działaniu nadawcy okres odtwarzania związanego z szybką retransmisją.

16.5.5. Kolejne zdarzenia lokalnego przeciążenia i szybkiej retransmisji

Kolejne cztery zdarzenia stanowią lokalne przeciążenie, szybka retransmisja i dwa dalsze epizody lokalnego przeciążenia. Są one bardzo podobne do typów zdarzeń, które już widzieliśmy, więc zostaną w tym miejscu tylko krótko podsumowane.

16.5.5.1. Ponownie CWR (zdarzenie 3.)

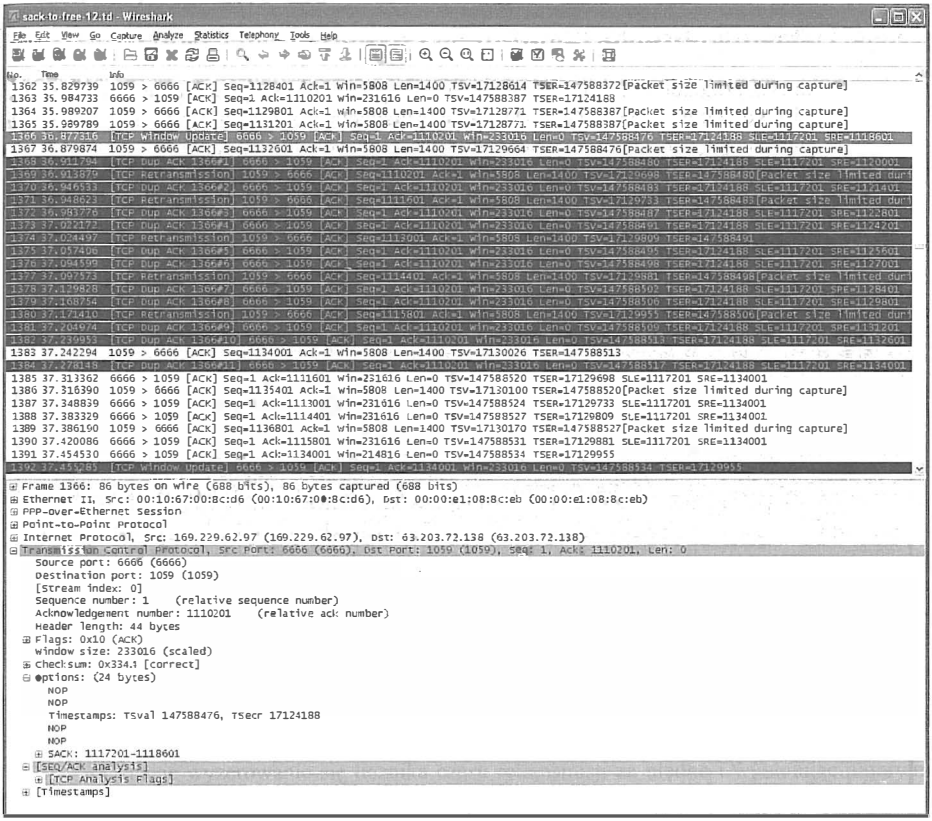
W czasie 30,745 ma miejsce kolejne zdarzenie CWR z powodu lokalnego przeciążenia. W tym momencie w sieci pozostaje 1 090 601 + 1400 – 1 051 401 = 40 600 bajtów (29 pakietów), a rozmiar okna *cwnd* wynosi 31. Powinno to pozwolić na wprowadzenie do sieci dwóch dodatkowych pakietów, ale żaden nie zostaje wprowadzony z powodu lokalnego przeciążenia. W tym szczególnym przypadku okno *cwnd* otrzymuje wartość *rozmiar przelotu* + 1 = 30, a próg *ssthresh* zostaje zmniejszony do 15. Protokół TCP wychodzi ze stanu CWR, kiedy rozmiar okna *cwnd* osiąga wartość *ssthresh*. Dzieje się to w czasie 34,759, po kolejnym znacznym wzroście czasu RTT połączenia.

16.5.5.2. Druga szybka retransmisja (zdarzenie 4.)

W czasie 36,914 dochodzi do kolejnej szybkiej retransmisji, kiedy *cwnd* = 16. Korzystając z podstawowego ekranu programu Wireshark, możemy łatwo dostrzec takie retransmisje (patrz rysunek 16.14).

Na rysunku potwierdzenie ACK przychodzące w czasie 36,878 (pakiet 1366) zawiera blok SACK [1117201, 1118601] oraz numer ACK 1110201. To przenosi TCP systemu Linux do stanu Disorder (nieporządek, nieład), w którym przychodzące pakiety uwalniają każdy po jednym pakiecie (podobnie jak w przypadku ograniczonej transmisji) nowych danych. W tym przypadku uwolnionym pakietem jest pakiet 1367.

Wraz z przyjściem segmentu ACK w czasie 36,912 (pakiet 1368), zawierającego blok SACK [1117201, 1120001] i zduplikowane potwierdzenie ACK, protokół TCP przechodzi do stanu Recovery (odtwarzania) i uruchamia szybką retransmisję w czasie 36,914 (pakiet 1369). Najwyższy dotychczas numer sekwencyjny ma wartość 1 132 601 + 1400 – 1 = 113400. Proces odtwarzania ostatecznie kończy się w czasie 37,455 wraz z przyjściem potwierdzenia ACK zawierającego numer sekwencyjny 1134001 (pakiet 1391). Zauważmy, że bezpośrednio po tym potwierdzeniu ACK następuje aktualizacja okna. W przypadku transferu masowych danych, tak jak w obecnym przykładzie, gdzie okno odbiorcy jest duże w stosunku do iloczynu pasmo-opóźnienie dla sieci, takie aktualizacje nie mają zwykle dużych konsekwencji. Kiedy mamy do czynienia z ruchem interaktywnym, małymi oknami lub serwerami, które tylko od czasu do czasu wykonują odczyty danych z sieci, aktualizacje te mogą stać się dość ważne, co widzieliśmy w rozdziale 15. Kiedy ma miejsce pierwsza retransmisja, próg *ssthresh* zostaje zmniejszony z 16 do 8. Ostatecznie, kiedy kończy się odtwarzanie, *cwnd* = 4, a *ssthresh* = 8. To umieszcza nadawcę w fazie powolnego startu, ponieważ okno *cwnd* jest mniejsze od progu *ssthresh*.



Rysunek 16.14. *Nadawczy protokół TCP systemu Linux wchodzi w stan Disorder (nieporządek, nieład) po odebraniu zduplikowanego potwierdzenia ACK lub potwierdzenia ACK z informacją SACK. Pakiety przychodzące w czasie, gdy protokół znajduje się w tym stanie, uruchamiają transmisję nowych danych. Kolejne zduplikowane potwierdzenia ACK (lub obecność informacji SACK) przenoszą nadawcę do stanu Recovery (odtwarzania), w którym mają miejsce retransmisje*

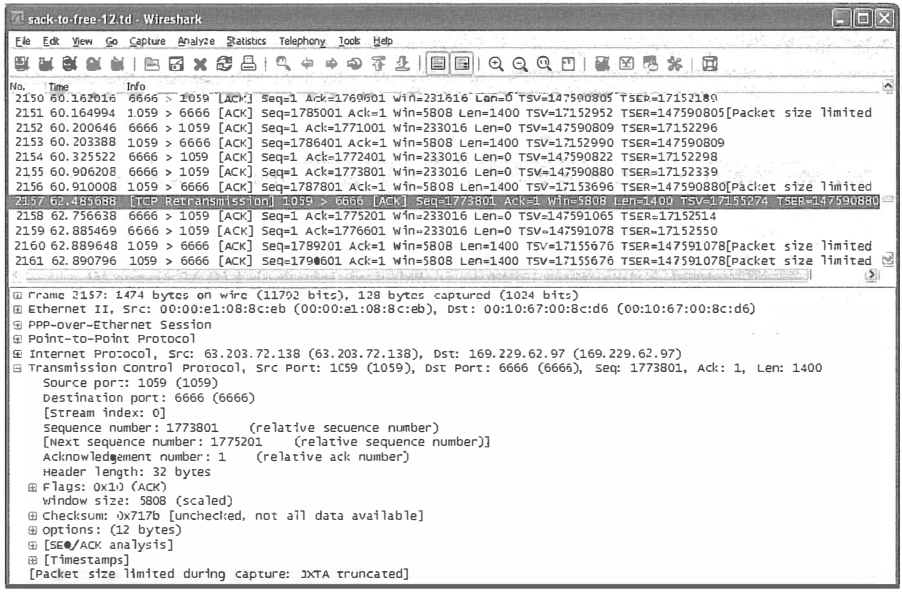
16.5.5.3. Kolejne wystąpienie stanu CWR (zdarzenia 5. i 6.)

Po przyjściu potwierdzenia ACK dla numeru sekwencyjnego 1359401 w czasie 43,356 protokół TCP jeszcze raz wchodzi w stan CWR z powodu lokalnego przeciążenia, kiedy próbuje wysłać kolejne pakiety. To ostatecznie zmniejsza próg *ssthresh* do 8, a okno *cwnd* przybiera wartość 15. Drugie niepowodzenie w transmisji, do którego dochodzi w stanie CWR, obniża wartość progu *ssthresh* do 12. Stan CWR zostaje zakończony, gdy *cwnd* = 7, a *ssthresh* = 8.

Kolejne pojawienie się lokalnego przeciążenia w czasie 59,652 wymusza przejście protokołu TCP do stanu CWR w sytuacji, gdy *cwnd* = 19, a *ssthresh* = 10. W tym przypadku stan CWR zostaje przerwany przez przeterminowanie, które umieszcza TCP w stanie Loss (utraty danych). Jest to przykład nowego typu zdarzenia, które chcemy zbadać.

16.5.6. Przeterminowania, retransmisje i wycofywanie zmian okna cwnd

Chociaż protokół TCP utrzymuje licznik czasu oczekiwania przed retransmisją, na wypadek gdy utrata danych nie może zostać naprawiona przez szybką retransmisję, nie widzieliśmy go jeszcze w działaniu. Jest to szczęśliwa okoliczność, ponieważ na ogół, gdy występuje przeterminowanie, połączenie doznaje znaczącego przeciążenia i ma problemy z wydajnością. W następnym fragmencie śladu, pokazanym na rysunku 16.15, widzimy, w jaki sposób protokół TCP radzi sobie z sytuacją, gdy licznik czasu retransmisji pokazuje przeterminowanie.



Rysunek 16.15. *Nadawca doświadcza swojego pierwszego przeterminowania w sytuacji, gdy czas RTO = 1,57 s. W tym przypadku nadawca ustala, że przeterminowanie jest fałszywe, i wycofuje modyfikacje zmieniające stan kontroli przeciążenia*

16.5.6.1. Pierwsze przeterminowanie (zdarzenie 7.)

Do retransmisji dochodzi w czasie 62,486 (pakiet 2157) dla numeru sekwencyjnego 1773801 (podświetlona pozycja na rysunku 16.15). Bezpośrednio przedtem nie ma żadnego śladu zduplikowanych potwierdzeń ACK lub opcji SACK.

Widzimy na rysunku 16.15, że w czasie 62,486 upłynęło ok. 1,58 s od momentu odebrania ostatniego potwierdzenia ACK, ale według rysunku 16.8 szacunkowy czas RTT wynosi w tym momencie tylko ok. 800 ms. Wobec tego możemy dojść do wniosku, że retransmisja jest wynikiem wyzerowania się licznika czasu retransmisji. Zdarzenie to umieszcza protokół TCP w stanie Loss, co zwykle powoduje drastyczną redukcję okna *cwnd* i w efekcie przenosi protokół TCP do etapu powolnego startu. W naszym przypadku TCP wykonuje ustawienia: *cwnd* = 1 i *ssthresh* = 5, co zgodnie z oczekiwaniem

umieszcza protokół w fazie powolnego startu. Przeterminowanie wymusza również usunięcie wszystkich przechowywanych informacji dotyczących opcji SACK. Jednakże odbiorca nie przestaje wysyłać informacji SACK, więc nadawca może wciąż korzystać z nowych informacji SACK, które odbiera.

Protokół TCP ma „zapomnieć" swoją wiedzę pochodzącą z odebranych informacji SACK, kiedy doświadcza przeterminowania, ze względu na możliwość unieważnienia przez odbiorcę dostarczonych wcześniej informacji SACK. Sugeruje to dokument [RFC2018] z powodu (nieokreślonej) możliwości, że odbiorca mógłby chcieć uregulować swoje buforowanie przez usunięcie danych poza kolejnością, które zostały wcześniej nagromadzone. Chociaż nie jest to częste, takie działanie jest dozwolone. Kiedy odbiorca unieważnia wcześniej przekazane informacje, ma obowiązek umieścić informację o ostatnio odebranych blokach danych w pierwszym bloku SACK generowanych przez siebie potwierdzeń ACK, nawet jeśli dane zostały usunięte. Natomiast należy zaprzestać przekazywania informacji o danych, które już nie są przechowywane przez odbiorcę, w dodatkowych blokach SACK.

Co najciekawsze, w tym przypadku działanie związane z obsługą przeciążenia zostało **wycofane**. Jak zauważyliśmy wcześniej, algorytm odpowiedzi Eifel może zostać wywołany, kiedy protokół TCP uważa, że przeterminowanie skutkujące retransmisją było błędne. W tym przypadku przeterminowanie zostało uznane za błędne na podstawie informacji przekazanej w znaczniku czasu. Potwierdzenie ACK odebrane w czasie 62,757 dla numeru sekwencyjnego 1775201 (pakiet 2158) zawiera opcję TSOPT z polem TSER o wartości 17152514. Jednakże retransmitowany segment zawiera w polu TSV wartość 17155274. Ponieważ pole TSER w potwierdzeniu ACK obejmującym retransmitowany segment wskazuje czas (w znaczeniu przyjętym w znacznikach czasu) **wcześniejszy** niż czas retransmisji, luka, którą miała wypełnić retransmisja, nie była w rzeczywistości luką. Natomiast wyzerowanie się licznika czasu retransmisji musiało być błędne.

Uznając wyzerowanie się licznika czasu retransmisji za błędne i wywołując algorytm odpowiedzi podobny do algorytmu Eifel, protokół TCP przywraca poprzednie wartości parametrów *cwnd* i *ssthresh*, w obydwu przypadkach równe 10, i natychmiast przechodzi do stanu normalnego działania. To uaktywnia algorytm unikania przeciążenia i protokół TCP kontynuuje pracę bez większego zamieszania.

16.5.6.2. Szybka retransmisja (zdarzenie 8.)

Przyjście zduplikowanego potwierdzenia ACK dla numeru sekwencyjnego 1789201 zawierającego blok SACK [1792001, 1793401] w czasie 67,510 (pakiet 2179) przenosi ponownie protokół TCP do stanu Disorder. Największy numer sekwencyjny wysłany do momentu wejścia w ten stan ma wartość 1806000. Dodatkowe przychodzące potwierdzenia z opcją SACK powodują przejście do stanu Recovery i uruchomienie kolejnej szybkiej retransmisji w czasie 67,550 dla numeru sekwencyjnego 1789201 (pakiet 2182). To zmniejsza próg *ssthresh* do 5 pakietów, a okno *cwnd* zaczyna kurczyć się, aż także osiąga wartość 5. Odtwarzanie kończy się wraz z przyjściem potwierdzenia ACK w czasie 67,916 zawierającego numer ACK 1806001 (pakiet 2197).

16.5.6.3. Jeszcze jedno wystąpienie stanu CWR (zdarzenie 9.)

Występuje kolejne zdarzenie lokalnego przeciążenia w czasie 77,121, gdy *cwnd* = 18. To powoduje ustawienie progu *ssthresh* = 9 i umieszcza protokół TCP ponownie w stanie CWR. Jednak redukcja okna *cwnd* w stanie CWR zostaje tym razem wcześniej przerwana przez przeterminowanie, gdy wartość *cwnd* została zmniejszona o 1, do 8 pakietów.

16.5.6.4. Drugie przeterminowanie (zdarzenie 10.)

Kolejne przeterminowanie uruchamia retransmisję w czasie 78,515 dla numeru sekwencyjnego 2175601 (nieprzedstawioną na rysunku). Powoduje to ustawienie *cwnd* = 1; wartość progu *ssthresh* nadal wynosi 9, a retransmitowany segment zawiera opcję TSOPT z polem TSV o wartości 17171306. Tak jak w przypadku zdarzenia 7. (pierwszego przeterminowania), działanie związane z obsługą przeciążenia zostaje także wycofane na skutek przyjścia potwierdzenia ACK w czasie 80,093 dla numeru sekwencyjnego 2179801 (pakiet 2641) zawierającego w polu TSER opcji TSOPT wartość 17169948. Kiedy to się dzieje, szacunkowa wartość *rozmiaru przelotu* wynosi 2 184 001 + 1400 – 2 179 801 = 5600 bajtów (cztery pakiety). Jeśli okno *cwnd* byłoby natychmiast przywrócone do stanu sprzed przeterminowania (do 8 pakietów), pozwoliłoby to na natychmiastowe wprowadzenie czterech pakietów do sieci. Takie postępowanie uważa się za niepożądane, ponieważ może ono doprowadzić do większych zmian ilości odrzuconych pakietów za przyczyną nagłych wzrostów szybkości transmisji.

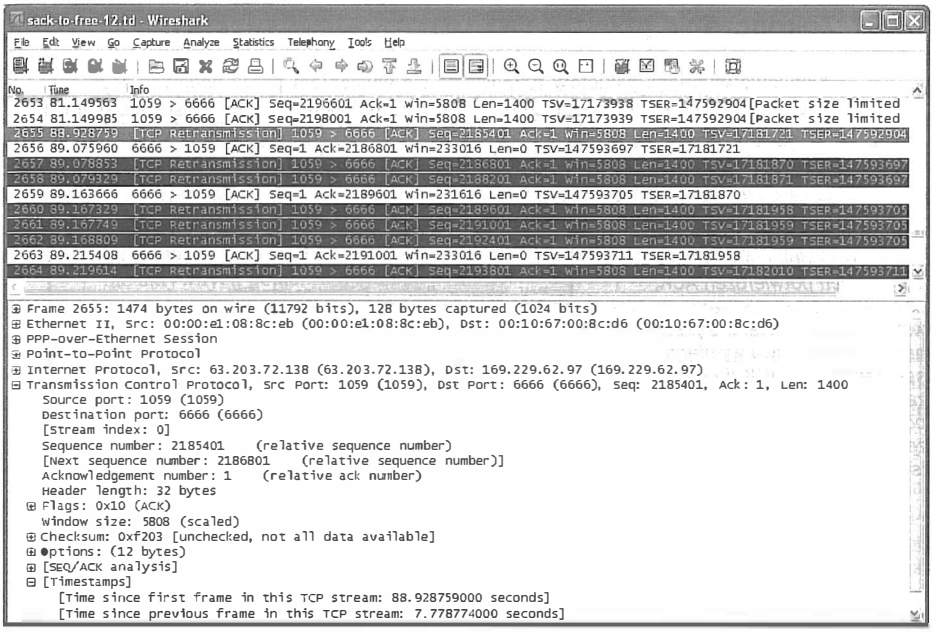
Aby zapobiec takim nierównomiernie rozłożonym transmisjom, ta linuksowa implementacja protokołu TCP dysponuje procedurą moderacji okna przeciążenia (*congestion window moderation*), która ogranicza maksymalną ilość pakietów wygenerowanych w odpowiedzi na pojedyncze potwierdzenie ACK do wartości ***maxburst*** (*maximum burst* — maksymalna seria transmisji) wynoszącej w naszym przykładzie 3 pakiety. Dlatego w tym przypadku okno *cwnd* otrzymuje wartość (*rozmiar przelotu* + *maxburst*) = 4 + 3 = 7. Ta regulacja jest zbliżona do parametru o tej samej nazwie zaproponowanego dla protokołu TCP i przebadanego przy użyciu symulatora sieci NS-2. Symulator ten jest szeroko używany w badaniu i opracowywaniu algorytmów protokołu TCP (patrz [NS2]).

16.5.6.5. Przeterminowanie i końcowe odtwarzanie (zdarzenie 11.)

W czasie 88,929 wyzerował się licznik czasu retransmisji i dochodzi do retransmisji dla numeru sekwencyjnego 2185401, co zostało przedstawione na rysunku 16.16.

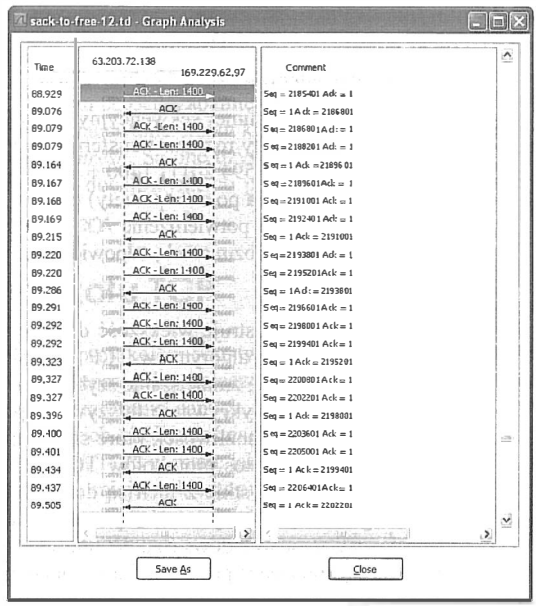
Zerujący się licznik czasu przenosi nadawcę do fazy powolnego startu z progiem *ssthresh* = 5. Tym razem protokół TCP nie może wycofać przeterminowania, więc okno *cwnd* otrzymuje wartość 1 i przebiega procedura powolnego startu. Można to zobaczyć wyraźniej w śladzie przepływu (patrz rysunek 16.17).

Retransmisja dla numeru sekwencyjnego 2185401 jest podświetlona. W następstwie retransmisji widzimy sposób działania typowy dla procedury powolnego startu, jaki mogliśmy zobaczyć podczas rozpoczynania połączenia, kiedy każde przychodzące potwierdzenie ACK uwalnia dwa lub trzy pakiety, zależnie od tego, ile pakietów zostało objętych potwierdzeniem ACK. Od czasu 89,434, kiedy okno *cwnd* osiąga próg *ssthresh* równy 5, TCP kontynuuje działanie zgodne z procedurą unikania przeciążenia.



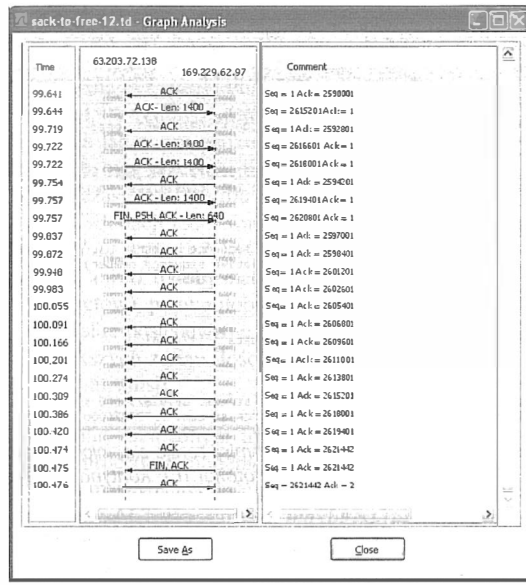
Rysunek 16.16. *Licznik czasu retransmisji zeruje się, inicjując retransmisję na podstawie przeterminowania, która nie może zostać wycofana. Protokół TCP kontynuuje działanie, wykonując powolny start*

Rysunek 16.17.
Jak pokazuje program Wireshark, po przeterminowaniu powodującym retransmisję wykonywana jest procedura powolnego startu. Każde przychodzące potwierdzenie ACK uwalnia dwa lub trzy pakiety



16.5.7. Zakończenie połączenia

Końcowa wymiana pakietów rozpoczyna się wysłaniem przez nadawcę segmentu FIN w czasie 99,757. Po tej transmisji przychodzi 13 potwierdzeń ACK, a następnie segment FIN odbiorcy. Ostatni pakiet (końcowe potwierdzenie ACK) zostaje wysłany w czasie 100,476. Ta wymiana pakietów została przedstawiona na rysunku 16.18.



Rysunek 16.18. *W trakcie procedury zamknięcia połączenia odbiorca generuje 13 czystych potwierdzeń ACK, aby wskazać, że odebrał wszystkie dane wysłane przez nadawcę. Końcowa wymiana segmentów FIN-ACK finalizuje zamknięcie drugiego kierunku połączenia. Zauważmy, że segmenty FIN zawierają istotne numery ACK*

Największy wysłany numer sekwencyjny ma wartość 2 620 801 + 640 – 1 = 2621440, taką samą jak całkowity rozmiar transferu, 2,5 MB. W czasie 99,757 pozostaje w sieci (2 619 401 + 1400 – 2 594 201) / 1400 + 1 = 20 pakietów. Przyjście 13 potwierdzeń ACK (z których 7 potwierdza po dwa pakiety) obejmuje całe okno (2 · 7) + (13 – 7) = 20 pakietów. Zauważmy, że potwierdzenie ACK przychodzące w czasie 100,474 potwierdza ostatnie dwa pakiety o rozmiarach, odpowiednio, 1400 i 640 bajtów: 2 621 442 – 2 619 401 = 1400 + 640.

Ten długi przykład ilustruje większość dotąd opisanych algorytmów i obejmuje różne aspekty podstawowych algorytmów TCP (powolnego startu, unikania przeciążenia): selektywne potwierdzanie, zmniejszanie szybkości transmisji o połowę oraz pewne nowsze procedury, takie jak wykrywanie fałszywych przeterminowań RTO. Teraz omówimy pewne modyfikacje i możliwości, które są mniej rozpowszechnione, bardziej teoretyczne lub bardzo nowe. Stos protokołów TCP systemu Linux implementuje wiele z tych procedur, ale nie wszystkie z nich są domyślnie włączone. Często mała zmiana przy użyciu programu `sysctl` wystarcza, by z nimi poeksperymentować. Nowsze wersje stosu protokołów systemu Windows (tzn. Windows Vista i późniejsze) również implementują ulepszenia wykraczające poza dotąd omawiane mechanizmy.

16.6. Współdzielenie stanu przeciążenia

W dotychczasowej analizie i ostatnio przedstawionym przykładzie skupiliśmy się na tym, jak pojedyncze połączenie TCP przystosowuje się do przeciążenia występującego w ścieżce transmisji. Jeżeli później są nawiązywane inne połączenia między tymi samymi hostami, te kolejne połączenia zazwyczaj muszą ustalić swoje własne wartości progu *ssthresh* i okna *cwnd*, co wymaga pewnego czasu, jak pisaliśmy wcześniej. W wielu przypadkach następujące po sobie połączenia mogłyby potencjalnie dowiedzieć się o tych wartościach od wcześniejszych połączeń z tymi samymi hostami lub od innych aktualnie aktywnych połączeń z tymi samymi hostami. Ten pomysł wiąże się ze współdzieleniem stanu przeciążenia przez wiele połączeń na tym samym komputerze. Wczesny opis zawarty w dokumencie [RFC2140] przedstawia możliwy sposób jego realizacji. Praca ta zwraca uwagę na różnicę między pojęciem *temporal sharing* (współdzielenie w czasie; nowe połączenia współdzielą informacje z innymi połączeniami, które są aktualnie zamknięte — w stanie `CLOSED`) a pojęciem *ensemble sharing* (współdzielenie w zespole; nowe połączenia współdzielą stan z innymi aktywnymi połączeniami).

W usiłowaniu uogólnienia tego pomysłu i rozciągnięcia go na protokoły i aplikacje inne niż TCP dokument [RFC3124] opisuje moduł oprogramowania o nazwie *Congestion Manager* (menedżer przeciążenia), który oferuje lokalną usługę systemu operacyjnego dostępną dla implementacji protokołów, umożliwiającą uzyskanie takich informacji jak współczynnik strat dla ścieżki, szacunkowe przeciążenie, czas RTT itd. dla poszczególnych hostów docelowych.

W systemie Linux zamysł ten jest udostępniony w tym samym podsystemie, który zawiera informacje związane z wyznaczaniem tras, i jest znany pod nazwą „mierniki punktów docelowych" (*destination metrics*), które poznaliśmy w rozdziale 15. Mierniki te są włączone (ale były wyłączone w naszym długim przykładzie przez przypisanie wartości 1 zmiennej `net.ipv4.tcp_no_metrics_save` za pomocą polecenia `sysctl`. Kiedy połączenie TCP przechodzi do stanu `CLOSED`, zapamiętywane są następujące informacje: miary czasu RTT (*srtt* i *rttvar*), oszacowanie zmian kolejności pakietów i zmienne kontroli przeciążenia, czyli *cwnd* i *ssthresh*. Są one używane, kiedy uruchamiane są nowe połączenia z tym samym hostem docelowym, co pomaga w inicjacji odpowiednich parametrów pomiarowych.

16.7. Przyjazność protokołu TCP

Ponieważ TCP jest dominującym protokołem warstwy transportowej w Internecie, powszechnie występuje sytuacja, w której kilka połączeń TCP współdzieli jeden lub więcej routerów w swojej ścieżce dostarczania danych. Jeśli nawet nie zawsze dzielą one pasmo równo między siebie w takich okolicznościach, to przynajmniej reagują na dynamiczne własności innych połączeń TCP, w miarę jak się pojawiają i ustępują w czasie. Nie ma jednak gwarancji, że ma to miejsce, kiedy protokół TCP rywalizuje o dostępne pasmo z innymi (różnymi od TCP) protokołami lub kiedy współzawodniczy z protokołem TCP używającym jakiegoś alternatywnego zestawu parametrów kontrolujących jego okno przeciążenia.

W celu dostarczenia wskazówek dla projektantów protokołów, zmierzających do uniknięcia „nieczystego" współzawodnictwa w transmisjach TCP w sytuacji wspólnego działania w Internecie, naukowcy opracowali limit nazywany „kontrolą szybkości transmisji opartą na równaniu" (*equation-based rate control*), który nakłada ograniczenie na szerokość pasma używanego przez konwencjonalny protokół TCP działający w konkretnym środowisku. Metoda ta nosi nazwę „kontroli przyjaznej szybkości transmisji w TCP" (TFRC, *TCP Friendly Rate Control*, patrz [RFC5348] [FHPW00]). Została zaprojektowana w celu wprowadzenia limitu szybkości transmisji opartego na kombinacji parametrów połączenia i czynników środowiskowych, takich jak czas RTT i stopień utraty pakietów. Daje ona także bardziej stabilny profil wykorzystania pasma w stosunku do konwencjonalnego TCP, więc będzie prawdopodobnie odpowiednia dla aplikacji stosujących przesyłanie strumieniowe, które używają umiarkowanie dużych pakietów (np. transfer danych wideo). W metodzie TFRC do wyznaczenia szybkości przesyłania użyto następującego równania:

$$X = s/\left(R\sqrt{2bp/3}\right) + 3pt_{RTO}\left(1 + 32p^2\right)\sqrt{3bp/8} \qquad [2]$$

W powyższym równaniu X oznacza limit przepływności (w bajtach na sekundę), s jest rozmiarem pakietu (w bajtach, z wyłączeniem nagłówków), R to czas RTT (w sekundach), p oznacza stosunek liczby zdarzeń utraty pakietu do ogólnej liczby wysłanych pakietów, mieszczący się w przedziale [0, 1], t_{RTO} reprezentuje czas oczekiwania na retransmisję (w sekundach), a b jest maksymalną liczbą pakietów potwierdzanych przez pojedyncze potwierdzenie ACK. Zaleca się, aby wartość t_{RTO} była równa $4R$, a rekomendowana wartość b wynosi 1.

Szybkość transmisji protokołu TCP może być wyrażona w inny sposób, na podstawie tego, jak protokół koryguje rozmiar swojego okna w wyniku odebrania dobrego potwierdzenia ACK w czasie wykonywania procedury unikania przeciążenia. Z wcześniejszej analizy pamiętamy, że standardowy protokół TCP w trakcie używania algorytmu unikania przeciążenia zwiększa okno *cwnd* addytywnie, o składnik $1/cwnd$ dla każdego przychodzącego dobrego potwierdzenia ACK i zmniejsza je multiplikatywnie, stosując czynnik dzielący je na pół po wystąpieniu zdarzenia utraty pakietu. Nazywa się to kontrolą przeciążenia przez addytywne zwiększanie / multiplikatywne zmniejszanie okna (AIMD, *Additive Increase/Multiplicative Decrease*). Możemy teraz przedstawić uogólnione równania unikania przeciążenia z zastosowaniem AIMD, w których wartości 1 i $^1/_2$ zastępujemy odpowiednio przez zmienne a i b:

$$cwnd_{t+1} = cwnd_t + a / cwnd_t$$
$$cwnd_{t+1} = cwnd_t - b \cdot cwnd_t$$

Na podstawie wyników przedstawionych w pracy [FHPW00] możemy z powyższych równań wyprowadzić następującą zależność, określającą szybkość transmisji protokołu TCP w pakietach na okres RTT:

$$T = \frac{\sqrt{\frac{a(2-b)}{2b}}}{\sqrt{p}} \qquad [3]$$

Dla normalnego protokołu TCP, gdzie $a = 1$ i $b = 0{,}5$, powyższy wzór upraszcza się do postaci: $T = 1{,}2/\sqrt{p}$, znanej jako **uproszczona standardowa funkcja odpowiedzi protokołu TCP** (*simplified standard TCP response function*). Wiąże on szybkość transmisji protokołu TCP (regulację okna *cwnd*) ze stopą utraty pakietów, z jaką TCP ma do czynienia, bez uwzględnienia retransmisji na skutek przeterminowania . Kiedy protokół TCP nie jest ograniczony przez inne czynniki (bufory nadawcy lub odbiorcy, skalowanie okna itd.), ta zależność wyznacza wydajność protokołu TCP w łagodnych środowiskach operacyjnych.

Każda zmiana w funkcji odpowiedzi protokołu TCP ewidentnie wpływa na sposób, w jaki ten protokół (lub inny protokół implementujący podobny mechanizm kontroli przeciążenia) współzawodniczy ze standardowym protokołem TCP. Dlatego też nowo proponowane mechanizmy kontroli przeciążenia są zazwyczaj analizowane przy użyciu miary **względnej bezstronności** (*relative fairness*). Względna bezstronność określa stosunek szybkości transmisji protokołu używającego zmodyfikowanego mechanizmu kontroli przeciążenia do szybkości standardowego protokołu TCP jako funkcję stopy utraty pakietów. Jest to silny wskaźnik stopnia bezstronności takich zmodyfikowanych mechanizmów w odniesieniu do współdzielenia pasma dostępnego we wspólnej ścieżce Internetu.

Zauważmy, że zrozumienie tych równań jest tylko pierwszym krokiem w tworzeniu reżymu regulacji szybkości transmisji, który współzawodniczy w bezstronny sposób ze standardowym protokołem TCP. Szczegóły implementacji metody TFRC w dowolnym konkretnym protokole mogą być subtelne i obejmować takie kwestie jak prawidłowy pomiar czasu RTT, stopa zdarzeń utraty pakietu i rozmiar pakietu. Kwestie te są analizowane szczegółowo w dokumencie [RFC5348].

16.8. TCP w szybkich środowiskach

W szybkich sieciach o dużych iloczynach BDP (np. w sieciach WAN o przepustowości 1 Gb/s lub większej), konwencjonalny protokół TCP może nie działać wydajnie, ponieważ używany w nim algorytm zwiększania okna (szczególnie algorytm unikania przeciążenia) potrzebuje długiego czasu, by okno stało się wystarczająco duże, żeby osiągnąć stan nasycenia ścieżki sieciowej. Mówiąc inaczej, TCP może nie wykorzystać szybkości sieci nawet wtedy, gdy nie ma żadnego przeciążenia. Kwestia ta wynika głównie ze sposobu działania procedury unikania przeciążenia polegającej na stałym, addytywnym zwiększaniu okna. Jeśli weźmiemy pod uwagę protokół TCP używający 1500-bajtowych pakietów przesyłanych przez długie łącze o przepustowości 10 Gb/s, to liczbę segmentów w sieci potrzebną do pełnego wykorzystania dostępnej szerokości pasma możemy określić na ok. 83 000, przy założeniu braku utraconych pakietów lub błędów w pięciu miliardach pakietów. Przy czasie RTT równym 100 ms potrzeba ok. 1,5 godziny do osiągnięcia tej liczby. Aby zaradzić tej niewydolności, szereg badaczy i projektantów bada sposoby takiej modyfikacji protokołu TCP, aby pracował wydajniej w tego typu sieciach, zachowując jednocześnie pewien stopień bezstronności w stosunku do standardowego protokołu TCP, szczególnie w przypadku bardziej rozpowszechnionych wolniejszych środowisk.

16.8.1. Protokół HighSpeed TCP (HSTCP) i ograniczony powolny start

Eksperymentalne specyfikacje szybkiego protokołu TCP (HSTCP, *HighSpeed TCP*) zawarte w dokumentach [RFC3649] i [RFC3742] proponują zmianę standardowego sposobu działania protokołu TCP w sytuacji, gdy okno przeciążenia jest większe od bazowej wartości *Low_Window* (dolny limit okna), równej, zgodnie z sugestią, 38 segmentom o rozmiarze MSS. Wartość ta odpowiada stopie utraty pakietów wynoszącej 10^{-3}, zgodnie z wcześniej przedstawioną, uproszczoną funkcją odpowiedzi protokołu TCP. Funkcja ta wygląda jak funkcja liniowa na wykresie podwójnie logarytmicznym przedstawiającym szybkość transmisji w zależności od stopy utraty pakietów, więc w rzeczywistości jest funkcją zgodną z prawem potęgi.

Funkcje, których wykres w skali podwójnie logarytmicznej tworzy linię prostą, są funkcjami wyrażającymi **prawo potęgi**. Ich równania mają postać $y = ax^k$, co oznacza, że $\log y = \log a + k \log x$ (a i k są stałymi). To ostatnie równanie daje linię prostą o nachyleniu k na wykresie podwójnie logarytmicznym.

Aby skonstruować wymagany typ funkcji zgodnej z prawem potęgi, wybieramy dwa punkty i tworzymy równanie, które opisuje łączącą je linię. Oznaczmy te dwa punkty jako (p_1, w_1) i (P_0, W_0), gdzie $w_1 > W_0 > 0$ oraz $0 < p_1 < P_0$. Na wykresie liniowym otrzymalibyśmy linię o nachyleniu $(w_1 - W_0)/(p_1 - P_0)$, ale na wykresie podwójnie logarytmicznym powstaje linia o nachyleniu $S = (\log w_1 - \log W_0)/(\log p_1 - \log P_0)$. Następnie na podstawie równania podanego w uwadze mamy $w = Cp^S$ i potrzebujemy jakiegoś punktu, powiedzmy (P_0, W_0), aby wyznaczyć wartość C. Po pewnych przekształceniach algebraicznych okazuje się, że $C = P_0^{-S} W_0$, co oznacza, że $w = p^S P_0^{-S} W_0$.

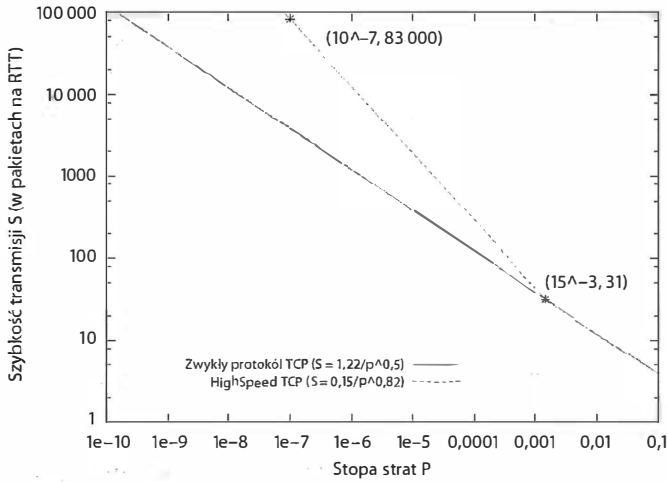
Na rysunku 16.19 widzimy wykres funkcji odpowiedzi konwencjonalnego protokołu TCP oraz proponowanej funkcji odpowiedzi dla protokołu HSTCP wyznaczonej przez punkt $(P_0, W_0) = (0{,}0015, 31)$ i nachylenie $S = -0{,}82$. Zauważmy, że dla większych stóp utraty pakietów (większych niż ok. 0,001) funkcje odpowiedzi są takie same, więc te równania mają zastosowanie tylko dla pewnej maksymalnej wartości p. Porównując obie linie wykresu, zauważamy, że przy wystarczająco małych stopach utraty pakietów protokół HSTCP może wysyłać dane w bardziej agresywny sposób.

Aby protokół TCP mógł uzyskać taką funkcję odpowiedzi, procedura unikania przeciążenia została zmodyfikowana przez uwzględnienie aktualnego rozmiaru okna w momencie wykonywania zmian. Ma to miejsce, tak jak w konwencjonalnym protokole TCP, po przyjściu dobrego potwierdzenia ACK. Odpowiedź na przyjście dobrego potwierdzenia ACK została uogólniona w następujący sposób:

$$cwnd_{t+1} = cwnd_t + a(cwnd_t)/cwnd_t$$

Natomiast odpowiedź na zdarzenie przeciążenia (takie jak utrata pakietu, wskaźnik ECN) przedstawia się następująco:

$$cwnd_{t+1} = cwnd_t - b(cwnd_t) \cdot cwnd_t$$



Rysunek 16.19. *W przypadku protokołu HighSpeed TCP funkcja odpowiedzi TCP została zmieniona, tak aby była bardziej agresywna dla niskich stóp utraty pakietów i dużych okien, co pozwala uzyskać większe wartości przepływności w sieciach z dużym iloczynem pasmo-opóźnienie. Rysunek pochodzi z prezentacji Sally Floyd dla IETF TSVWG, marzec 2003*

W tym miejscu $a()$ oznacza funkcję określającą addytywne zwiększenie okna, a $b()$ oznacza funkcję określającą jego multiplikatywne zmniejszenie. W tym uogólnieniu standardowego TCP są to funkcje aktualnego rozmiaru okna. Aby uzyskać pożądaną funkcję odpowiedzi, zaczynamy od uogólnienia równania [3]:

$$W_0 = \frac{\sqrt{\frac{a(w)(2-b(w))}{2b(w)}}}{\sqrt{P_0}}$$

To daje:

$$a(w) = 2P_0 W_0^2 b(w)/(2-b(w))$$

Powyższa relacja nie ma jednego rozwiązania — znaczy to, że istnieje wiele kombinacji $a()$ i $b()$, które spełniają tę relacje, nawet jeśli niektóre z nich mogą nie mieć praktycznego czy odpowiedniego zastosowania.

Dodatkowe szczegóły dotyczące zmian zaproponowanych dla procedury unikania przeciążenia w protokole TCP, które sugeruje wariant HSTCP, są dostępne w dokumencie [RFC3649]. W towarzyszącym mu dokumencie [RFC3742] opisano, w jaki sposób można zmodyfikować procedurę powolnego startu, aby pomóc protokołowi TCP w uzyskaniu dobrze działającego okna przeciążenia w takich środowiskach. Zmieniona procedura nazywa się **ograniczonym powolnym startem** (*limited slow start*) i została zaprojektowana w celu spowolnienia powolnego startu w taki sposób, żeby protokół TCP pracujący z dużymi oknami (o rozmiarze rzędu tysięcy lub dziesiątek tysięcy pakietów) nie podwajał swojego okna w ciągu jednego czasu RTT.

W procedurze ograniczonego powolnego startu został wprowadzony nowy parametr o nazwie *max_ssthresh*. Wartość tego parametru nie jest maksymalną wartością *ssthresh*, ale jest progiem dla okna *cwnd*, który funkcjonuje następująco: jeśli *cwnd* <= *max_ssthresh*, powolny start przebiega w normalny sposób. Jeśli *max_ssthresh* < *cwnd* <= *ssthresh*, to okno *cwnd* jest zwiększane co najwyżej o (*max_ssthresh* /2) SMSS na jeden okres RTT. Jest to realizowane przez następującą modyfikację zarządzania oknem *cwnd* podczas powolnego startu:

```
if (cwnd <= max_ssthresh) {
      cwnd = cwnd + SMSS          (zwykły powolny start)
} else {
      K = int(cwnd / (0.5 * max_ssthresh))
      cwnd = cwnd + int((1/K)*SMSS)          (ograniczony powolny start)
}
```

Sugerowana możliwa wartość początkowa dla `max_ssthresh` wynosi 100 pakietów, czyli 100·SMSS bajtów.

16.8.2. Kontrola przeciążenia z binarnym zwiększaniem okna (BIC i CUBIC)

Wariant HSTCP jest jedną z kilku propozycji modyfikacji protokołu TCP w celu uzyskania większej przepływności w przypadku sieci o dużym iloczynie BDP. Chociaż rozwiązanie to rozpatruje kwestie przepływności i bezstronności w odniesieniu do konwencjonalnego protokołu TCP działającego w podobnych warunkach i decyduje się w pewnych okolicznościach na bardziej agresywne działanie w porównaniu do standardowego protokołu TCP, to jednak nie usiłuje bezpośrednio kontrolować tego, co się dzieje, kiedy połączenia HSTCP z różniącymi się czasami RTT rywalizują między sobą (jest to kwestia tzw. bezstronności RTT, *RTT fairness*). Badano to zagadnienie w kontekście standardowego protokołu TCP kilka lat temu; wykazano wtedy, że protokoły TCP z krótszymi czasami RTT uzyskują większy dostęp do szerokości pasma we współdzielonych łączach w porównaniu z tymi, które mają większe czasy RTT, w sytuacji gdy używany jest ten sam rozmiar pakietu i taka sama strategia potwierdzania odebranych danych (patrz [F91]). W protokołach TCP, które zwiększają okno *cwnd* w sposób zależny od jego rozmiaru (zwanych skalowalnymi względem pasma, *bandwidth-scalable*), ten brak bezstronności może być nawet poważniejszy. To, czy bezstronność RTT powinna być uważana za pożądaną, jest przedmiotem dyskusji. Chociaż bezstronność RTT mogłaby wydawać się atrakcyjna z zasadniczych względów, to jednak połączenia z większymi czasami RTT będą prawdopodobnie zużywać więcej zasobów sieciowych (np. przechodząc przez większą liczbę routerów), co może uzasadniać przydzielanie im nieco mniejszej przepustowości. W każdym razie sama wiedza o tym, jak funkcjonuje bezstronność RTT (lub jej brak), jest czynnikiem napędowym, kryjącym się za popularnymi wariantami protokołu TCP, które zbadamy w następnej kolejności.

16.8.2.1. Protokół BIC-TCP

W wyniku wysiłków podejmowanych w celu utworzenia skalowalnego protokołu TCP i uporania się z kwestią bezstronności RTT został opracowany protokół BIC-TCP (zwany dawniej protokołem BI-TCP; patrz [XHR04]), który został zaimplementowany w jądrze

systemu Linux, od wersji 2.6.8. Głównym celem protokołu BIC-TCP jest zapewnienie **liniowej bezstronności RTT** (*linear RTT fairness*) nawet wtedy, gdy okna przeciążenia są całkiem duże (co jest wymagane w przypadku używania łączy o dużej szerokości pasma). Liniowa bezstronność RTT polega na tym, że połączenia otrzymują przydziały pasma w odwrotnej proporcji do swoich czasów RTT, a nie na podstawie jakiejś bardziej skomplikowanej lub nieznanej funkcji.

To podejście modyfikuje działanie nadawcy w standardowym protokole TCP przez dodanie dwóch algorytmów, takich jak **zwiększanie okna metodą binarnego przeszukiwania** (*binary search increase*) oraz **addytywne zwiększanie okna** (*additive increase*). Algorytmy te są uruchamiane po wystąpieniu oznaki przeciążenia (np. utraty pakietu), ale w danym momencie czasu jest aktywny tylko jeden z nich. Algorytm zwiększania okna metodą binarnego przeszukiwania działa w następujący sposób: **bieżące okno minimalne** (*current minimum window*) jest ostatnim rozmiarem okna, przy którym połączenie nie doznało utraty pakietu w ciągu całego czasu RTT. **Okno maksymalne** (*maximum window*) to rozmiar okna, przy którym połączenie po raz ostatni doświadczyło utraty pakietu, o której wiadomo. Pożądany rozmiar okna znajduje się gdzieś pomiędzy tymi dwoma wartościami. Korzystając z techniki binarnego przeszukiwania, protokół BIC-TCP wybiera **okno próbne** (*trial window*) leżące w środku między tymi dwoma wartościami i cała procedura jest powtarzana w sposób rekurencyjny. Jeśli oknu próbnemu towarzyszy dalsza utrata pakietów, staje się ono nowym oknem maksymalnym i proces się powtarza. W przeciwnym przypadku staje się ono oknem minimalnym i proces również się powtarza. Procedura kończy się, kiedy różnica między oknem maksymalnym a oknem minimalnym staje się mniejsza od wstępnie zdefiniowanego progu, zwanego **minimalnym przyrostem** (*minimum increment*), a oznaczonego przez S_{min}.

Algorytm odnajduje na ogół pożądany rozmiar okna zwany także **punktem nasycenia** (*saturation point*) w liczbie prób określonej przez funkcję logarytmiczną, podczas gdy standardowy protokół TCP wymaga liczby prób określonych zależnością liniową (przeciętnie równej połowie różnicy rozmiarów okien). Dlatego też podejście to sprawia, że protokół BIC-TCP jest bardziej agresywny od standardowego protokołu TCP w pewnych okresach działania, ale jest to pożądane, gdyż pozwala na lepsze wykorzystanie szybkich środowisk bez niepotrzebnej zwłoki. Protokół jest nietypowy w porównaniu z innymi propozycjami, ponieważ jego funkcja określająca wzrost rozmiaru okna jest w pewnych punktach wklęsła — znaczy to, że szybkość jej wzrostu staje się mniejsza w miarę zbliżania się do punktu nasycenia. W większości pozostałych algorytmów używa się dużych przyrostów zmiany wielkości okna najbliżej punktu nasycenia.

Algorytm addytywnego zwiększania okna działa następująco: w trakcie używania procedury zwiększania okna metodą binarnego przeszukiwania może powstać sytuacja, w której różnica dzieląca aktualny rozmiar okna od punktu środkowego (w rozumieniu opisanej wcześniej procedury binarnego przeszukiwania) jest duża. Zwiększenie rozmiaru okna do wartości wyznaczonej przez punkt środkowy w czasie jednego okresu RTT może być nierozsądne ze względu na potencjalną możliwość spowodowania dużych chwilowych wzrostów ilości pakietów w sieci. Zapobiega temu algorytm addytywnego zwiększania okna, który jest wywoływany wtedy, kiedy różnica dzieląca punkt środkowy od aktualnego rozmiaru okna jest większa niż pewna określona wartość S_{max}. W opisanej sytuacji przyrost zostaje ograniczony do wartości S_{max} w pojedynczym czasie RTT, co nazywa się **zaciskaniem okna** (*window clamping*). Kiedy różnica między punktem

środkowym a oknem próbnym staje się mniejsza od S_{max}, do akcji powraca algorytm zwiększania okna metodą binarnego przeszukiwania. Ujmując rzecz całościowo, po wykryciu utraty danych okno jest zmniejszane o multiplikatywny czynnik β, po czym ponownie następuje jego wzrost przez zwiększanie addytywne, które przełącza się na metodę przeszukiwania binarnego z chwilą, kiedy pożądana wielkość wzrostu staje się mniejsza niż S_{max}. Autorzy nazywają te połączone algorytmy **binarnym zwiększaniem okna** (BI, *Binary Increase*).

Kiedy rozmiar okna wzrośnie ponad aktualne maksimum lub maksimum nie jest jeszcze znane, ponieważ nie wystąpiło żadne zdarzenie utraty danych, maksimum to musi zostać ustanowione. Jest to wykonywane przez procedurę znaną jako **sondowanie maksimum** (*max probing*). Celem sondowania maksimum jest wykorzystanie pasma z chwilą, gdy staje się dostępne. Przebiega ono w sposób symetryczny do algorytmów addytywnego i binarnego zwiększania okna. Rozpoczyna działanie od małych początkowych przyrostów, po których następują większe przyrosty, jeśli nie wystąpią oznaki przeciążenia. Podejście to wykazuje dobrą stabilność, ponieważ w pobliżu punktu nasycenia, kiedy sieć przypuszczalnie działa na granicy swojej największej pojemności, wykonywane są małe zmiany.

System Linux (wersje jądra od 2.6.8 do 2.6.17) zawiera implementację protokołu BIC-TCP, która jest domyślnie włączona. Jej działaniem sterują cztery parametry polecenia `sysctl`, mianowicie `net.ipv4.tcp_bic`, `net.ipv4.tcp_bic_beta`, `net.ipv4.tcp_bic_low_window` i `net.ipv4.tcp_bic_fast_convergence`. Ustawienie pierwszej zmiennej, typu logicznego, decyduje o tym, czy jest używana procedura BIC (a nie konwencjonalne procedury szybkiej retransmisji i szybkiego odtwarzania). Następna zmienna zawiera czynnik skalujący dla okna *cwnd*, służący do ustalenia wielkości S_{max} (domyślnie 819). Kolejny parametr określa minimalny rozmiar okna przeciążenia, przy którym sterowanie przejmują algorytmy protokołu BIC-TCP. Jego wartość domyślna wynosi 14, co oznacza, że dla małych wartości okna są używane procedury kontroli przeciążenia standardowego protokołu TCP. Ostatni parametr jest domyślnie włączoną flagą. Kiedy jest ustawiony, wpływa na sposób określania nowego okna maksymalnego i okna docelowego w sytuacji, gdy algorytm binarnego zwiększania okna wykazuje tendencję do jego zmniejszania. Podczas redukcji okna nowe okna, maksymalne i minimalne, otrzymują, odpowiednio, wartość bieżącego i przeskalowanego (w dół, o czynnik β) okna *cwnd*. Jeśli włączona jest szybka konwergencja (*fast convergence*), a wartość nowego maksimum jest mniejsza od jego poprzedniej wartości, przed przypisaniem mu wartości *cwnd*, wartość okna maksymalnego jest zmniejszana jeszcze bardziej, do średniej ze swojej własnej wartości i rozmiaru okna minimalnego. Po tej operacji, niezależnie od tego, czy szybka konwergencja jest włączona, czy nie, okno docelowe jest równe średniej z wartości maksymalnej i wartości minimalnej. To pomaga szybciej osiągnąć równomierny podział pasma w sytuacji, gdy wiele przepływów BIC-TCP współdzieli ten sam router.

16.8.2.2. Algorytm CUBIC

Autorzy protokołu BIC-TCP zrewidowali swoje podstawowe algorytmy, tworząc nowy algorytm kontroli przeciążenia o nazwie CUBIC (patrz [HRX08]). Jest on domyślnym algorytmem kontroli przeciążenia używanym w protokole TCP systemu Linux od wersji jądra 2.6.18. Wychodzi on naprzeciw zgłaszanym obawom, że protokół BIC-TCP może być w pewnych okolicznościach zbyt agresywny. Upraszcza także procedury zarządza-

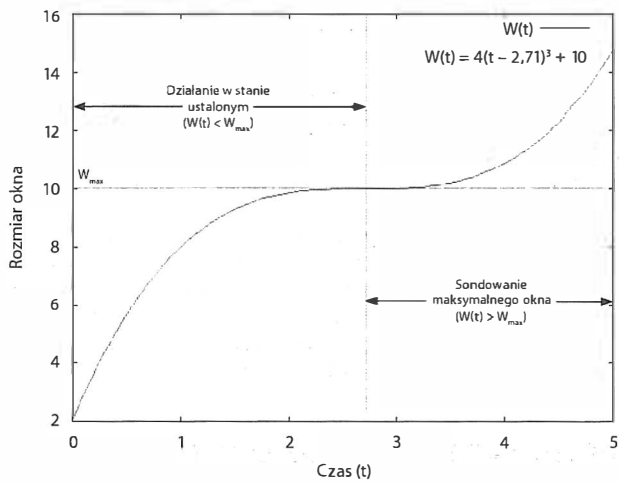
jące wzrostem okna. Zamiast wykorzystywania progu (S_{max}) do podejmowania decyzji, kiedy należy uruchomić zwiększanie okna z przeszukiwaniem binarnym w miejsce zwiększania addytywnego, do kontroli wzrostu okna jest używana funkcja wielomianowa nieparzystego stopnia, a szczególnie funkcja sześcienna. Funkcje sześcienne posiadają zarówno części wypukłe, jak i wklęsłe, co oznacza, że mogą rosnąć w pewnych fragmentach (wklęsłych) wolniej, a w innych (wypukłych) szybciej. Do momentu pojawienia się algorytmów BIC i CUBIC praktycznie cała literatura dotycząca protokołu TCP opowiadała się za użyciem wypukłych funkcji wzrostu okna. Konkretna funkcja wzrostu okna używana przez algorytm CUBIC do ustalania wartości *cwnd* wygląda następująco:

$$W(t) = C(t-K)^3 + W_{\max}$$

W powyższym równaniu $W(t)$ oznacza okno w czasie t, C jest stałym parametrem równania (domyślnie 0,4), t oznacza wyrażony w sekundach czas, który upłynął od ostatniej redukcji okna, a K oznacza okres czasu, jakiego potrzebuje funkcja, aby zwiększyć swoją wartość W do wartości W_{max}, kiedy nie dochodzi do kolejnego zdarzenia utraty danych. Wartość W_{max} jest równa ostatniemu rozmiarowi okna przed ostatnią korektą okna. Wartość K można obliczyć w następujący sposób:

$$K = \sqrt[3]{\frac{\beta W_{\max}}{C}}$$

gdzie β oznacza stałą określającą multiplikatywne zmniejszenie okna (domyślnie równą 0,2). Ilustracja funkcji wzrostu okna algorytmu CUBIC dla $K = 2{,}71$, $W_{max} = 10$ i $C = 0{,}4$ w przedziale czasu $t = [0, 5]$ została przedstawiona na rysunku 16.20.



Rysunek 16.20. *Funkcja wzrostu okna algorytmu CUBIC jest sześcienną funkcją czasu t. Ma ona część wklęsłą w obszarze, gdzie W(t) < W_{max}. W tym obszarze algorytm CUBIC poszukuje punktu nasycenia, zwiększając wartość cwnd z malejącą agresywnością. Po osiągnięciu wartości W_{max} funkcja wzrostu staje się wypukła, gdzie poszukuje dostępnej pojemności, zwiększając wartość cwnd ze wzrastającą agresywnością*

Na rysunku pokazujemy, że funkcja wzrostu okna algorytmu CUBIC zawiera zarówno część wklęsłą, jak i część wypukłą. Kiedy dochodzi do szybkiej retransmisji, parametr W_{max} otrzymuje wartość *cwnd*, a okno *cwnd* i próg *ssthresh* otrzymują nową wartość równą $\beta \cdot cwnd$. Algorytm CUBIC używa dla współczynnika β domyślnej wartości 0,8. Wartość $W(t + RTT)$ określa następną wartość docelowego okna przeciążenia. Kiedy w czasie obowiązywania procedury unikania przeciążenia przychodzi kolejne potwierdzenie ACK, okno *cwnd* jest zwiększane o $(W(t + RTT) - cwnd) / cwnd$.

Warto zauważyć, że określenie t jako ilości czasu, który upłynął od ostatniego zdarzenia redukcji okna, pomaga zapewnić bezstronność RTT. Zamiast zwiększania okna o jakąś stałą wielkość, kiedy przychodzą potwierdzenia ACK, wielkość zmiany okna jest funkcją czasu, jaki upłynął od ostatniej zmiany okna. To oddziela operacje zmiany okna od konkretnego wzorca przychodzenia potwierdzeń ACK.

Oprócz obszaru działania opartego na funkcji sześciennej, algorytm CUBIC posiada również obszar „zgodności z TCP" (który funkcjonuje, kiedy okno jest małe), aby zagwarantować, że protokół używający algorytmu CUBIC nie będzie pozbawiony równych szans we współzawodnictwie ze zwykłym protokołem TCP. Określimy to dokładniej: rozmiar okna standardowego protokołu TCP w funkcji czasu t, który upłynął, czyli $W_{tcp}(t)$, jest dany wzorem:

$$W_{tcp}(t) = 3\frac{t}{RTT}\frac{(1-\beta)}{(1+\beta)} + W_{\max}\beta$$

Jeśli więc okno *cwnd* jest mniejsze niż $W_{tcp}(t)$, kiedy przychodzi potwierdzenie ACK w czasie działania procedury unikania przeciążenia, algorytm CUBIC wykonuje podstawienie: $cwnd = W_{tcp}(t)$. Zapewnia to zgodność z TCP w często występujących sieciach o szybkościach od niskiej do umiarkowanej, gdzie w przeciwnym wypadku protokół stosujący algorytm CUBIC byłby w niekorzystnej sytuacji.

Jak już wcześniej wspomniano, CUBIC jest domyślnym algorytmem kontroli przeciążenia dla jądra systemu Linux, od wersji 2.6.18. Jednakże już od wersji jądra 2.6.18 system Linux obsługuje **dołączalne moduły unikania przeciążenia** (*pluggable congestion avoidance modules*; patrz [P07]), pozwalając użytkownikowi na wybranie algorytmu, którego chce używać. Zmienna `net.ipv4.tcp_congestion_control` zawiera aktualny domyślny algorytm kontroli przeciążenia (domyślna wartość: `cubic`). Zmienna `net.ipv4.tcp_available_congestion_control` zawiera algorytmy kontroli przeciążenia załadowane w systemie (ogólnie mówiąc, dodatkowe algorytmy mogą być załadowane jako moduły jądra). Zmienna `net.ipv4.tcp_allowed_congestion_control` zawiera te algorytmy, które mogą być używane przez aplikacje (albo specjalnie wybrane, albo domyślne). Ustawienie domyślne obejmuje algorytmy CUBIC i Reno.

16.9. Kontrola przeciążenia oparta na opóźnieniu

Sposoby kontroli przeciążenia, które dotąd poznaliśmy, są zwykle uruchamiane po utracie pakietów wykrywanej przy użyciu pewnej kombinacji potwierdzeń ACK lub opcji SACK, wskaźnika ECN (jeśli jest dostępny) lub wyzerowania licznika czasu retransmisji. Wskaźnik ECN (patrz podrozdział 16.11) umożliwia poinformowanie nadawczego protokołu

TCP o przeciążeniu, zanim sieć będzie zmuszona do odrzucania pakietów, ale wymaga to udziału routerów znajdujących się w sieci, co może okazać się niemożliwe. Jednak nawet bez wskaźnika ECN istnieje wciąż możliwość próby ustalenia ze strony hosta, czy aktualnie zagraża wystąpienie przeciążenia w sieci. Jedną ze wskazówek świadczących o tym, że może tworzyć się przeciążenie, jest wzrost zmierzonego czasu RTT po wprowadzeniu większej liczby pakietów do sieci. Widzieliśmy tę sytuację na rysunku 16.8, gdzie dodatkowe pakiety były raczej kolejkowane niż dostarczane, co przyczyniało się do wzrostu wartości mierzonego czasu RTT (dopóki pakiety nie zostały ostatecznie odrzucone). Kilka technik kontroli przeciążenia polega na tej obserwacji. Są one nazywane algorytmami kontroli przeciążenia **opartymi na opóźnieniu** (*delay-based*), dla odróżnienia od algorytmów kontroli przeciążenia **opartych na utracie danych** (*loss-based*), które dotąd omawialiśmy.

16.9.1. Vegas

W roku 1994 został wprowadzony protokół ***TCP Vegas*** (patrz [BP95]). Było to pierwsze, w ramach protokołu TCP, podejście do kontroli przeciążenia oparte na opóźnieniu, opublikowane i przetestowane przez społeczność projektantów protokołu TCP. Funkcjonowanie algorytmu Vegas polega na oszacowaniu ilości danych, jaką spodziewa się przesłać w pewnym odcinku czasu, i porównaniu jej z ilością danych, którą udaje mu się rzeczywiście przesłać. Jeżeli wymagana ilość danych nie zostaje przesłana, prawdopodobnie część ich zatrzymuje się w kolejce routera znajdującego się w ścieżce. Jeśli ten stan rzeczy utrzymuje się, nadawca używający algorytmu Vegas spowalnia transmisję. Kontrastuje to z podejściem standardowego protokołu TCP, który wymusza wystąpienie utraty pakietu w celu ustalenia punktu, w którym sieć staje się przeciążona.

Kiedy protokół Vegas znajduje się w fazie unikania przeciążenia, mierzy ilość przesłanych danych i otrzymaną liczbę dzieli przez minimalne opóźnienie zaobserwowane w połączeniu. Utrzymuje dwa progi α i β (gdzie $\alpha < \beta$). Kiedy różnica oczekiwanej przepływności (rozmiar okna podzielony przez najmniejszy zaobserwowany czas RTT) i rzeczywiście osiągniętej przepływności jest mniejsza niż α, okno przeciążenia jest zwiększane; jeśli ta różnica jest większa niż β, okno przeciążenia jest zmniejszane. W pozostałych przypadkach okno jest pozostawione bez zmian. Wszystkie zmiany okna przeciążenia mają charakter liniowy, co oznacza, że mamy do czynienia z mechanizmem kontroli przeciążenia opartym na zasadzie addytywnego zwiększania/addytywnego zmniejszania (AIAD, *Additive Increase/Additive Decrease*) okna.

Autorzy opisują progi α i β w kategoriach wykorzystania bufora w łączu stanowiącym wąskie gardło. Najmniejsze interesujące wartości wynoszą 1 dla α i 3 dla β. Uzasadnienie przyjęcia tych wartości jest następujące: przynajmniej jeden bufor pakietu w ścieżce sieciowej powinien być zajęty (tzn. znajdować się w kolejce routera związanego z łączem o najmniejszej szerokości pasma w całej ścieżce), żeby utrzymywać wykorzystanie sieci. Jeśli staje się dostępne dodatkowe pasmo, zajęcie dwóch dodatkowych buforów (w sumie 3, co jest wartością progu β) likwiduje potrzebę odczekiwania dodatkowego czasu RTT, żeby wprowadzić do sieci więcej danych, co by było wymagane, gdyby protokół Vegas próbował utrzymywać tylko jeden pełny bufor. Co więcej, traktowanie obszaru ($\beta - \alpha$) jako zakresu działania pozostawia pewne miejsce na pomniejsze zmiany w przepływności bez powodowania natychmiastowej zmiany okna — jest to rodzaj tłumienia, który ma na celu zmniejszenie oscylacji szybkości transmisji.

Przy niewielkiej modyfikacji podejście to może zostać zastosowane także do okresu powolnego startu. W tym przypadku zwiększanie okna *cwnd* o 1 dla każdego dobrego potwierdzenia ACK jest dozwolone tylko w **co drugim** czasie RTT. Dla tych czasów RTT, w których okno nie jest zwiększane, wykonywany jest pomiar w celu upewnienia się, że przepływność wzrasta. Jeśli jest inaczej, nadawca przełącza się na wykonywanie procedury unikania przeciążenia właściwej dla wariantu Vegas.

W pewnych okolicznościach protokół Vegas może zostać „oszukany" błędnie sądzić, że opóźnienie w kierunku docelowym jest większe niż jego rzeczywista wartość. Tak się dzieje, kiedy ma miejsce znaczące przeciążenie w odwrotnym kierunku (pamiętajmy, że ścieżki wykorzystywane przez obydwa kierunki połączenia TCP mogą być różne i znajdować się w odmiennym stanie pod względem przeciążenia). W takich przypadkach pakiety (potwierdzenia ACK) wracające do nadawczego protokołu TCP są opóźnione, mimo że nadawca w rzeczywistości nie przyczynia się do przeciążenia (w odwrotnej ścieżce). To powoduje zmniejszenie okna przeciążenia przez Vegas nawet w sytuacji, gdy taka korekta nie jest naprawdę potrzebna. Jest to potencjalna pułapka dla większości technik opartych na pomiarach czasu RTT jako podstawie decyzji dotyczących kontroli przeciążenia. Faktycznie, znaczny ruch w odwrotnym kierunku może wyraźnie zakłócać (patrz [M92]) funkcjonowanie zegara ACK (patrz rysunek 16.1).

Protokół Vegas jest bezstronny w stosunku do innych protokołów TCP typu Vegas dzielących tę samą ścieżkę, ponieważ każdy z nich zmusza sieć do przechowywania tylko minimalnej ilości danych. Jednak Vegas i przepływy obsługiwane przez standardowy protokół TCP nie współdzielą ścieżek na równych zasadach. Standardowy protokół TCP wykazuje tendencję do zapełniania kolejek w sieci, podczas gdy Vegas zwykle zachowuje je niemal puste. W konsekwencji, kiedy standardowy nadawca wprowadza więcej pakietów, protokół Vegas zauważa zwiększone opóźnienie i spowalnia swoje transmisje. Ostatecznie prowadzi to do nieuczciwej stronniczości na korzyść standardowego protokołu TCP. Algorytm Vegas jest obsługiwany przez system Linux, ale nie jest domyślnie włączony. W wersjach jądra wcześniejszych od wersji 2.6.13 o tym, czy używany jest wariant Vegas, rozstrzyga ustawienie zmiennej `net.ipv4.tcp_vegas_cong_avoid` (jej domyślna wartość wynosi 0) dostępnej w poleceniu `sysctl`. Zmienne `net.ipv4.tcp_vegas_alpha` (o wartości domyślnej 2) i `net.ipv4.tcp_vegas_beta` (o wartości domyślnej 6) odpowiadają wcześniej opisanym progom α i β, ale ich wartości są wyrażone w połówkach pakietów (np. wartość 6 odpowiada trzem pakietom). Zmienna `net.ipv4.tcp_vegas_gamma` (o wartości domyślnej 2) określa, ile połówek pakietów protokół Vegas powinien starać się utrzymywać w sieci podczas powolnego startu. Dla wersji jądra następujących po wersji 2.6.13 Vegas musi być ładowany jako osobny moduł jądra i jest włączany przez nadanie zmiennej `net.ipv4.tcp_congestion_control` wartości `vegas`.

16.9.2. FAST

Protokół TCP FAST został opracowany z poświęceniem szczególnej uwagi działaniu w szybkich środowiskach o dużych iloczynach pasmo-opóźnienie (patrz [WJLH06]). Podobny do Vegas pod względem pomysłu, wariant ten koryguje okno na podstawie różnicy między oczekiwaną a doświadczaną przepływnością. Różni się od algorytmu Vegas, bo koryguje okno nie tylko w oparciu o rozmiar okna, ale uwzględnia także na różnicę między aktualną i oczekiwaną wydajnością. Aktualizuje szybkość wysyłania danych co

drugi czas RTT, używając techniki regulowania szybkości transmisji (*rate-pacing*). Jeśli zmierzone opóźnienie ma wartość znacznie poniżej progu, okno jest aktualizowane w agresywny sposób, po czym następuje okres, w którym zwiększanie okna jest mniej agresywne. Kiedy opóźnienie zwiększa się, ma miejsce proces odwrotny. Algorytm FAST różni się od pozostałych omawianych podejść, ponieważ jest przedmiotem kilku patentów i jest w sposób niezależny komercjalizowany. Jest też nieco słabiej zbadany przez społeczność naukową, ale niezależna ocena (patrz [S09]) pokazała, że charakteryzuje go dobra stabilność i bezstronność.

16.9.3. Protokoły TCP Westwood i TCP Westwood+

Protokoły ***TCP Westwood*** (**TCPW**) i ***TCP Westwood+*** (**TCPW+**) mają na celu obsługę ścieżek z dużym iloczynem pasmo-opóźnienie przez modyfikację zachowania nadawcy w konwencjonalnym protokole TCP NewReno. TCPW+ jest korektą oryginalnego algorytmu TCPW, dlatego też będziemy się odwoływać do każdego z nich przy użyciu akronimu TCPW. W protokole TCPW oszacowanie odpowiedniej szybkości (ERE, *Eligible Rate Estimate*) jest oszacowaniem pasma dostępnego w połączeniu. Jest ono ciągle obliczane w sposób nieco podobny do algorytmu Vegas (na podstawie różnicy między oczekiwaną a osiąganą szybkością transmisji), ale ze zmiennym interwałem pomiarów szybkości transmisji, opartym na dynamice przychodzenia potwierdzeń ACK. Kiedy poziom przeciążenia jest niski, interwał pomiaru jest mały i na odwrót. Kiedy wykrywana jest utrata pakietu, TCPW, zamiast zmniejszenia okna *cwnd* o połowę, oblicza szacunkową wartość BDP (wartość ERE pomnożona przez minimalny zaobserwowany czas RTT) i używa jej jako nowej wartości progu *ssthresh*. **Sprawne sondowanie** (*agile probing*; patrz [WYSG05]) w sposób adaptacyjny i ciągły ustawia wartość *ssthresh*, podczas gdy w innym przypadku połączenie działałoby zgodnie z procedurą powolnego startu. To sprawia, że okno *cwnd* rośnie wykładniczo w sytuacjach, gdy próg *ssthresh* został zwiększony (przez zainicjowanie powolnego startu). Algorytm Westwood może być włączony w systemie Linux w wersjach jądra następujących po wersji 2.6.13 przez załadowanie modułu TCPW i nadanie zmiennej `net.ipv4.tcp_congestion_control` wartości `westwood`.

16.9.4. Protokół Compound TCP

Począwszy od systemu Windows Vista, istnieje możliwość wyboru procedury kontroli przeciążenia (tzw. „dostawcy" — *provider*), której powinien użyć protokół TCP, w sposób podobny do dołączalnych modułów unikania przeciążenia systemu Linux. Jedna z takich opcji (ale nie domyślna, z wyjątkiem systemu Windows Server 2008) nosi nazwę ***Compound TCP*** (CTCP; patrz [TSZS06]) — złożony protokół TCP. Protokół CTCP wykonuje korekty okna na podstawie utraty pakietów, ale również na podstawie pomiarów opóźnień. W pewnym sensie stanowi on połączenie standardowego protokołu TCP i Vegas, ale z właściwościami skalowalności protokołu HSTCP.

Autorzy rozpoczynają od opisania szeregu wyników badań nad algorytmami Vegas i FAST, które sugerują, że mechanizmy kontroli przeciążenia oparte na opóźnieniu wykazują lepsze wykorzystanie przepustowości, mniej samoczynnie wywołanych zdarzeń utraty pakietów, szybszą konwergencję (do prawidłowego punktu działania), a ponadto lepszą bezstronność RTT i stabilizację. Jednak, jak już wspominaliśmy, podejścia oparte na opóźnieniu przejawiają tendencje do tracenia pasma w sytuacji, gdy rywalizują z podejściami

opartymi na utracie danych. Protokół CTCP próbuje zaradzić tej sytuacji przez połączenie podejścia opartego na opóźnieniu z podejściem opartym na utracie danych. Aby to zrobić, protokół CTCP wprowadza nową zmienną sterującą oknem o nazwie *dwnd* (*delay window* — okno opóźnienia). Wówczas okno użyteczne przedstawia się następująco:

$$W = \min(cwnd + dwnd, awnd)$$

Obsługa okna *cwnd* jest podobna do tej, która ma miejsce w standardowym protokole TCP, ale dodanie wartości *dwnd* umożliwia wysłanie dodatkowych pakietów, jeśli warunki związane z opóźnieniem są odpowiednie. Kiedy przychodzą potwierdzenia ACK w czasie obowiązywania procedury unikania przeciążenia, okno *cwnd* jest aktualizowane w następujący sposób:

$$cwnd = cwnd + 1/(cwnd + dwnd)$$

Zarządzanie oknem *dwnd* jest oparte na algorytmie Vegas i ma ono wartość niezerową tylko w czasie unikania przeciążenia (CTCP używa konwencjonalnej procedury powolnego startu). W trakcie funkcjonowania połączenia w zmiennej *baseRTT* utrzymywana jest minimalna wartość pomiaru czasu RTT. Następnie obliczana jest różnica *diff* między oczekiwaną a rzeczywistą ilością danych pozostających w sieci według następującego wzoru: $diff = W \cdot (1 - (baseRTT/RTT))$, gdzie *RTT* jest (wygładzonym) oszacowaniem czasu RTT. Wartość *diff* oszacowuje liczbę pakietów (lub bajtów) zakolejkowanych w sieci. Protokół CTCP, podobnie jak większość procedur opartych na opóźnieniu, stara się utrzymywać wartość *diff* na poziomie pewnego progu, o nazwie γ, w celu zagwarantowania, że sieć jest wykorzystywana, ale nie jest przeciążona. Po uwzględnieniu tego celu proces określania wartości *dwnd* wyrażony jest następującym wzorem:

$$dwnd(t+1) = \begin{cases} dwnd(t) + (\alpha \cdot win(t)^k - 1)^+, \text{ jeśli } diff < \gamma \\ (dwnd(t) - \zeta \cdot diff)^+, \text{ jeśli } diff \geq \gamma \\ (win(t) \cdot (1-\beta) - cwnd/2)^+, \text{ jeśli wykryto utratę danych} \end{cases}$$

gdzie $(x)^+$ oznacza $\max(x, 0)$. Zauważmy, że okno *dwnd* nigdy nie może być ujemne. Natomiast może być równe 0, w którym to przypadku CTCP zachowuje się jak standardowy protokół TCP.

W pierwszym przypadku, w którym sieć może być niewykorzystana w pełni, protokół CTCP zwiększa okno *dwnd* zgodnie z wielomianem $\alpha \cdot win(t)^k$. Jest to forma dwumiennego wzrostu i stanowi sposób uczynienia protokołu CTCP bardziej agresywnym (podobnym do HSTCP) wtedy, kiedy szacunkowa zajętość buforów jest mniejsza niż γ. W drugim przypadku, w którym zajętość buforów wydaje się wzrastać powyżej pożądanego progu γ, stała ζ dyktuje, jak szybko powinien być zmniejszany składnik oparty na opóźnieniu (ale pamiętajmy, że wartość *dwnd* jest zawsze dodawana do *cwnd)*. I to przyczynia się do bezstronności protokołu CTCP względem RTT i TCP. W sytuacji, gdy wykryta zostaje utrata danych, do okna *dwnd* stosowany jest jego własny współczynnik multiplikatywnego zmniejszania β.

Jak można zobaczyć, protokół CTCP może być poddany regulacji przy użyciu parametrów k, α, β, γ i ζ. Wartość parametru k wpływa na poziom agresywności. Wartość wynosząca ok. 0,8 była pożądaną wartością zapewniającą podobieństwo do protokołu

HSTCP, ale ze względów implementacyjnych wybrano wartość 0,75. Wartości α i β wpływają na płynność i czas reakcji. Wartości domyślne wynoszą odpowiednio 0,125 i 0,5. Dla parametru γ autorzy sugerują przyjęcie wartości domyślnej 30 pakietów na podstawie oceny empirycznej. Jeśli wartość ta jest za mała, może nie być wystarczającej ilości pakietów pozostających w sieci, aby uzyskać dobre pomiary opóźnienia. Na odwrót, zbyt duże wartości mogłyby skutkować niepożądanym trwałym przeciążeniem.

CTCP jest stosunkowo nowym protokołem, więc bez wątpienia zostaną wykonane dalsze eksperymenty i oceny, które pozwolą przekonać się, jak dobrze i bezstronnie współzawodniczy on ze standardowym protokołem TCP i jak dobrze potrafi się przystosować do znaczących zmian dostępnej szerokości pasma. W studium opartym na symulacji autor pracy [W08] zauważył, że CTCP może osiągać słabą wydajność, kiedy bufory w sieci są małe (np. mniejsze niż γ). Sugeruje również, że CTCP może paść ofiarą pewnych problemów charakterystycznych dla wariantu Vegas, łącznie ze zmianą tras (wymagającą przystosowania się do nowych ścieżek z innymi opóźnieniami) i trwałym przeciążeniem. Na koniec zauważa, że w sytuacji, gdy wiele przepływów CTCP, z których każdy stara się utrzymać γ pakietów w przelocie, współdzieli to samo łącze będące wąskim gardłem, wydajność może być kiepska.

Jak już wcześniej wspominaliśmy, algorytm CTCP nie jest domyślnie włączony w większości wersji systemu Windows. Jednak, żeby wybrać CTCP jako „dostawcę" obsługi przeciążenia, można użyć następującego polecenia:

```
C:\> netsh interface tcp set global congestionprovider=ctcp
```

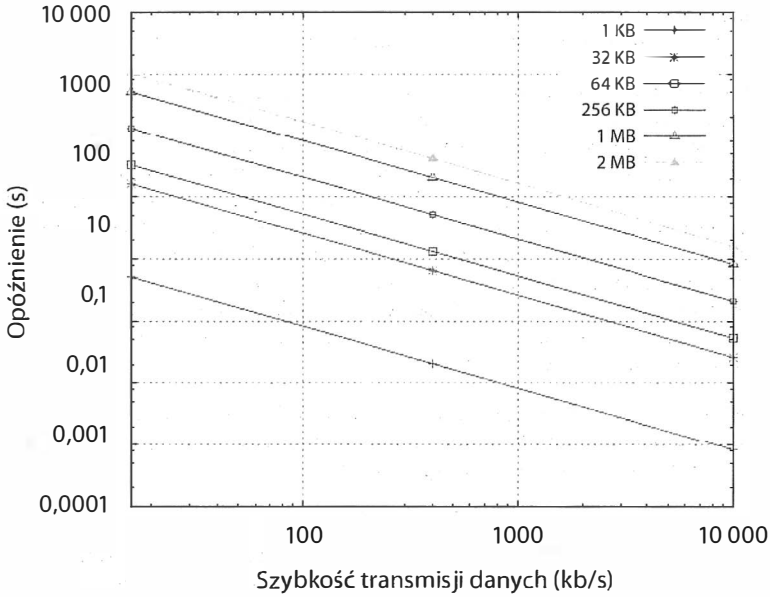
Można protokół wyłączyć, wybierając innego dostawcę (lub używając terminu `none`). Wariant CTCP został również przeniesiony do systemu Linux jako dołączalny moduł unikania przeciążenia, ale nie jest zawarty w standardowym zestawie.

16.10. Rozdęcie buforów

Chociaż pamięć tradycyjnie była droga (i pozostaje taka w przypadku wysokiej klasy routerów), teraz łatwo znajdziemy komercyjny sprzęt sieciowy, który zawiera znaczną ilość pamięci, potencjalnie wiele megabajtów buforów na pakiety. Być może jest to ironia losu, ale duża ilość pamięci (w porównaniu do tradycyjnych urządzeń sieciowych) może faktycznie prowadzić do pogorszenia wydajności w przypadku takich protokołów jak TCP. Problem ten został określony terminem „rozdęcie buforów" (*buffer bloat*; patrz [G11] [DHGS07]). Odnosi się do dużych wartości opóźnienia sieci wprowadzonego przez opóźnienie kolejkowania, głównie po stronie łącza nadrzędnego bram i punktów dostępu znajdujących się w domach i małych biurach. Algorytmy kontroli przeciążenia standardowego protokołu TCP, które zwykle utrzymują zapełnione bufory w wąskich gardłach sieci, nie działają dobrze, kiedy występuje duży zakres buforowania pomiędzy nadawcą a odbiorcą, ponieważ dostarczenie do nadawcy informacji o zdarzeniu wskazującym na przeciążenie (czyli utracie pakietu) zabiera dużo czasu.

W pracy [KWNP10] autorzy odkrywają, że szerokość pasma w kierunku wysyłania w modemach kablowych i DSL w Stanach Zjednoczonych mieści się w zakresie od 256 kb/s do 4 Mb/s. Doszli również do wniosku, że rozmiar bufora w komercyjnych routerach

mieści się zakresie od 16 kB do 256 kB. Na rysunku 16.21 pokazujemy, w jaki sposób opóźnienie jest związane z szybkością transmisji dla kilku rozmiarów bufora, by dostarczyć właściwej perspektywy dla tych wyników badań.



Rysunek 16.21. *Podwójnie logarytmiczny wykres pokazuje opóźnienie sieci wynikające z opóźnienia kolejkowania, jakiego doświadczają dane w pełni obciążonych kolejkach różnych rozmiarów. W sytuacji, gdy duże bufory pozostają zapełnione (jest to tzw. „rozdęcie buforów"), aplikacje interaktywne mogą być narażone na niedopuszczalne opóźnienia rzędu kilku sekund*

Na tym rysunku wykres w skali podwójnie logarytmicznej pokazuje wielkość opóźnienia spotykającego dane, które muszą być kolejkowane, dla różnych rozmiarów bufora (1 kB – 2 MB). Szybkości wysyłania danych do sieci używane w domowym (lub biurowym) Internecie (zwykle od 250 kb/s do 10 Mb/s) mogą prowadzić do opóźnień rzędu kilku sekund, jeśli rozmiary buforów wynoszą kilkaset kilobajtów lub więcej). Aplikacje interaktywne wymagają na ogół, aby opóźnienia w jednym kierunku nie przekraczały 150 ms, by można było dostarczyć użytkownikom usługi dobrej jakości (patrz [G114]). Tak więc, jeśli bufory pozostają zapełnione do swojej maksymalnej pojemności z powodu jednej lub większej liczby współzawodniczących operacji wysyłania danych do sieci (np. w przypadku dystrybucji plików z wykorzystaniem protokołu BitTorrent), interaktywne aplikacje mogą odczuć negatywne skutki tej sytuacji.

Rozdęcie buforów nie jest problemem dotyczącym całego sprzętu sieciowego. Istotnie, główny kłopot zdaje się dotyczyć urządzeń dostępowych użytkowników końcowych, wyposażonych w zbyt duże bufory. Istnieje wiele potencjalnych sposobów zajęcia się tą kwestią, łącznie z modyfikacją protokołów (np. kontrolą przeciążenia opartą na opóźnieniu, taką jak w przypadku protokołu Vegas, na którą jednak może mieć negatywny wpływ wysoki stopień rozsynchronizowania; patrz [DHGS07]), dynamicznym określaniem

rozmiaru buforów w urządzeniach dostępowych (sugerowanym w pracy [KWNP10]) lub kombinacją tych dwóch podejść. W następnej kolejności zajmiemy się podejściem kombinowanym, które nie tylko może pomóc w rozwiązaniu problemu rozdęcia buforów, ale daje także szereg innych korzyści.

16.11. Aktywne zarządzanie kolejkami i znacznik ECN

Analiza reakcji protokołu TCP na przeciążenie dotychczas zakładała, że jedynym sposobem, w jaki protokół TCP dochodzi do wniosku, że ma miejsce przeciążenie, jest obserwacja zdarzeń utraty pakietu. Szczególnie routery (które są najbardziej narażone na przeciążenie) zazwyczaj nie pomagają w informowaniu protokołu TCP każdego z hostów, że przeciążenie „wisi w powietrzu". Zamiast tego, odrzucają po prostu pakiety, kiedy nie ma już więcej miejsca w buforze (tzw. „obcięcie ogona" — *drop tail*), i wysyłają dalej pakiety, które już wcześniej dotarły w sposób właściwy dla kolejki FIFO (*first-in-first-out* — wysyłanie w kolejności przychodzenia). Kiedy routery w Internecie są **pasywne** (*passive*; tzn., po prostu odrzucają pakiety, kiedy są przeciążone, i nie dostarczają informacji zwrotnej dotyczącej ich stanu przeciążenia), protokół TCP niewiele więcej może uczynić, niż zareagować już po fakcie. Jeśli jednak te routery posiadałyby sposób na bardziej aktywne zarządzanie swoimi kolejkami (np. przez używanie bardziej wyrafinowanej strategii szeregowania i zarządzania buforami niż FIFO/*drop tail*), być może sytuacja mogłaby ulec poprawie. Jeśli by mogły również sygnalizować swój stan przeciążenia punktom końcowym protokołu TCP, byłoby jeszcze lepiej.

O routerach, które stosują strategie szeregowania i zarządzania buforami inne niż FIFO/*drop tail*, mówi się zwykle, że są **aktywne** (*active*), a odpowiednie metody, których używają do zarządzania swoimi kolejkami nazywają się mechanizmami **aktywnego zarządzania kolejkami** (AQM, *Active Queue Management*). Autorzy dokumentu [RFC2309] przedstawiają analizę potencjalnych korzyści z zastosowania AQM. Chociaż system AQM może być pożyteczny sam w sobie, staje się jeszcze bardziej pożyteczny, kiedy routery i przełączniki implementujące AQM posiadają wspólną metodę przekazywania informacji o swoim stanie do systemów końcowych. W przypadku protokołu TCP zostało to opisane w dokumencie [RFC3168] i rozszerzone o dodatkowe zabezpieczenia w eksperymentalnej specyfikacji [RFC3540]. Te dokumenty RFC opisują **jawne powiadomienie o przeciążeniu** (ECN, *Explicit Congestion Notification*), które określa sposób, w jaki routery **oznaczają** pakiety (przez zagwarantowanie ustawienia obu bitów ECN w nagłówku IP), aby pokazać pojawienie się przeciążenia.

Bramy (*gateways*) z **losowym wczesnym wykrywaniem** (RED, *Random Early Detection*, patrz [FJ93]) są jednym z mechanizmów polecanych jako zdolne do wykrywania początku przeciążenia i kontroli oznaczania pakietów. Bramy te implementują dyscyplinę zarządzania kolejkami, która mierzy przeciętną zajętość kolejki w czasie. Jeśli zajętość przekracza minimum (nazwane *minthresh* — próg minimalny) i jest mniejsza niż maksimum (nazwane *maxthresh* — próg maksymalny), pakiet jest oznaczany z rosnącym prawdopodobieństwem. Jeśli przeciętna zajętość kolejki przekracza próg *maxthresh*, pakiety są oznaczane z konfigurowalnym maksymalnym prawdopodobieństwem (nazwanym *MaxP*), które może być równe 1,0. Mechanizm RED może być także skonfigurowany do odrzucania pakietów zamiast ich oznaczania.

Algorytm RED stanowi podstawę szeregu wariantów (np. WRED firmy Cisco, który używa różnych instancji algorytmu RED na podstawie pola DSCP lub wartości pierwszeństwa w nagłówku IP) obsługiwanych w wielu routerach i przełącznikach.

Po odebraniu pakietu przez protokół TCP znacznik przeciążenia wskazuje, że pakiet przeszedł przez przeciążony router. Oczywiście, to **nadawca** (a nie odbiorca) naprawdę potrzebuje tej informacji, aby zareagować i spowolnić transmisję. Tak więc odbiorca odsyła echo tego wskaźnika do nadawcy w serii pakietów ACK.

Mechanizm ECN działa częściowo w warstwie IP, więc potencjalnie może zostać zastosowany w protokołach transportowych innych niż TCP, chociaż większość działań związanych z obsługą ECN znajduje się w gestii protokołu TCP i właśnie je tutaj omawiamy. Kiedy zdolny do obsługi ECN router, doświadczający utrzymującego się przeciążenia, odbiera pakiet IP, zagląda do nagłówka IP, szukając wskaźnika **ECT** (*ECN-Capable Transport* — protokół transportowy jest zdolny do obsługi ECN), aktualnie zdefiniowanego jako fakt ustawienia dowolnego z dwóch bitów ECN w nagłówku IP). Jeśli bit jest ustawiony, znaczy to, że protokół transportowy odpowiedzialny za wysłanie pakietu rozumie mechanizm ECN. W tym momencie router ustawia wskaźnik CE (*Congestion Experienced*; wystąpiło przeciążenie) w nagłówku IP przez nadanie obu bitom wartości 1 i przesyła datagram dalej. Routery nie powinny ustawiać wskaźnika CE w sytuacji, gdy przeciążenie nie wydaje się być trwałe (np. po pojedynczym, w ostatnim okresie, odrzuceniu pakietu z powodu przepełnienia kolejki), ponieważ zakłada się, że protokół transportowy zareaguje nawet na *pojedyncze* wystąpienie wskaźnika CE.

Protokół TCP odbiorcy, widząc pakiet przychodzących danych z ustawionym wskaźnikiem CE, jest zobowiązany do odesłania tego wskaźnika do nadawcy (istnieje eksperymentalne rozszerzenie przewidujące dodanie ECN również do segmentów SYN+ACK; patrz [RFC5562]). Ponieważ odbiorca zwykle zwraca informacje do nadawcy przy użyciu (zawodnych) pakietów ACK, istnieje znacząca możliwość, że wskaźnik przeciążenia zostanie zgubiony. Z tego powodu protokół TCP implementuje mały, poprawiający niezawodność, protokół odsyłania wskaźnika do nadawcy. Po odebraniu przychodzącego pakietu z ustawionym wskaźnikiem CE protokół TCP odbiorcy ustawia bit ECN-Echo w każdym wysyłanym pakiecie ACK, dopóki nie otrzyma od protokołu TCP nadawcy kolejnego pakietu danych z bitem CWR ustawionym na 1. Ustawienie bitu CWR wskazuje, że okno przeciążenia (a zatem szybkość transmisji) zostało zmniejszone.

Chociaż mechanizmy RED i ECN są znane od prawie dwóch dekad, nie znalazły szerokiego zastosowania w Internecie. Podawano różne powody, dlaczego tak się dzieje (np. trudność w ustawianiu parametrów RED, dostrzeganie ograniczonych korzyści). Rewizja mechanizmu ECN (patrz [K05]) wykonana w 2005 roku wykazała, że używanie ECN tylko w pakietach danych znacznie ogranicza oferowane przez ten mechanizm korzyści. Eksperymentalne rozszerzenie zawarte w dokumencie [RFC5562] definiuje użycie ECN w pakietach SYN+ACK z możliwością znacznego zwiększenia użyteczności ECN dla pewnych rodzajów zadań (np. dla ruchu związanego z obsługą stron WWW).

Nadawczy protokół TCP, odbierając wskaźnik ECN-Echo zawarty w pakiecie ACK, reaguje w taki sam sposób, jakby zareagował po wykryciu pojedynczej utraty pakietu — koryguje okno *cwnd* i powoduje ponadto ustawienie bitu CWR w kolejnym pakiecie danych. Wywoływana jest normalna w tej sytuacji reakcja na przeciążenie algorytmów szybkiej retransmisji/szybkiego odtwarzania (oczywiście, bez retransmisji pakietu), powodująca

spowolnienie działania protokołu TCP, zanim dojdzie do utraty pakietów. Zauważmy, że protokół TCP nie powinien posuwać się za daleko; szczególnie nie powinien reagować więcej niż jeden raz dla tego samego okna danych. Ten sposób postępowania stawiałby protokół TCP obsługujący ECN w zbyt niekorzystnym położeniu w stosunku do innych protokołów.

W Windows Vista i późniejszych systemach mechanizm ECN musi zostać włączony, by mógł być używany:

```
C:\> netsh int tcp set global ecncapability=enabled
```

W systemie Linux mechanizm ECN jest włączony, jeżeli zmienna logiczna polecenia `sysctl`, o nazwie `net.ipv4.tcp_ecn`, ma niezerową wartość. Ustawienie domyślne różni się w zależności od używanej dystrybucji Linuksa, najczęściej wskazuje stan wyłączenia. W systemie Mac OS 10.5 i późniejszych zmienne `net.inet.tcp.ecn_initiate_out` i `net.inet.tcp.ecn_negotiate_in` określają, czy mechanizm ECN jest włączony, odpowiednio, dla ruchu wychodzącego i ruchu przychodzącego z ustawionymi flagami `ECN`. Oczywiście, bez współdziałania ze strony routerów lub przełączników użyteczność mechanizmu ECN jest w każdym wypadku ograniczona. Czas pokaże, czy wizja wykorzystania mechanizmów AQM zostanie kiedykolwiek w pełni zrealizowana w globalnym Internecie.

Mechanizmy RED i ECN są z powodzeniem używane w środowiskach operacyjnych radykalnie innych od tych, dla których były zaprojektowane. Microsoft i Uniwersytet Stanforda opracowały *Data Center TCP* (DCTCP; TCP dla centrów danych; patrz [A10]), w którym użyto algorytmu RED zaimplementowanego w przełącznikach warstwy 2. z uproszczonymi parametrami do oznaczania pakietów w sytuacji, gdy dochodzi do chwilowego przeciążenia. Autorzy tego rozwiązania modyfikują również zachowanie protokołu TCP odbiorcy, który ustawia bit `ECN-Echo` w pakietach `ACK` tylko wtedy, kiedy ostatnio odebrany pakiet zawiera znacznik CE. Informują oni o 90% redukcji zajętości buforów przy porównywalnej przepływności TCP, co umożliwia dziesięciokrotne zwiększenie obsługiwanego ruchu sieciowego w tle.

16.12. Ataki związane z kontrolą przeciążenia protokołu TCP

Widzieliśmy już, jak protokół TCP może być zaatakowany przez generowanie pakietów, które powodują, że automat stanów połączenia protokołu TCP kończy połączenie. Protokół TCP może również zostać zaatakowany (lub przynajmniej sprowokowany do zachowywania się w dziwny sposób), kiedy pracuje w stanie `ESTABLISHED`. Większość ataków na system kontroli przeciążenia protokołu TCP stara się zmusić TCP do szybszego lub wolniejszego wysyłania danych, niżby to miało miejsce w zwykłych okolicznościach.

Być może, najwcześniejszy typ ataku jest związany z fabrykowaniem komunikatów ICMPv4 *Source Quench* (tłumienie nadawcy). Kiedy zostaną one dostarczone do hosta używającego protokołu TCP, każde połączenie z adresem IP zawartym w „winnym" zaistniałej sytuacji datagramie, znajdującym się wewnątrz komunikatu ICMP, spowalnia swoje działanie. Mogło to stanowić lukę bezpieczeństwa ileś tam lat temu, ale używanie

komunikatów *Source Quench* przez routery do kontroli przeciążenia zostało zarzucone, od mniej więcej 1995 roku (na podstawie punktu 5.3.6 dokumentu [RFC1812]). Z drugiej strony, w przypadku hostów końcowych dokument [RFC1122] postanowił, że protokół TCP musi zareagować na komunikat *Source Quench* przez spowolnienie transmisji. Po łącznym uwzględnieniu tych dwóch faktów najprostszym rozwiązaniem jest zablokowanie ruchu ICMP *Source Quench* w routerze lub w hoście, co jest obecnie powszechną praktyką.

Bardziej wyrafinowany i współczesny zestaw ataków został rozważony po przyjrzeniu się **niewłaściwie zachowującym się odbiorcom** (patrz [SCWA99]). Autorzy opisują trzy typy ataków, które mogą powodować, że protokół TCP nadawcy wprowadza dane do sieci z szybkością większą od zamierzonej. Takie ataki mogłyby np. zostać wykorzystane do uzyskania przez jakiegoś klienta WWW nieuczciwej przewagi nad innymi współzawodniczącymi klientami. Ataki te noszą nazwy: **podział potwierdzeń ACK** (*ACK division*), **fałszowanie zduplikowanych potwierdzeń ACK** (*DupACK spoofing*) i **optymistyczne potwierdzanie** (*Optimistic ACKing*), a zostały zaimplementowane w wariancie protokołu TCP, który autorzy (żartobliwie) nazwali „TCP Daytona".

Podział potwierdzeń ACK funkcjonuje jako generowanie więcej niż jednego potwierdzenia ACK dla potwierdzanego zakresu bajtów. Ponieważ kontrola przeciążenia zazwyczaj działa w oparciu o przychodzenie pakietów ACK (a nie na podstawie zawartości pola *ACK* zawartego w pakiecie ACK), ten sposób działania może spowodować, że nadawczy protokół TCP będzie zwiększał okno *cwnd* szybciej, niż czyniłby to normalnie. Ten problem może zostać złagodzony przez oparcie obliczeń związanych z kontrolą przeciążenia raczej na ilości potwierdzonych danych niż na fakcie przyjścia pakietu, jak to się dzieje w metodzie ABC.

Fałszowanie zduplikowanych potwierdzeń ACK powoduje, że nadawca zwiększa okno przeciążenia w czasie szybkiego odtwarzania. Przypomnijmy sobie z naszej wcześniejszej analizy, że w czasie standardowego szybkiego odtwarzania okno *cwnd* jest inkrementowane po każdym odebranym potwierdzeniu ACK. Atak wiąże się z tworzeniem dodatkowych zduplikowanych potwierdzeń ACK, które sprawiają, że dzieje się to szybciej, niż zostało zamierzone. Obrona przed tym atakiem jest trudniejsza, ponieważ nie istnieje prosty sposób przyporządkowania odebranych zduplikowanych potwierdzeń ACK do segmentów, które potwierdzają (identyfikator jednorazowy, *nonce* — dodatkowa wartość zmieniająca się w czasie, którą omawiamy w rozdziale 18., rozwiązałby ten problem). Chociaż opcja *Znaczniki czasu* odnosi się do tego problemu, jest to tylko opcja i może zostać wyłączona w zależności od połączenia. Najlepszym podejściem do rozwiązania tego problemu okazuje się być modyfikacja po stronie nadawcy, która ogranicza ilość danych pozostających w sieci w czasie odtwarzania.

Optymistyczne potwierdzanie wiąże się z generowaniem potwierdzeń ACK dla segmentów, które **jeszcze nie przyszły**. Ponieważ obliczenia związane z kontrolą przeciążenia są oparte na czasach RTT dla całej drogi transmisyjnej, potwierdzanie danych, które jeszcze nie dotarły, sprawia, że nadawca reaguje szybciej niż normalnie, ponieważ mylnie sądzi, że czasy RTT są krótsze niż w rzeczywistości. Co więcej, można to czynić prawie bezkarnie, ponieważ nadawca na ogół ignoruje potwierdzenia ACK dla danych, których jeszcze nie wysłał. Chociaż to podejście nie zachowuje niezawodności dostarczania danych na poziomie warstwy TCP w przeciwieństwie do pozostałych ataków (tzn. potwierdzone

dane nadal mogą zostać utracone), często mamy do czynienia z sytuacją (np. w przypadku protokołu HTTP/1.1), że brakujące dane mogą zostać odtworzone przez protokół warstwy aplikacji lub warstwy sesji. Autorzy opisują **kumulatywny identyfikator jednorazowy** (*cumulative nonce*), który może rozwiązać ten problem, oraz sposób zmieniania rozmiaru wysyłanych segmentów w czasie, aby lepiej skojarzyć potwierdzenia ACK z wysłanymi segmentami. Jeśli potwierdzenia ACK nie odpowiadają wysłanym segmentom, nadawca może podjąć stosowne działania.

Opisane wyżej problemy związane z niewłaściwym zachowaniem odbiorców stały się również przedmiotem zainteresowania niektórych autorów pracy [SCWA99] w aspekcie funkcjonowania mechanizmu ECN. Przypomnijmy sobie, że w systemie AQM wykorzystującym mechanizm ECN protokół TCP odbiorcy zwraca wskaźnik ECN do nadawcy w potwierdzeniu ACK. Wtedy nadawca powinien zareagować spowolnieniem transmisji. Jeśli odbiorcy nie uda się zwrócić sygnału ECN do nadawcy (lub routery w sieci skasują wskaźniki), nadawca nigdy nie zostanie poinformowany o przeciążeniu i nie spowolni wysyłania danych. W dokumencie [RFC3540] autorzy opisują eksperymentalny sposób wykorzystania bitu ECT (2-bitowego) pola ECN pakietu IP jako formy identyfikatora jednorazowego. Nadawca umieszcza losową wartość binarną w polu ECT, a odbiorca zwraca 1-bitową sumę (wynik operacji XOR) poszczególnych wartości przekazanych w tym polu w ciągu czasu. Generując potwierdzenie ACK, odbiorca umieszcza tę sumę w najmniej znaczącym bicie (NS) pola *Zarezerwowane* nagłówka TCP (patrz rozdział 12.). Niepoprawnie zachowujący się odbiorca ma 50% szans na odgadnięcie sumy. Ponieważ każdy pakiet reprezentuje niezależną próbę losową, a oszukujący odbiorca musi prawidłowo odgadnąć wszystkie sumy, aby odnieść sukces, jego szansa na osiągnięcia swojego celu wynosi $1/2^k$ dla k pakietów (znikomo mała w przypadku połączenia trwającego przez sensowny okres czasu).

16.13. Podsumowanie

Protokół TCP został zaprojektowany jako podstawowy niezawodny protokół transportowy dla Internetu. Chociaż jego początkowy projekt zawierał możliwość sterowania przepływem, wykorzystywaną do spowolnienia nadawcy w sytuacji, gdy odbiorca nie mógł nadążyć z odbieraniem danych, nie dostarczał on żadnych zabezpieczeń, które uniemożliwiałyby nadawcy zalewanie danymi znajdującej się pośrodku sieci. W późnych latach 80. ubiegłego wieku zostały opracowane algorytmy powolnego startu i unikania przeciążenia umożliwiające regulowanie agresywności nadawcy w celu uniknięcia gubienia pakietów z powodu występującego w sieci przeciążenia. Algorytmy te polegają na wykorzystaniu domyślnego sygnału, którym jest utrata pakietu, jako wskaźnika przeciążenia. Są one uruchamiane, kiedy zostaje wykryta utrata danych, albo przez algorytm szybkiej retransmisji, albo przez przekroczenie limitu czasu oczekiwania na retransmisję.

Algorytmy powolnego startu i unikania przeciążenia regulują działanie nadawcy przez wprowadzenie okna przeciążenia po stronie nadawcy. Jest ono używane w połączeniu z konwencjonalnym oknem (opartym na propozycjach okna dostarczanych przez odbiorcę). Standardowy protokół TCP ogranicza swoje okno do rozmiaru wyznaczonego przez mniejsze z tych dwóch okien. Powolny start powoduje wykładniczy wzrost rozmiaru okna w czasie, a algorytm unikania przeciążenia zwiększa okno mniej więcej liniowo w funkcji czasu. Tylko jeden z tych algorytmów jest aktywny w dowolnym czasie, a decyzja

dotycząca wyboru algorytmu jest podejmowana przez porównanie aktualnej wartości okna przeciążenia z progiem powolnego startu: jeśli okno przeciążenia przekracza wartość progu, sterowanie przejmuje algorytm unikania przeciążenia; w przeciwnym przypadku używany jest algorytm powolnego startu. Powolny start jest stosowany na początku, kiedy protokół TCP ustanawia swoje połączenie i w warunkach restartu wynikającego z przeterminowania. Algorytm ten może być również używany, kiedy połączenie pozostawało przez znaczący czas w stanie bezczynności. Próg powolnego startu jest dynamicznie regulowany w trakcie połączenia.

Kontrola przeciążenia była przedmiotem uwagi społeczności badaczy zagadnień sieciowych przez lata. Po tym, jak zebrano więcej doświadczeń na temat protokołu TCP, jego procedur powolnego startu i unikania przeciążenia, zaproponowano, zaimplementowano i zestandaryzowano szereg ulepszeń. Przez zachowywanie informacji o tym, kiedy protokół TCP wykonuje odtwarzanie po utracie grupy pakietów, wariant protokołu TCP o nazwie NewReno unika niektórych przestojów, które mogą wystąpić w wariantach Reno w sytuacji, gdy zostaje utraconych wiele pakietów z pojedynczego okna danych. Protokół TCP wykorzystujący opcje SACK może działać jeszcze lepiej niż NewReno, pozwalając nadawcy na inteligentną naprawę więcej niż jednej utraty pakietu w pojedynczym czasie RTT. W przypadku, gdy TCP używa opcji SACK, musi być prowadzone staranne rozliczanie, aby zagwarantować, że nadawca nie będzie nadmiernie agresywny w odniesieniu do innych protokołów TCP, z którymi może współdzielić ścieżkę w Internecie.

Niektóre z nowszych zmian w obsłudze przeciążenia przez protokół TCP obejmują dwukrotną redukcję tempa, walidację i moderację okna przeciążenia oraz procedury wycofywania zmian. Algorytm dwukrotnej redukcji tempa sprawia, że okno przeciążenia jest zmniejszane stopniowo, a nie natychmiast po wykryciu zdarzeń utraty danych. Walidacja okna przeciążenia próbuje zagwarantować, że okno przeciążenia nie będzie nadmiernie duże, gdy przez jakiś czas aplikacja była bezczynna lub nie mogła wysyłać danych. Moderacja okna przeciążenia ogranicza rozmiar nagłego wzrostu ruchu sieciowego w reakcji na odbiór pojedynczego potwierdzenia ACK. Procedury wycofywania zmian, takie jak algorytm odpowiedzi Eifel, wycofują modyfikacje okna przeciążenia, jeżeli sygnał utraty pakietu zostaje uznany za fałszywy (stan wykrywany przy użyciu szeregu technik). W takich przypadkach negatywny wpływ redukcji okna przeciążenia na wydajność jest minimalizowany przez przywrócenie stanu sprzed redukcji okna przeciążenia.

Po zdobyciu znaczącego doświadczenia związanego z protokołem TCP zauważono, że procedura unikania przeciążenia może potrzebować dużo czasu na znalezienie i wykorzystanie dodatkowego pasma, które staje się dostępne. W rezultacie zostały opracowane liczne propozycje wariantów protokołu TCP skalowalnych względem pasma. Jedną z najbardziej znanych wersji (opisanych w dokumentach IETF) jest protokół HSTCP, który pozwala na dużo bardziej agresywny wzrost okna przeciążenia w takich reżimach funkcjonowania, gdzie gubionych jest mało pakietów, a okna są duże w porównaniu do standardowego protokołu TCP. Kolejne sugestie obejmowały protokoły FAST i CTCP, które opierają swoje procedury wzrostu okna na utracie pakietów i pomiarach opóźnienia sieci. Szeroko stosowane w systemie Linux algorytmy BIC-TCP i CUBIC korzystają z funkcji wzrostu okna, które są wypukłe w pewnych fragmentach i wklęsłe w innych. Pozwala to na stosowanie w pobliżu punktu nasycenia małych zmian okna

prowadzących do zwiększonej stabilności przy możliwym koszcie nieco wolniejszej reakcji na pojawienie się nowego dostępnego pasma (ale wciąż szybszej niż w przypadku standardowego protokołu TCP).

Znacząca zmiana w działaniu protokołu TCP i routerów Internetu została zaproponowana przez specyfikację mechanizmu ECN — jawnego powiadomienia o przeciążeniu — który pozwoliłby protokołowi TCP na wykrycie pojawienia się przeciążenia przed doświadczeniem utraty pakietu. Chociaż symulacje i wyniki badań pokazały, że korzystanie z tego mechanizmu jest pożądane, wymaga ono jednak wprowadzenia umiarkowanej zmiany w implementacjach protokołu TCP i znacznej zmiany w sposobie działania routerów w Internecie. Zakres zastosowania tej możliwości pokaże przyszłość.

Chociaż protokół TCP dostarcza najszerzej używanej metody niezawodnego przesyłania danych w Internecie, nie implementuje wiele w kwestii swego własnego bezpieczeństwa. Jest generalnie podatny na ataki z fałszowaniem pakietów, które mogą spowodować zakłócanie połączeń; napastnik potrzebuje jedynie trafnego odgadnięcia „nadającego się" (tzn. mieszczącego się w oknie) numeru sekwencyjnego, żeby uruchomić takie ataki. W dodatku modyfikacja strumienia potwierdzeń `ACK` (lub bitów `ECN`, jeśli są obsługiwane) może sprawić, że nadawca będzie zachowywał się w sposób niesprawiedliwy względem innych połączeń TCP. Co więcej, nic fizycznie nie zapobiega prostemu łamaniu wszystkich reguł dotyczących kontroli przeciążenia przez zbyt agresywnego nadawcę.

Połączenie wszystkich tych algorytmów i technik opracowanych dla protokołu TCP w pojedynczą implementację tego protokołu nie jest łatwym zadaniem (stos protokołów TCP/IPv4 systemu Linux 2.6.38 to ok. 20 000 wierszy kodu w języku C), a analizowanie śladów działania protokołu TCP w rzeczywistych warunkach może być czasochłonne. Takie narzędzia jak `tcpdump`, Wireshark i `tcptrace` znacznie ułatwiają to zadanie. Ze względu na dynamiczne przystosowywanie się protokołu TCP do wydajności sieci, zrozumienie sposobu jego działania może zostać najłatwiej osiągnięte za pomocą technik wizualizacji opartych na wykresach szeregów czasowych, takich jak te, z których korzystaliśmy w tym rozdziale.

16.14. Bibliografia

[A10] M. Alizadeh i in., „Data Center TCP (DCTCP)", *Proc. ACM SIGCOMM*, sierpień – wrzesień 2010.

[ASA00] A. Aggarwal, S. Savage, T. Anderson, „Understanding the Performance of TCP Pacing", *Proc. INFOCOM*, marzec 2004.

[BP95] L. Brakmo, L. Peterson, *TCP Vegas: End to End Congestion Avoidance on a Global Internet*, „IEEE JSAC", październik 1995, nr 13(8).

[DHGS07] M. Dischinger, A. Haeberlen, K. Gummadi, S. Saroiu, „Characterizing Residential Broadband Networks", *Proc. ACM IMC*, październik 2007.

[F91] S. Floyd, *Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-Way Traffic*, „ACM Computer Communication Review", 1991, nr 21.

[FF96] S. Floyd, K. Fall, *Simulation-Based Comparisons of Tahoe, Reno, and SACK TCP*, „ACM Computer Communications Review", lipiec1996.

[FHPW00] S. Floyd, M. Handley, J. Padhye, J. Widmer, „Equation-Based Congestion Control for Unicast Applications", *Proc. ACM SIGCOMM*, sierpień 2000.

[FJ93] S. Floyd, V. Jacobson, *Random Early Detection Gateways for Congestion Avoidance*, „IEEE/ACM Transactions on Networking", sierpień 1993, nr 1(4).

[G11] J. Gettys, *Bufferbloat: Dark Buffers in the Internet*, „Internet Computing", maj/czerwiec 2011.

[G114] *One-Way Transmission Time*, International Telecommunication Union Recommendation G.114, maj 2003.

[H96] J. Hoe, „Improving the Start-up Behavior of a Congestion Control Scheme for TCP", *Proc. ACM SIGCOMM*, sierpień 1996.

[HRX08] S. Ha, I. Rhee, L. Xu, *CUBIC: A New TCP-Friendly High-Speed TCP Variant*, *http://netsrv.csc.ncsu.edu/export/cubic_a_new_tcp_2008.pdf*

[J88] V. Jacobson, „Congestion Avoidance and Control", *Proc. ACM SIGCOMM*, sierpień 1988. Referat ten został zaktualizowany w roku 1992. Współautorem tej aktualizacji jest M. Karels. Jest dostępna pod adresem *http://ee.lbl.gov/papers/congavoid.pdf*

[J90] V. Jacobson, *Modified TCP Congestion Avoidance Algorithm*, post wysłany na listę mailingową grupy end2end-interest, kwiecień 1990, dostępny pod adresem *ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt*

[K05] A. Kuzmanovic, „The Power of Explicit Congestion Notification", *Proc. ACM SIGCOMM*, sierpień 2005.

[KWNP10] C. Kreibich, N. Weaver, B. Nechaev, V. Paxson, „Netalyzr: Illuminating Edge Network Neutrality, Security and Performance", *Proc. ACM IMC*, listopad 2010.

[LARTC] *http://lartc.org*

[M92] J. Mogul, „Observing TCP Dynamics in Real Networks", *Proc. ACM SIGCOMM*, sierpień 1992.

[MM05] M. Mathis, przekaz osobisty, wrzesień 2005.

[MM96] M. Mathis, J. Mahdavi, „Forward Acknowledgment: Refining TCP Congestion Control", *Proc. ACM SIGCOMM*, sierpień 1996.

[NB08] J. Nievelt, V. Bhanu, *Developing TCP Chimney Drivers for Windows 7*, prezentacja na Microsoft Windows Drivers Developer Conference, 2008.

[NS2] *http://www.isi.edu/nsnam/ns* (patrz również NS3 pod adresem *http://www.nsnam.org*)

[P07] *http://lwn.net/Articles/128681*

[PSCRH] M. Mathis, J. Mahdavi, J. Semke, *TCP Rate Halving*, *http://tools.ietf.org/html/draft-mathis-tcp-ratehalving-00*

[RFC1122] R. Braden, red., *Requirements for Internet Hosts — Communication Layers*, Internet RFC 1122/STD 0003, październik 1989.

[RFC1812] F. Baker, red., *Requirements for IP Version 4 Routers*, Internet RFC 1812, czerwiec 1995.

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, *TCP Selective Acknowledgment Options*, Internet RFC 2018, październik 1996.

[RFC2140] J. Touch, *TCP Control Block Interdependence*, Internet RFC 2140, kwiecień 1997.

[RFC2309] B. Braden i in., *Recommendations on Queue Management and Congestion Avoidance in the Internet*, Internet RFC 2309 (informational), kwiecień 1998.

[RFC2861] M. Handley, J. Padhye, S. Floyd, *TCP Congestion Window Validation*, Internet RFC 2861 (experimental), czerwiec 2000.

[RFC3042] M. Allman, H. Balakrishnan, S. Floyd, *Enhancing TCP's Loss Recovery Using Limited Transmit*, Internet RFC 3042, styczeń 2001.

[RFC3124] H. Balakrishnan, S. Seshan, *The Congestion Manager*, Internet RFC 3124, czerwiec 2001.

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black, *The Addition of Explicit Congestion Notification (ECN) to IP*, Internet RFC 3168, wrzesień 2001.

[RFC3465] M. Allman, *TCP Congestion Control with Appropriate Byte Counting (ABC)*, Internet RFC 3465 (experimental), luty 2003.

[RFC3517] E. Blanton, M. Allman, K. Fall, L. Wang, *A Conservative Selective Acknowledgment (SACK)-Based Loss Recovery Algorithm for TCP*, Internet RFC 3517, kwiecień 2003.

[RFC3540] N. Spring, D. Wetherall, D. Ely, *Robust Explicit Congestion Notification (ECN) Signaling with Nonces*, Internet RFC 3540 (experimental), czerwiec 2003.

[RFC3649] S. Floyd, *HighSpeed TCP for Large Congestion Windows*, Internet RFC 3649 (experimental), grudzień 2003.

[RFC3742] S. Floyd, *Limited Slow-Start for TCP with Large Congestion Windows*, Internet RFC 3742 (experimental), marzec 2004.

[RFC3782] S. Floyd, T. Henderson, A. Gurtov, *The NewReno Modification toTCP's Fast Recovery Algorithm*, Internet RFC 3782, kwiecień 2004.

[RFC4015] R. Ludwig, A. Gurtov, *The Eifel Response Algorithm for TCP*, Internet RFC 4015, luty 2005.

[RFC5348] S. Floyd, M. Handley, J. Padhye, J. Widmer, *TCP Friendly Rate Control (TFRC): Protocol Specification*," Internet RFC 5348, wrzesień 2008.

[RFC5562] A. Kuzmanovic, A. Mondal, S. Floyd, K. Ramakrishnan, *Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets*, Internet RFC 5562 (experimental), czerwiec 2009.

[RFC5681] M. Allman, V. Paxson, E. Blanton, *TCP Congestion Control*, Internet RFC 5681, wrzesień 2009.

[RFC5690] S. Floyd, A. Arcia, D. Ros, J. Iyengar, *Adding Acknowledgement Congestion Control to TCP*, Internet RFC 5690 (informational), luty 2010.

[RFC6077] D. Papadimitriou, red., M. Welzl, M. Sharf, B. Briscoe, *Open Research Issues in Internet Congestion Control*, Internet RFC 6077 (informational), luty 2011.

[S09] B. Sonkoly, *Fairness and Stability Analysis of High Speed Transport Protocols*, praca doktorska, Budapeszteński Uniwersytet Techniczno-Ekonomiczny, 2009.

[SCWA99] S. Savage, N. Cardwell, D. Wetherall, T. Anderson, *TCP Congestion Control with a Misbehaving Receiver*, „ACM Computer Communication Review", kwiecień 1999.

[SK02] P. Sarolahti, A. Kuznetsov, „Congestion Control in Linux TCP", *Proc. Usenix Freenix Track*, czerwiec 2002.

[TCPTRACE] *http://www.tcptrace.org*

[TSZS06] K. Tan, J. Song, Q. Zhang, M. Sridharan, „A Compound TCP Approach for High-Speed and Long-Distance Networks", *Proc. INFOCOM*, kwiecień 2006.

[W08] X. Wu, *A Simulation Study of Compound TCP*, *http://cir.nus.edu.sg/reactivetcp/publication/ctcp_study.pdf*

[WJLH06] D. Wei, C. Jin, S. Low, S. Hegde, *FAST TCP: Motivation, Architecture, Algorithms, Performance*, „IEEE/ACM Trans. on Networking", marzec 2006.

[WQOS] *http://technet.microsoft.com/en-us/network/bb530836.aspx*

[WYSG05] R. Wang, K. Yamada, M. Sanadidi, M. Gerla, *TCP with Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes*, „IEEE JSAC", luty 2005, nr 23(2).

[XHR04] L. Xu, K. Harfoush, I. Rhee, „Binary Increase Congestion Control for Fast Long-Distance Networks", *Proc. INFOCOM*, marzec 2004.

Rozdział 17. Mechanizm podtrzymania aktywności w protokole TCP

17.1. Wprowadzenie

Wiele osób nieposiadających doświadczenia w zakresie protokołu TCP/IP jest zaskoczonych, kiedy dowiadują się, że przez bezczynne połączenie nie przepływają w ogóle żadne dane. To oznacza, że jeśli żaden z procesów działających na końcach połączenia TCP nie wysyła danych do drugiej strony, między dwoma punktami końcowymi TCP nie zachodzi jakakolwiek wymiana komunikatów. Nie ma np. żadnego odpytywania (*polling*), które można odnaleźć w innych protokołach sieciowych. Możemy zatem uruchomić proces klienta, który ustanawia połączenie TCP z serwerem i wyjść na kilka godzin, dni, tygodni lub miesięcy, a połączenie powinno pozostać uruchomione. Teoretycznie pośredniczące routery mogą ulec awarii i zostać uruchomione ponownie, linie transmisji danych mogą zostać wyłączone i ponownie włączone, ale dopóki żaden z hostów na końcach połączenia nie zostanie zrestartowany (lub nie zmieni się jego adres IP), połączenie pozostaje w stanie ustanowienia. W taki właśnie sposób został zaprojektowany protokół TCP/IP.

Powyższe stwierdzenie zakłada, że żadna z aplikacji — ani klient, ani serwer — nie korzysta z liczników czasu zaimplementowanych na poziomie aplikacji, które pomagałyby wykrywać brak aktywności, powodując zakończenie każdej z aplikacji. Zakłada ono również, że żaden pośredni router (taki jak urządzenie NAT) nie przechowuje informacji o stanie połączenia, koniecznej do właściwego funkcjonowania, którą mógłby usunąć z powodu nieaktywności lub utracić na skutek awarii systemu. We współczesnym Internecie są to daleko idące założenia.

W pewnych okolicznościach wiedza o zakończeniu lub utracie połączenia z odległym końcem może być użyteczna dla klienta lub dla serwera. W innych okolicznościach wskazane jest utrzymywanie przepływu minimalnej ilości danych przez połączenie nawet wtedy, gdy aplikacje nie mają żadnych danych do wymiany. Mechanizm **podtrzymania aktywności** (*keepalive*) dostarcza możliwości użytecznej w obu tych przypadkach. Podtrzymanie aktywności jest metodą stosowaną przez TCP do sondowania drugiej strony połączenia bez wpływania na zawartość strumienia danych. Jest ono sterowane licznikiem czasu podtrzymania aktywności. Kiedy licznik czasu sygnalizuje wyczerpanie się

limitu czasu, wysyłana jest **sonda podtrzymania aktywności** (zwana krócej **podtrzymaniem aktywności**), a strona połączenia odbierająca tę sondę odpowiada potwierdzeniem ACK.

Podtrzymania aktywności nie są częścią specyfikacji protokołu TCP. Dokument RFC określający wymagania dla hostów w Internecie [RFC1122] wyjaśnia to tym, że podtrzymania aktywności (1) mogłyby powodować przerwanie zupełnie dobrych połączeń w czasie przejściowych awarii w Internecie, (2) zużywają niepotrzebnie pasmo, (3) kosztują pieniądze w przypadku ścieżki Internetu, w której obowiązują opłaty za pakiet. Pomimo wszystko, większość implementacji oferuje możliwości korzystania z podtrzymań aktywności.

Podtrzymanie aktywności jest kontrowersyjnym mechanizmem. Wiele osób uważa, że odpytywanie drugiej strony nie powinno mieć miejsca w protokole TCP, ale powinno być wykonywane przez aplikację, jeśli jest to wymagane. Z drugiej strony, jeśli wiele aplikacji wymaga takiej funkcjonalności, wygodnie jest umieścić ją w protokole TCP, żeby jej implementacja mogła być współużytkowana. Podtrzymanie aktywności jest mechanizmem włączanym opcjonalnie, który może spowodować zakończenie skądinąd dobrego połączenia między dwoma procesami z powodu chwilowej utraty łączności w sieci łączącej dwa systemy końcowe. Jeśli np. sondy podtrzymania aktywności są wysyłane w czasie, gdy pośredni router jest w trakcie ponownego uruchomienia po wcześniejszej awarii, protokół TCP błędnie uważa, że awarii uległ host po drugiej stronie połączenia.

Mechanizm podtrzymania aktywności był pierwotnie przeznaczony dla aplikacji serwera, które mogłyby wiązać jakieś zasoby na potrzeby klienta i chciałyby być poinformowane, gdy host klienta ma awarię lub użytkownik odchodzi od komputera. Używanie mechanizmu podtrzymania aktywności protokołu TCP do wykrywania nieaktywnych klientów jest najbardziej pożyteczne w przypadku serwerów nastawionych na stosunkowo krótkotrwały dialog z nieinteraktywnym klientem (np. serwerów WWW, serwerów pocztowych obsługujących protokoły POP i IMAP). Serwery implementujące usługi w bardziej interaktywnym stylu, które trwają przez długi czas (np. zdalne logowanie, takie jak w przypadku ssh i zdalnego pulpitu systemu Windows), mogłyby unikać używania mechanizmu podtrzymywania aktywności.

Powszechnie wykorzystywanym przykładem pokazującym użyteczność mechanizmu podtrzymania aktywności we współczesnych warunkach jest sytuacja, w której użytkownik korzysta z programu zdalnego logowania ssh (bezpiecznej powłoki), aby zalogować się do zdalnego hosta przez router NAT. Jeżeli użytkownik ustanowiłby połączenie, wykonał pewną pracę, a następnie wyłączył komputer na zakończenie dnia bez wylogowania się, pozostałoby częściowo otwarte połączenie. W rozdziale 13. pokazaliśmy, że wysłanie danych przez połączenie częściowo otwarte spowodowało zwrócenie żądania resetowania, które jednak przyszło od punktu końcowego serwera, do którego klient wysyłał dane. Jeśli znika klient, pozostawiając połączenie częściowo otwarte po stronie serwera, a serwer oczekuje na jakieś dane od klienta, taki serwer będzie czekać w nieskończoność. Mechanizm podtrzymania aktywności jest przeznaczony do wykrywania takich częściowo otwartych połączeń po stronie serwera.

Inny powód używania podtrzymań aktywności dotyczy sytuacji w pewnym sensie odwrotnej. Jeżeli użytkownik nie wyłącza komputera, ale pozostawia otwarte połączenie na całą noc (i pragnie korzystać z niego następnego dnia), połączenie to pozostaje bezczynne

przez wiele godzin. W rozdziale 7. omawialiśmy sposób, w jaki większość routerów NAT wykorzystuje mechanizm przeterminowania, który czyści stan połączenia po pewnym okresie nieaktywności. Jeżeli limit czasu stosowany przez NAT jest mniejszy niż te kilka godzin, które upłyną, zanim użytkownik zacznie ponownie korzystać ze swojej sesji logowania, a router NAT nie jest wystarczająco inteligentny, aby sondować stację końcową w celu upewnienia się, że wciąż jest aktywna, lub gdy NAT ulegnie awarii, połączenie zostanie zakończone. Aby uniknąć tego często występującego problemu, program ssh może być skonfigurowany do używania podtrzymań aktywności protokołu TCP. Program ssh potrafi również używać podtrzymań aktywności **zarządzanych przez aplikację** (*application-managed*), a obydwa te mechanizmy funkcjonują w różny sposób, szczególnie pod względem swoich właściwości związanych z bezpieczeństwem. (Więcej informacji na ten temat można znaleźć w podrozdziale 17.3).

17.2. Opis

Każdy z końców połączenia TCP może dla swojego kierunku połączenia zażądać podtrzymań aktywności, które są domyślnie wyłączone. Podtrzymanie aktywności może być skonfigurowane dla jednej strony połączenia, obu stron lub nie być skonfigurowane dla żadnej z nich. Istnieje kilka konfigurowalnych parametrów, które kontrolują działanie podtrzymań aktywności. Jeżeli w połączeniu przez pewien okres czasu (wyznaczony przez parametr ***keepalive time***) brakuje aktywności, strona (strony) z włączonym podtrzymaniem aktywności wysyła (wysyłają) sondę (sondy) podtrzymania aktywności do drugiej strony. Jeśli nie zostanie odebrana żadna odpowiedź, sonda jest okresowo powtarzana z okresem wyznaczonym przez parametr ***keepalive interval***, dopóki liczba wysłanych sond nie osiągnie wartości ***keepalive probes***. Jeśli to nastąpi, system partnera zostaje uznany za niedostępny i połączenie jest kończone.

Sonda podtrzymania aktywności jest pustym (lub 1-bajtowym) segmentem z numerem sekwencyjnym o jeden mniejszym od największego numeru ACK dotąd otrzymanego od partnera. Ponieważ ten numer sekwencyjny został już potwierdzony przez odbiorczy protokół TCP, przychodzący segment nie wyrządza żadnej szkody, ale powoduje wysłanie potwierdzenia ACK, które jest wykorzystane do ustalenia, czy połączenie wciąż funkcjonuje. Ani sonda, ani jej potwierdzenie nie zawierają żadnych nowych danych (są to dane „śmieciowe") i nie są one retransmitowane przez TCP w razie ich utraty. Dokument [RFC1122] stwierdza, że ze względu na ten fakt brak odpowiedzi na pojedynczą sondę podtrzymania aktywności nie powinien być uważany za wystarczające świadectwo, że połączenie przestało funkcjonować. Jest to uzasadnienie dla ustawienia wcześniej wspomnianego parametru *keepalive probes*. Zwróćmy uwagę, że niektóre (przeważnie starsze) implementacje protokołu TCP nie odpowiadają na podtrzymania aktywności niezawierające „śmieciowego" bajta danych.

W dowolnym czasie swojego działania protokół TCP używający podtrzymań aktywności może zastać swojego partnera w jednym z czterech stanów.

- Host drugiego końca połączenia jest wciąż włączony, pracuje i jest dostępny. Protokół TCP odległego końca odpowiada w normalny sposób, a strona pytająca wie, że druga strona jest nadal czynna. TCP strony pytającej resetuje licznik czasu podtrzymania aktywności na późniejszy użytek (przypisując mu wartość parametru

keepalive time). Jeśli w połączeniu pojawi się ruch generowany przez aplikację, zanim kolejny licznik czasu wyzeruje się, licznik czasu zostanie ponownie ustawiony na wartość parametru *keepalive time*.

- Host na drugim końcu połączenia miał awarię i albo jest wyłączony, albo znajduje się w trakcie ponownego uruchamiania. W każdym z tych przypadków jego protokół TCP nie odpowiada. Strona pytająca nie otrzymuje odpowiedzi na swoją sondę, a po czasie określonym przez parametr *keepalive interval* następuje wyczerpanie się czasu oczekiwania. Strona pytająca wysyła w sumie tyle sond, ile wyznacza parametr *keepalive probes*, w odstępach czasu równych parametrowi *keepalive interval*, i jeśli nie otrzymuje odpowiedzi, uznaje, że host po drugiej stronie jest wyłączony i kończy połączenie.
- Host klienta uległ awarii, ale został uruchomiony ponownie. W tym przypadku serwer otrzymuje odpowiedź na swoją sondę podtrzymania aktywności, ale tą odpowiedzią jest segment z żądaniem resetowania, który powoduje zakończenia połączenia przez stronę pytającą.
- Host drugiej strony połączenia jest włączony i pracuje, ale jest niedostępny dla strony pytającej z jakiegoś powodu (np. sieć nie może obsłużyć ruchu i poinformować o tym fakcie stron połączenia przy użyciu komunikatu ICMP). Sytuacja ta ma taki sam skutek jak stan 2., ponieważ TCP nie potrafi odróżnić tych dwóch stanów. Wszystko, co TCP może stwierdzić, to brak odpowiedzi na jego sondy.

Strona pytająca nie musi martwić się o host drugiej strony, który zostaje łagodnie zamknięty, a następnie zrestartowany (w przeciwieństwie do nagłego zamknięcia spowodowanego awarią). Kiedy system jest zamykany przez operatora, kończone są wszystkie procesy aplikacji (tzn. procesy strony odpowiadającej), co powoduje, że TCP strony odpowiadającej wysyła sygnał `FIN` dla połączenia. Odebranie sygnału `FIN` sprawia, że protokół TCP strony pytającej zgłasza koniec pliku procesowi aplikacji, pozwalając jej na wykrycie właściwego scenariusza i wykonanie operacji wyjścia.

W pierwszym z opisanych stanów aplikacja strony pytającej nie ma pojęcia o tym, że wysyłane są sondy podtrzymania aktywności (z wyjątkiem sytuacji, gdzie aplikacja jako pierwsza podjęła decyzję o włączeniu podtrzymań aktywności). Wszystko jest obsługiwane na poziomie warstwy TCP. Proces ten jest przezroczysty dla aplikacji, dopóki nie zostanie ustalony jeden ze stanów: 2., 3. lub 4. W tych przypadkach do aplikacji strony pytającej zostaje zwrócony przez protokół TCP kod błędu. (Zazwyczaj strona pytająca ma zainicjowaną operację odczytu z sieci i oczekuje na dane od swojego partnera. Jeżeli mechanizm podtrzymania aktywności zwraca kod błędu, zostaje on przekazany stronie pytającej jako wartość zwrotna operacji odczytu). W scenariuszu 2. komunikat błędu jest tekstem w rodzaju „Połączenie zakończone na skutek przeterminowania", a w przypadku scenariusza 3. oczekujemy komunikatu „Połączenie zresetowane przez drugą stronę". Scenariusz 4. może wyglądać podobnie jak przy przeterminowaniu połączenia lub może spowodować zwrócenie innego błędu, w zależności od tego, czy został odebrany komunikat ICMP odnoszący się do tego połączenia (przekazujący informację o błędzie) i w jaki sposób jest on przetwarzany (patrz rozdział 8.). Wszystkim czterem scenariuszom przyjrzymy się w następnym podrozdziale.

Wartości zmiennych *keepalive time*, *keepalive interval* i *keepalive probes* zwykle mogą być zmieniane. Niektóre systemy umożliwiają wprowadzanie tych zmian na bazie poszczególnych połączeń, podczas gdy inne pozwalają na wykonywanie tych ustawień

tylko dla całego systemu (w pewnych przypadkach możliwe są oba warianty). W systemie Linux zmienne te są dostępne jako parametry polecenia `sysctl` o podanych kolejno nazwach: `net.ipv4.tcp_keepalive_time`, `net.ipv4.tcp_keepalive_intvl` i `net.ipv4.tcp_keepalive_probes`. Ich wartości domyślne odpowiednio wynoszą: 7200 sekund (2 godziny), 75 sekund i 9 (sond).

W systemach FreeBSD i Mac OS X pierwsze dwie zmienne są również dostępne jako parametry polecenia `sysctl` o nazwach `net.inet.tcp.keepidle` oraz `net.inet.tcp.keepintvl`, z wartościami domyślnymi wynoszącymi odpowiednio: 7 200 000 milisekund (2 godziny) i 75 000 milisekund (75 s). Systemy te posiadają także zmienną logiczną o nazwie `net.inet.tcp.always_keepalive`. Jeśli wartość tej zmiennej wynosi 1, wszystkie połączenia TCP mają włączoną funkcję podtrzymania aktywności nawet wtedy, gdy aplikacja jej nie zażądała. W systemach tych liczba sond ma stałą wartość domyślną: 8 (FreeBSD) lub 9 (Mac OS X).

W systemie Windows wartości te są dostępne do modyfikacji przez zapisy rejestru pod kluczem systemu:

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
```

Wartość domyślna parametru `KeepAliveTime` wynosi 7 200 000 ms (2 godziny), a parametr `KeepAliveInterval` ma wartość domyślną 1000 ms (1 s). Jeśli brakuje odpowiedzi dla dziesięciu sond podtrzymania aktywności, system Windows kończy połączenie.

Zauważmy, że dokument [RFC1122] nakłada pewne ograniczenia na używanie podtrzymań aktywności: parametr *keepalive time* musi być konfigurowalny, a jego wartość domyślna nie może być mniejsza niż 2 godziny; dodatkowo podtrzymania aktywności nie mogą być włączone, dopóki aplikacja nie zażąda wysłania sondy (chociaż ten sposób zachowania zostaje naruszony, jeśli ustawiona jest zmienna `net.inet.tcp.always_keepalive`). System Linux nie dostarcza wbudowanej możliwości dodawania podtrzymań aktywności do aplikacji, które tego nie żądają, ale może być **wstępnie zainstalowana** specjalna biblioteka (tzn. zainstalowana wcześniej niż zwykłe wspólne biblioteki), aby uzyskać ten efekt (patrz [LKA]).

17.2.1. Przykłady dotyczące podtrzymania aktywności

Przejdziemy teraz przez wszystkie stany opisane w tym podrozdziale, aby przyjrzeć się pakietom wymienianym w ramach działania mechanizmu podtrzymania aktywności. Działanie w stanie 1. zostanie zilustrowane w trakcie przyglądania się pozostałym przypadkom.

17.2.1.1. Drugi koniec połączenia ma awarię

Zobaczmy, co się dzieje, kiedy host serwera ulega awarii i nie zostaje uruchomiony ponownie. Aby zasymulować tę sytuację, wykonamy następujące kroki.

- Używając programu `regedit` w systemie Windows pełniącym rolę klienta, modyfikujemy klucz rejestru i ustawiamy wartość parametru `KeepAliveTime` na 7000 ms (7 s). Akceptacja nowej wartości może wymagać ponownego uruchomienia systemu.

- Ustanawiamy połączenie ssh między klientem w systemie Windows a serwerem uruchomionym w systemie Linux przy użyciu opcji, która włącza podtrzymania aktywności.
- Sprawdzamy, że dane mogą przepływać przez połączenie.
- Obserwujemy, jak TCP klienta wysyła co 7 s pakiety podtrzymania aktywności i jak są one potwierdzane przez TCP serwera.
- Odłączamy kabel sieciowy od serwera i pozostawiamy go w stanie odłączenia do zakończenia przykładu. To każe klientowi sądzić, że host serwera uległ awarii.
- Oczekujemy, że klient wyśle dziesięć sond podtrzymania aktywności w odstępach 1 s, zanim uzna połączenie za nieaktywne.

Oto interaktywny dialog po stronie klienta:

```
C:\> ssh -o TCPKeepAlive=yes 10.0.1.1
(następuje pytanie o hasło i kontynuacja logowania)
Write failed: Connection reset by peer (ok. 15 sekund po rozłączeniu)
```

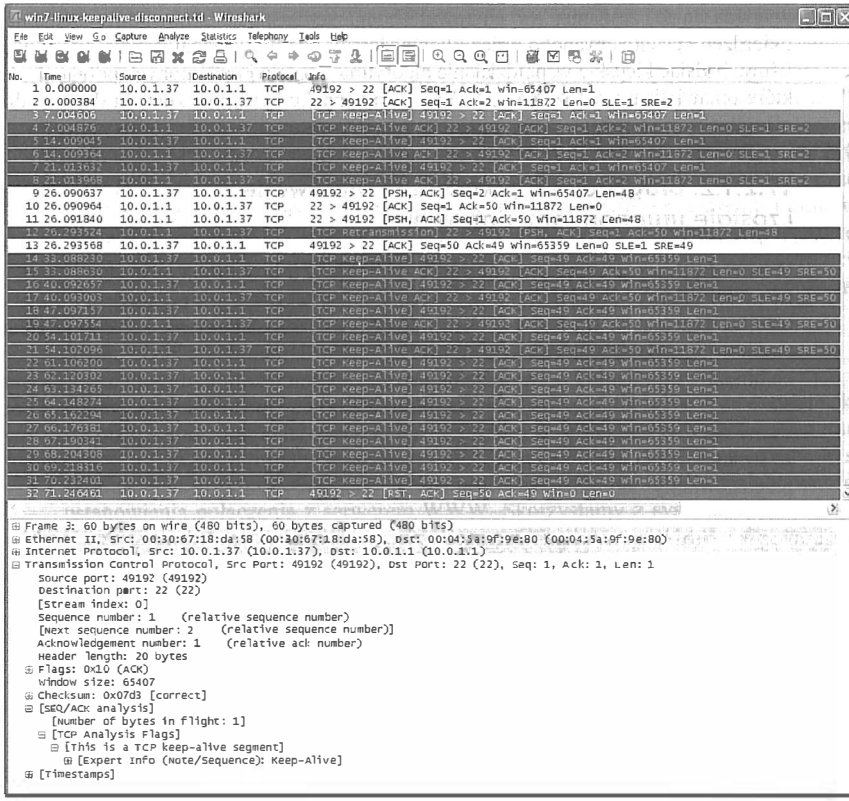
Na rysunku 17.1 pokazujemy wyniki przy użyciu programu Wireshark. W tym przykładzie połączenie zostało już ustanowione. Informacje wyświetlone przez program Wireshark rozpoczynają się od podtrzymania aktywności (pakiet 1.), które nie jest zidentyfikowane jako takie. W tym momencie program Wireshark nie przetworzył jeszcze wystarczającej ilości pakietów, aby ustalić, że pojedynczy numer sekwencyjny zawarty w pakiecie 1. znajduje się poniżej lewej krawędzi okna odbiorcy i dlatego jest podtrzymaniem aktywności. Pakiet 2. zawiera numer ACK, który umożliwia programowi Wireshark właściwe przetwarzanie numerów sekwencyjnych zawartych w kolejnych pakietach.

Większa część tego połączenia składa się z podtrzymań aktywności i odpowiadających im potwierdzeń ACK. Pakiety 1., 3., 5., 7., 14., 16., 18., 20. i od 22. do 31. są podtrzymaniami aktywności. Pakiety 2., 4., 6., 8., 15., 17., 19. i 21. to odpowiadające im potwierdzenia ACK. Podtrzymania aktywności są wysyłane okresowo co 7 s, przy założeniu, że są potwierdzane. Kiedy nie zostaje zwrócone potwierdzenie dla jednej z sond podtrzymania aktywności, nadawca przechodzi na interwał 1 s przy wysyłaniu kolejnych podtrzymań aktywności, zgodnie z domyślną wartością parametru KeepAliveInterval. Zaczyna się to od pakietu 23. w czasie 62,120. Nadawca generuje w sumie dziesięć niepotwierdzonych podtrzymań aktywności (pakiety od 22. do 31.), po czym kończy połączenie, co skutkuje wysłaniem końcowego segmentu z żądaniem resetowania (pakiet 32.), który nie zostanie nigdy odebrany przez odłączonego odbiorcę. Kiedy połączenie zostaje zakończone, użytkownik otrzymuje na wyjściu następujący komunikat:

```
Write failed: Connection reset by peer
```

Jest to wyraźna informacja o tym, że połączenie zostało zakończone, ale nie jest całkowicie dokładna. W rzeczywistości to nadawca zakończył połączenie, ale zrobił to na podstawie braku odpowiedzi od odbiorcy.

W tym połączeniu, oprócz segmentów podtrzymania aktywności, występują pewne inne interesujące szczegóły, o których krótko wspomnimy. Po pierwsze, serwer używa informacji DSACK (patrz rozdział 14.). Każde potwierdzenie ACK zawiera zakres numerów sekwencyjnych poprzednio odebranego segmentu, mieszczącego się w oknie. Następnie



Rysunek 17.1. *Podtrzymania aktywności protokołu TCP generowane są co 7 sekund po tym, jak połączenie staje się bezczynne. Każde z nich zawiera numer sekwencyjny lokujący się poniżej okna danych, który jest potwierdzany segmentem ACK przez drugą stronę połączenia. Odłączenie kabla po jednej minucie powoduje, że kolejne podtrzymania aktywności nie są potwierdzane. Klient podejmuje dziesięć prób przed rezygnacją i zakończeniem połączenia. Zakończenie jest sygnalizowane serwerowi przez końcowy segment z żądaniem resetowania (którego serwer nie może odebrać). Przykład ten ilustruje także użycie informacji DSACK przez serwer i zbędną retransmisję spowodowaną opóźnianiem potwierdzeń ACK przez klienta*

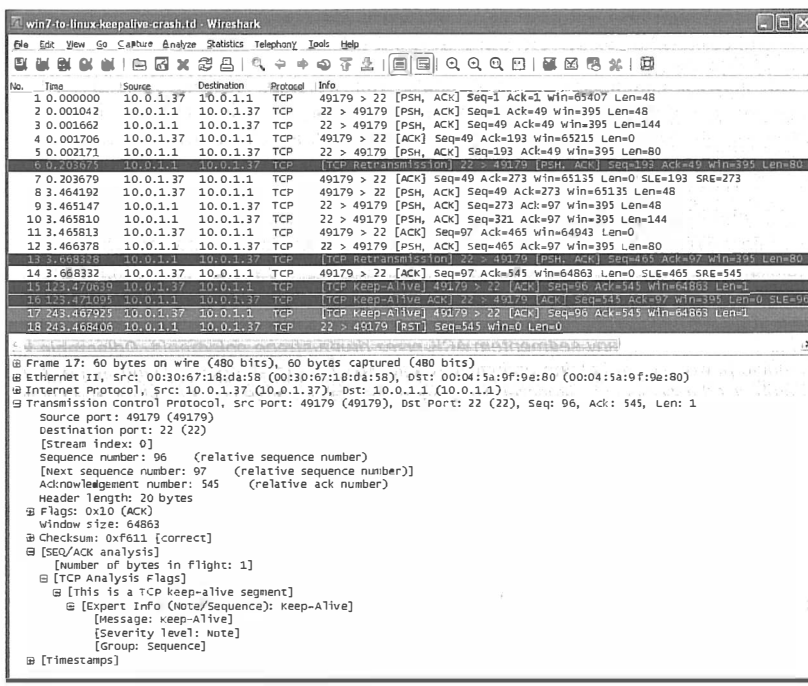
mała cząstka danych jest przesyłana w czasie 26,09. Dane te reprezentują pojedyncze naciśnięcie klawisza. Zostają one przesłane do serwera, potwierdzone przez serwer segmentem ACK i zostaje zwrócone ich echo. Dane są zaszyfrowane, co sprawia, że pakiety, które je zawierają, mają rozmiar 48 bajtów danych użytkownika (patrz rozdział 18.).

Co interesujące, echo znaku jest przesyłane dwukrotnie. Widzimy, że pakiet 11., który zawiera echo znaku, nie zostaje natycluniast potwierdzony. Przypominamy sobie z rozdziału 14., że Linux używa czasu RTO równego co najmniej 200 ms. Tu widzimy, że serwer linuksowy retransmituje echo znaku po 200 ms, co generuje natychmiastową odpowiedź klienta. Ponieważ opisywany test został przeprowadzony w nieobciążonej sieci LAN, jest wysoce nieprawdopodobne, że segment 11. został odrzucony. Natomiast

wygląda na to, że Linux wygenerował zbędną retransmisję z powodu opóźniania potwierdzeń ACK, stosowanego przez klienta. Jest to podobny rodzaj ryzyka, który widzieliśmy, badając kiepskie współdziałanie algorytmu Nagle'a i opóźnionych potwierdzeń ACK, które omawialiśmy w rozdziale 15. W tym przypadku dynamika zdarzeń doprowadza do niepotrzebnego opóźnienia wynoszącego ok. 200 ms.

17.2.1.2. Drugi koniec połączenia ma awarię i zostaje uruchomiony ponownie

W tym przykładzie zobaczymy, co się dzieje, kiedy druga strona połączenia ulega awarii i jest restartowana. Początkowy scenariusz jest taki sam jak poprzedni, z wyjątkiem tego, że tym razem wartość parametru KeepAliveTime ustalamy na 120 000 (2 minuty). Ustanawiamy połączenie, a następnie odczekujemy do momentu, kiedy upłyną 2 minuty, co umożliwi wysłanie i potwierdzenie przez ACK komunikatu podtrzymania aktywności. Następnie odłączamy serwer od sieci, restartujemy go i podłączamy ponownie. Spodziewamy się, że kolejna sonda podtrzymania aktywności wygeneruje żądanie resetowania ze strony serwera, ponieważ teraz serwer nic nie wie o tym połączeniu. Na rysunku 17.2 przedstawiamy ślad transmisji wyświetlony przez program Wireshark.



Rysunek 17.2. *Serwer został zrestartowany w czasie pomiędzy kolejnymi podtrzymaniami aktywności wysłanymi przez klienta. Ostatnie podtrzymanie aktywności powoduje wysłanie segmentu z żądaniem resetowania przez serwer, który już nic nie wie o połączeniu*

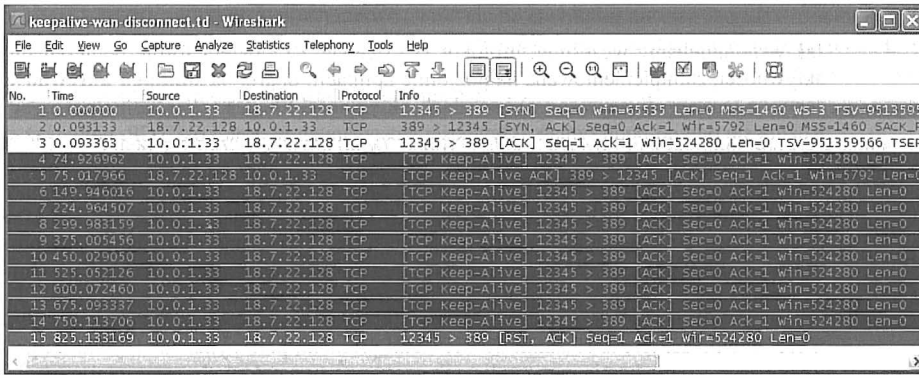
W tym przykładzie połączenie zostało ustanowione i dochodzi do dwóch wymian małej ilości danych, rozpoczynających się w czasach 0,00 i 3,46 s. Następnie połączenie staje się bezczynne. Po upływie 2 minut (zgodnie z parametrem *keepalive time*) klient wysyła pierwszą sondę podtrzymania aktywności w czasie 123,47 zawierającą „śmieciowy" bajt o numerze sekwencyjnym poniżej lewej krawędzi okna. Po jej potwierdzeniu serwer zostaje odłączony, zrestartowany i ponownie podłączony. W czasie 243,47, czyli 120 sekund później, klient wysyła swoją drugą sondę podtrzymania aktywności. Chociaż sonda ta dociera do miejsca przeznaczenia, serwer już nic nie wie o połączeniu i odpowiada wysłaniem segmentu RST (pakiet 18.). To informacja dla klienta o tym, że połączenie nie jest już aktywne, i w efekcie użytkownik otrzymuje komunikat błędu `Connection reset by peer` (połączenie zresetowane przez drugą stronę), który już widzieliśmy wcześniej.

17.2.1.3. Drugi koniec połączenia jest niedostępny

W tym przypadku serwer nie uległ awarii, ale staje się niedostępny w okresie, kiedy wysyłane są sondy podtrzymania aktywności. Mogło dojść do awarii pośredniego routera, chwilowej awarii linii telefonicznej czy czegoś podobnego. Aby zasymulować tę sytuację, użyjemy naszego programu `sock` z ustawioną opcją podtrzymania aktywności do ustanowienia połączenia z serwerem WWW. Skorzystamy z systemu Mac OS X jako klienta oraz z serwera LDAP (port 389) działającego na hoście `ldap.mit.edu`. Po skróceniu czasu ustawionego w parametrze *keepalive time* klienta (dla wygody) i otwarciu połączenia odłączamy sieć, aby zobaczyć skutki. Oto wiersze poleceń i dane wyjściowe wyświetlone na terminalu klienta:

```
Mac# sysctl -w net.inet.tcp.keepidle=75000
Mac% sock -K ldap.mit.edu 389
recv error: Operation timed out        ok. 14 minut później
```

Poniższy ślad został wyświetlony przy użyciu programu Wireshark (patrz rysunek 17.3).



Rysunek 17.3. *Połączenie WAN zostało przerwane po potwierdzeniu pierwszej sondy podtrzymania aktywności. Kolejne sondy są wysyłane co 75 s. Po wysłaniu dziewięciu podtrzymań aktywności bez odpowiedzi połączenie zostaje zakończone, a klient wysyła żądanie resetowania do swojego partnera. Dla klienta sytuacja jest bardzo podobna do tej, kiedy serwer ulega awarii, co można zobaczyć na rysunku 17.1*

Na rysunku 17.3 widzimy połączenie w całości. Po początkowym uzgadnianiu trójetapowym połączenie pozostaje bezczynne i mniej więcej w czasie 75 s zostaje wysłane i potwierdzone podtrzymanie aktywności (pakiet 4.). To pierwsze podtrzymanie aktywności jest uruchomione na podstawie wartości zmiennej `net.inet.tcp.keepidle`. Wkrótce potem sieć zostaje odcięta. Żaden z końców połączenia nie generuje danych, więc następnym zdarzeniem jest kolejne podtrzymanie aktywności, wysłane przez klienta w czasie 150 (czyli 75 s później, zgodnie z wartością zmiennej `net.inet.tcp.keepintvl`). Ten schemat czynności powtarzają pakiety od 7. do 14., bez obecności potwierdzeń ACK, mimo że serwer jest włączony i pracuje. W końcu po upływie 75 s klient rezygnuje (po swojej dziewiątej niepotwierdzonej sondzie podtrzymania aktywności). Zakończenie połączenia jest sygnalizowane serwerowi przez wysłanie segmentu z żądaniem resetowania (końcowy pakiet 15.). Oczywiście, serwer nie może odebrać tego pakietu, ponieważ nie działa sieć.

Kiedy klient TCP korzystający z podtrzymań aktywności nie może skomunikować się przez sieć z drugą stroną połączenia, co pokazujemy w tym przykładzie, ponawia próby pewną ilość razy przed ostateczną rezygnacją. Jest to zasadniczo to samo zachowanie, które widzieliśmy, kiedy drugi koniec połączenia uległ awarii. W większości przypadków protokół TCP nadawcy nie potrafi zidentyfikować różnicy. Są pewne wyjątki, takie jak wówczas, kiedy komunikat ICMP wskazuje, że host docelowy stał się niedostępny w taki czy inny sposób z powodu problemów występujących w sieci, ale te warunki zachodzą stosunkowo rzadko, ponieważ komunikaty ICMP często są blokowane. W rezultacie do wykrywania okresów braku połączenia są wykorzystywane takie mechanizmy jak podtrzymanie aktywności protokołu TCP (lub podobne mechanizmy implementowane przez aplikacje).

17.3. Ataki związane z mechanizmem podtrzymania aktywności protokołu TCP

Jak wspomnieliśmy wcześniej, protokół `ssh` (wersja 2.) zawiera formę podtrzymania aktywności, zaimplementowaną na poziomie aplikacji, opartą na **sondach aktywności serwera** (*server alive messages*) i **sondach aktywności klienta** (*client alive messages*). Różnią się one od podtrzymań aktywności stosowanych w protokole TCP, ponieważ są przesyłane przez zaszyfrowany kanał w warstwie aplikacji i zawierają dane. Podtrzymania aktywności protokołu TCP nie zawierają żadnych danych na poziomie użytkownika, więc użycie szyfrowania ma tu w najlepszym razie ograniczone zastosowanie. W konsekwencji, podtrzymania aktywności TCP mogą być fałszowane. W sytuacji, gdy podtrzymania aktywności TCP są fałszowane, ofiara ataku może być zmuszona do zachowywania przydzielonych zasobów przez niepotrzebnie długi okres.

Choć, być może, jest to stosunkowo niewielki problem, podtrzymania aktywności protokołu TCP są uruchamiane przez licznik czasu oparty na różnych, wcześniej omówionych parametrach konfiguracyjnych, a nie przez dynamicznie regulowany licznik czasu retransmisji, używany do retransmitowania segmentów z danymi. Pasywny obserwator mógłby zauważyć występowanie sond aktywności, odnotować czasy między ich kolejnymi przyjściami i, co jest do wyobrażenia, uzyskać informacje o parametrach konfiguracyjnych (być może zidentyfikować typ systemu nadawcy; taka technika została na-

zwana **fingerprintingiem** przez skojarzenie z analizą odcisków palca) lub o topologii sieci (tzn. czy podrzędne routery przekazują ruch dalej, czy nie). Te kwestie w pewnych środowiskach mogłyby stanowić problem.

17.4. Podsumowanie

Jak powiedzieliśmy wcześniej, mechanizm podtrzymania aktywności jest nieco kontrowersyjny. Eksperci w dziedzinie protokołów wciąż dyskutują, czy należy on do warstwy transportowej, czy też powinien być w całości obsługiwany przez aplikację. Wszystkie popularne implementacje protokołu TCP zawierają obecnie mechanizm podtrzymania aktywności, który może być opcjonalnie wykorzystany przez aplikacje do ustanowienia „tętna" ruchu przechodzącego przez połączenie. Jego użycie może pomóc serwerowi przez umożliwienie mu wykrycia nieodpowiadających klientów oraz może pomóc klientom przez utrzymywanie aktywności połączeń (np. utrzymywanie stanu aktywnego w urządzeniu NAT) nawet wtedy, gdy nie przepływają żadne dane warstwy aplikacji.

Działanie podtrzymań aktywności polega na wysyłaniu pakietu sondy (zawierającego zwykle bajt „śmieciowy", choć możliwe są również sondy zerowej długości) w połączeniu, które przedtem pozostawało bezczynne przez pewien, stosunkowo długi okres czasu, często wynoszący 2 godziny. Mogą wystąpić cztery różne scenariusze: druga strona połączenia pozostaje w gotowości, system drugiej strony uległ awarii, system po drugiej stronie połączenia został zrestartowany po wcześniejszej awarii lub druga strona jest aktualnie niedostępna. Każdy z tych scenariuszy został zilustrowany przykładem.

Odnośnie dwóch pierwszych zbadanych przez nas przykładów działania mechanizmu podtrzymania połączenia możemy stwierdzić, że jeśli nie byłyby użyte podtrzymania aktywności, to bez implementacji licznika czasu i stosownego działania na poziomie aplikacji protokół TCP nigdy nie dowiedziałby się, że druga strona połączenia uległa awarii (lub została uruchomiona ponownie po chwilowej awarii). Jednak w ostatnim przykładzie z drugą stroną nie działo się nic złego; chwilowo nieczynne było połączenie. Używając podtrzymań aktywności, musimy być świadomi tego ograniczenia i rozważyć, czy takie działanie jest pożądane, czy też nie.

Ataki przeciw mechanizmowi podtrzymania aktywności obejmują spowodowanie, że system zatrzymuje przydzielone zasoby dłużej niż to jest zamierzone, oraz możliwość uzyskania, w innym przypadku ukrytej, informacji o systemach końcowych (chociaż taka informacja może mieć ograniczoną użyteczność dla napastnika). W dodatku protokół TCP nie używa swego własnego szyfrowania, więc podtrzymania aktywności i potwierdzenia ACK dla podtrzymań aktywności mogą być fałszowane, podczas gdy podtrzymania aktywności zaimplementowane na poziomie aplikacji, które stosują szyfrowanie (np. ssh), sfałszowane być nie mogą.

17.5. Bibliografia

[LKA] *http://libkeepalive.sourceforge.net*

[RFC1122] Braden R., red., *Requirements for Internet Hosts*, Internet RFC 1122, październik 1989.

Rozdział 18. Bezpieczeństwo — EAP, IPsec, TLS, DNSSEC oraz DKIM

18.1. Wprowadzenie

W tym rozdziale przeanalizujemy kilka mechanizmów zabezpieczeń towarzyszących protokołom TCP/IP. Bezpieczeństwo jest niezwykle rozległym i interesującym tematem, którego pełne omówienie wykracza poza ramy tematyczne książki. Skoncentrujemy się więc na różnych zagrożeniach występujących w Internecie oraz na szczegółowym omówieniu zabezpieczeń znajdujących zastosowanie w działaniu takich protokołów jak IP i TCP oraz protokołów aplikacji pocztowych i DNS.

Choć klasyfikacja ta nie ma formalnego charakteru, zagrożenia sieciowe można podzielić na ataki skierowane przeciwko implementacjom (mające na celu wymuszenie na uruchomionych procesach działania innego niż założone przez twórcę programu), użytkownikom (w celu nakłonienia ich do wykorzystania niebezpiecznych programów) oraz protokołom sieciowym (wykonującym działania w poprawny, ale nieautoryzowany sposób). Jeden z pierwszych **robaków** internetowych (program zdolny do samodzielnego rozprzestrzeniania się w sieci) wykorzystywał mechanizm **przepełnienia bufora** do nadpisywania pamięci przydzielonej procesom serwera. Program kliencki mógł wtedy wstrzyknąć kod i wymusić na serwerze jego wykonanie. Dostarczony kod ponownie wykonywał tę operację i rozprzestrzeniał się automatycznie w sieci.

W praktyce często spotykane jest łączenie różnych rodzajów i technik ataków. Dlatego wraz ze wzrostem ilości informacji wymienianych za pośrednictwem Internetu konieczne okazało się opracowanie narzędzi analizujących zagrożenia sieciowe. Programy te są szczegółowo opisywane w wielu opracowaniach, w tym w [MSK09]. Obecnie niemal każde oprogramowanie uruchomione przez użytkownika (lub w jego imieniu) skierowane przeciwko temu użytkownikowi jest określane terminem **malware** pochodzącym od angielskich słów oznaczających „złośliwe oprogramowanie" (*malicious software*). Tworzeniem i walką ze skutkami działania tego rodzaju programów zajmują się zupełnie nowe gałęzie przemysłu komputerowego. Kod malware może być dostarczany za pośrednictwem wiadomości e-mail (np. w spamie), pobierany wraz ze stronami internetowymi lub kopiowany podczas korzystania z urządzeń przenośnych, takich jak dyski USB.

W pewnych przypadkach złośliwe oprogramowanie jest stosowane do przejęcia kontroli nad większą liczbą komputerów w Internecie (stanowiących **botnet**). Botnety są nadzorowane przez pojedyncze osoby lub przez organizacje i mogą być wykorzystywane na wielką skalę do realizacji takich zadań jak rozsyłanie spamu, włamywanie się do komputerów, wyszukiwanie informacji w zaatakowanych systemach (np. numerów kart kredytowych i kont bankowych oraz sekwencji klawiszy naciskanych przez użytkownika), realizowanie ataków DDoS (generowanie dużej liczby żądań skierowanych do jednego komputera docelowego lub większej liczby takich komputerów). Obecnie botnety są dostępne również w formie zamawianej usługi — klient wynajmuje zarządcę botnetu do wykonania określonego zadania. Jedną z najczęstszych operacji jest generowanie wiadomości e-mail, które mają na celu nakłonienie odbiorców do odwiedzenia wskazanej strony internetowej lub kupienia pewnego produktu (*phishing*). Skierowanie opisanych działań przeciwko konkretnej osobie określa się zazwyczaj terminem *spear phishing*.

Celem tego rozdziału jest przedstawienie zasad działania bezpiecznych protokołów internetowych, choć — jak na ironię — wiele robaków i wirusów wykorzystuje w działaniu protokoły zapewniające bezpieczną komunikację. Większość omawianych zagadnień dotyczy rozbudowy analizowanych wcześniej mechanizmów IP, TCP, e-mail i DNS o elementy zwiększające bezpieczeństwo pracy sieciowej (niekiedy w formie dodatkowych protokołów). Aby zrozumieć poszczególne rozwiązania oraz zapewniane przez nie poziomy zabezpieczeń, trzeba najpierw zdefiniować znaczenie słowa „bezpieczeństwo" w odniesieniu do protokołów komunikacyjnych. Dlatego prezentacja zagadnienia musi się rozpocząć od analizy cech systemu zabezpieczeń, które są istotne z punktu widzenia **bezpieczeństwa informacji**.

18.2. Bezpieczeństwo informacji — podstawowe założenia

Informacje mają trzy cechy istotne dla ich bezpieczeństwa, niezależnie od tego, czy są przekazywane przez sieć komputerową, czy nie. Oto one.

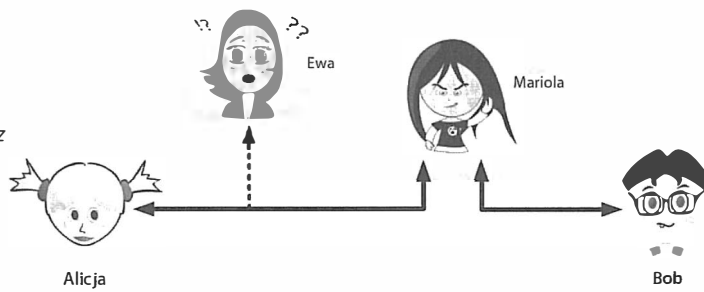
- **Poufność** — informacja jest znana tylko odbiorcom, dla których jest przeznaczona (odbiorcami mogą być również systemy przetwarzania danych).
- **Integralność** — informacja nie może być modyfikowana w sposób nieuprawniony przed dostarczeniem jej do odbiorcy.
- **Dostępność** — informacja musi być dostępna wtedy, gdy jest potrzebna.

Cechy te należą do najważniejszych właściwości informacji, choć można wyróżnić jeszcze kilka, w tym **uwierzytelnienie**, **niezaprzeczalność** oraz **audytowalność**. Uwierzytelnienie oznacza, że określona strona komunikacji (**podmiot zabezpieczeń** — *security principal*) nie podszywa się pod inną stronę. Niezaprzeczalność oznacza z kolei, że każde działanie podjęte przez jedną ze stron (np. zaakceptowanie warunków kontraktu) może być udowodnione w późniejszym czasie (tzn. strona nie może się wyprzeć swoich działań). Audytowalność oznacza natomiast możliwość sprawdzenia sposobu wykorzystania informacji zazwyczaj za pomocą wiarygodnego dziennika zdarzeń. Dzienniki zdarzeń okazują się również bardzo pomocne podczas działań dochodzeniowych (np. prokuratorskich).

Założenia te odnoszą się także do informacji przekazywanych w formie fizycznej (np. drukowanych). Dlatego od setek lat wykorzystuje się sejfy, strzeżone obiekty i strażników do zabezpieczania wymiany informacji, ich przechowywania i rozpowszechniania. Gdy informacje są przekazywane w niebezpiecznym środowisku, konieczne jest użycie dodatkowych technik ochrony. Aby zrozumieć, dlaczego tak być musi, przeanalizujemy rodzaje zagrożeń, na które informacje są narażone podczas przesyłania w niezabezpieczonym kanale komunikacyjnym.

18.3. Zagrożenia w komunikacji sieciowej

Zapewnienie, że budowa i działanie protokołu sieciowego spełni założenia dotyczące integralności, dostępności i poufności, może być sporym wyzwaniem. W nienadzorowanej sieci (jaką jest Internet) można spodziewać się wielu różnych ataków. Same ataki są klasyfikowane jako **pasywne** lub **aktywne** [VK83]. Zidentyfikowanie kategorii jest istotne, ponieważ od niego zależy zastosowanie określonej techniki obrony. Ataki pasywne polegają na monitorowaniu lub przechwytywaniu ruchu sieciowego. Nieuwzględnienie ich w zabezpieczeniach może doprowadzić do ujawnienia informacji (utraty poufności). Z kolei ataki aktywne skutkują zmianami w przekazywanych informacjach (utratą integralności) lub odmową wykonania usługi (utratą dostępności). Oba rodzaje ataków są zazwyczaj wykonywane przez „włamywaczy". Na rysunku 18.1 ilustrujemy opisane działania.



Rysunek 18.1. *Strony komunikacji (Alicja i Bob) próbują bezpiecznie przekazać informację, która może zostać podsłuchana przez Ewę lub zmodyfikowana przez Mariolę*

Na rysunku przedstawiono dwa podmioty zabezpieczeń (Alicję i Boba) komunikujące się ze sobą oraz dwie osoby atakujące (Ewę i Mariolę). Ewa (osoba podsłuchująca) może jedynie monitorować wymianę danych między Alicją i Bobem. Realizuje więc atak pasywny. Mariola może natomiast modyfikować i powielać dane wymieniane między Alicją i Bobem. Jest zatem zdolna do przeprowadzenia ataków pasywnego i aktywnego. W tabeli 18.1 zostały przedstawione najważniejsze kategorie ataków pasywnych i aktywnych, z którymi Alicja i Bob muszą się liczyć.

Tabela 18.1 jest ogólnym zestawieniem technik ataków pasywnych możliwych do zrealizowania przez Ewę oraz ataków aktywnych, które mogą zostać wykonane przez Mariolę. Ewa ma możliwość **podsłuchania** informacji (podsłuchiwanie jest również nazywane **przechwytywaniem** lub **sniffingiem** pakietów) oraz **przeanalizowania ruchu** przekazywanego między Alicją i Bobem. Przechwycenie ruchu może oznaczać utratę poufności, ponieważ zastrzeżone informacje trafiają do Ewy bez wiedzy Alicji i Boba.

Tabela 18.1. *Ataki odnoszące się do komunikacji między użytkownikami są klasyfikowane jako pasywne i aktywne. Ataki pasywne są trudniejsze do wykrycia, natomiast ataki aktywne są trudniejsze do wyeliminowania*

Ataki pasywne		Ataki aktywne	
Typ	Zagrożone	Typ	Zagrożone
Podsłuch	Poufność	Modyfikacja strumienia komunikatów	Uwierzytelnienie, integralność
Analiza ruchu	Poufność	Odmowa usługi (DoS)	Dostępność
		Fałszywe skojarzenie	Uwierzytelnienie

Dodatkowo analiza ruchu pozwala osobie atakującej na określenie pewnych cech komunikacji, takich jak rozmiar danych, czas wysyłania oraz elementy identyfikujące strony komunikacji. Choć dane te nie ujawniają treści komunikacji, bywają użyteczne w gromadzeniu informacji przydatnych w przyszłości do opracowania bardziej wyrafinowanego ataku.

Ataki pasywne są w zasadzie niewykrywalne dla Alicji i Boba. Działania Marioli można jednak znacznie łatwiej zauważyć, gdyż polegają one na **modyfikacji strumienia komunikatów** (MSM — *Message Stream Modification*), **odmowie usługi** (DoS — *Denial of Service*) lub **fałszywym skojarzeniu**. Ataki MSM (obejmujące wszelkie działania typu **człowiek pośrodku** [MITM — *Man In The Middle*]) wyznaczają bardzo rozległą kategorię operacji, w których przekazywany ruch sieciowy podlega takim modyfikacjom jak usuwanie, zmiana kolejności komunikatów oraz zmiana treści komunikatów. Ataki typu DoS mogą doprowadzić do usunięcia pakietów z sieci lub wygenerowania ruchu o dostatecznie dużym natężeniu, żeby przeciążyć komputer Alicji lub Boba bądź kanał komunikacyjny, z którego korzystają. Technika fałszywego skojarzenia obejmuje **maskaradę** (Mariola udaje Boba lub Alicję) oraz **odtwarzanie** komunikatów, które polega na wysyłaniu autentycznych komunikatów Alicji lub Boba z pamięci utrzymywanej w jednostce Marioli (w późniejszym czasie).

Zapobieganie pasywnym i aktywnym atakom wymaga zastosowania jednej z dwóch metod postępowania. Pierwsza z nich zakłada fizyczne zabezpieczenie komunikacji przez zapewnienie dostępu do infrastruktury przesyłowej jedynie zaufanym podmiotom. Rozwiązanie to znajduje zastosowanie w zamkniętych systemach, ale nie nadaje się do użycia w sieciach o dużym zasięgu geograficznym. W przypadku bezprzewodowych form komunikacji rozwiązanie to jest w zasadzie niemożliwe do wdrożenia. W związku z opisanymi trudnościami konieczne było opracowanie mechanizmu, który pozwoliłby na przekazywanie informacji przez niezabezpieczony kanał w taki sposób, aby próby osób atakujących (takich jak Ewa i Mariola) kończyły się niepowodzeniem (przynajmniej w większości). Takim mechanizmem jest **kryptografia**. Właściwe wykorzystanie technik kryptograficznych daje gwarancję, że ataki pasywne staną się nieefektywne, a ataki aktywne łatwe do wykrycia (oraz że w pewnym stopniu będzie można im zapobiegać).

18.4. Podstawowe mechanizmy kryptograficzne i zabezpieczające

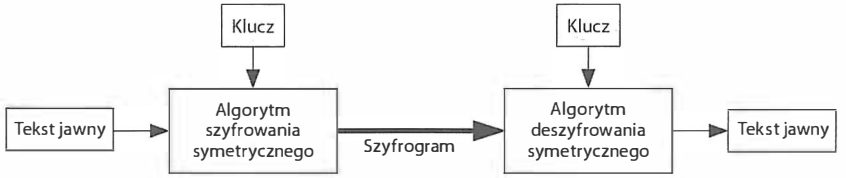
Rozwój kryptografii wynika z potrzeby ochrony poufności, integralności i autentyczności informacji przenoszonych w niezabezpieczonych kanałach komunikacyjnych. Cechy te są niezbędne w systemach odpowiedzialnych za zabezpieczanie tajnych informacji, takich jak rozkazy wojskowe, dane wywiadowcze lub procedury tworzenia wyjątkowo niebezpiecznych lub wartościowych materiałów. Kryptografia w swej najbardziej prymitywnej postaci była wykorzystywana już 3500 lat przed naszą erą. Pierwsze rozwiązania bazowały na stosowaniu odpowiednich **kodów**. Kodowanie polega na zastępowaniu grup słów, fraz lub zdań grupami cyfr lub liter z określonej **książki kodowej**. Utrzymanie poufności komunikacji wymaga przechowywania książek kodowych w bezpiecznych miejscach, a ich dystrybucja wiąże się z zachowaniem szczególnej ostrożności.

Bardziej zaawansowane systemy wykorzystywały **szyfry**, w których oprócz podmiany stosuje się również przestawianie tekstu. W średniowieczu znanych było kilka systemów kodowania i szyfrowania, a w końcu XIX wieku większość komunikacji dyplomatycznej i wojskowej była przetwarzana przez rozbudowane systemy kodujące i szyfrujące. Na początku XX wieku kryptografia była już rozwiniętą dziedziną nauki, ale dopiero w czasie II wojny światowej istotnie zyskała na znaczeniu. Opracowane w tym czasie niemieckie elektromechaniczne maszyny szyfrujące ENIGMA i maszyna Lorenza stały się wyzwaniem dla **analityków** (łamaczy kodów). Jeden z pierwszych cyfrowych komputerów — Colossus — został opracowany przez Brytyjczyków właśnie do rozszyfrowania komunikatów generowanych przez maszynę Lorenza. Działająca maszyna Colossus Mark 2 została zrekonstruowana w 2007 roku (po 14 latach prac) przez Tony'ego Sale'a z Narodowego Muzeum Komputerów w Bletchley Park w Wielkiej Brytanii [TNMOC].

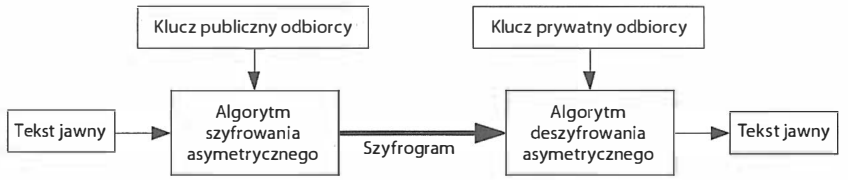
18.4.1. Systemy kryptograficzne

Mimo że kryptografia od wielu lat służy przede wszystkim zachowaniu poufności informacji, zapewnia również zachowanie integralności i autentyczności danych, w czym pomocne są techniki kryptograficzne oraz związane z nimi operacje matematyczne. Aby zrozumieć ich podstawy, warto przeanalizować rysunek 18.2, na którym przedstawiono dwa najważniejsze rodzaje algorytmów kryptograficznych — szyfrowanie z wykorzystaniem **klucza symetrycznego** oraz **klucza publicznego** (**asymetrycznego**).

Na rysunku zostały zaprezentowane ogólne operacje związane z działaniem mechanizmów szyfrowania symetrycznego i asymetrycznego. W obu przypadkach algorytm szyfrowania przetwarza wiadomość zapisaną w formie **jawnego tekstu** i generuje **szyfrogram** (tekst zaszyfrowany). Wykorzystywany w tej operacji **klucz** jest sekwencją bitową sterującą pracą **algorytmu szyfrowania**. Zastosowanie różnych kluczy w odniesieniu do jednakowych danych wejściowych spowodowałoby utworzenie różnych danych wyjściowych. Połączenie algorytmów szyfrowania z protokołami i technikami przetwarzania danych stanowi **system kryptograficzny** (*cryptosystem*). W **symetrycznym systemie kryptograficznym** klucze przeznaczone do szyfrowania i deszyfrowania



System szyfrowania symetrycznego



System szyfrowania asymetrycznego

Rysunek 18.2. *Niezaszyfrowany komunika (tekst jawny) jest przetwarzany przez algorytm szyfrowania w celu wygenerowania zaszyfrowanej wersji komunikatu (szyfrogramu). W systemie szyfrowania symetrycznego do szyfrowania i deszyfrowania jest wykorzystywany ten sam (tajny) klucz. W systemie szyfrowania asymetrycznego poufność informacji zapewnia użycie klucza publicznego do zaszyfrowania wiadomości oraz klucza prywatnego (tajnego) do jej rozszyfrowania*

są identyczne, podobnie jak algorytmy szyfrowania i deszyfrowania. W **asymetrycznym systemie kryptograficznym** każda strona komunikacji jest wyposażana w **parę** kluczy składającą się z jednego klucza publicznego i jednego klucza prywatnego. Klucz publiczny jest udostępniany każdemu podmiotowi, który zamierza przesłać zaszyfrowaną wiadomość do posiadacza klucza prywatnego. Klucz publiczny i prywatny są ze sobą matematycznie powiązane i stanowią wynik działania algorytmu **generowania kluczy**. Jedną z najważniejszych zalet systemów szyfrowania asymetrycznego jest to, że nie trzeba dostarczać tajnego klucza do wszystkich stacji zamierzających rozpocząć komunikację.

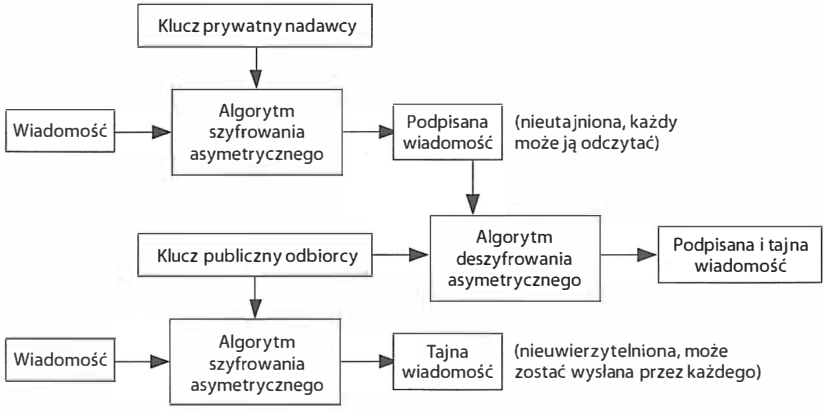
Bez znajomości klucza symetrycznego (w symetrycznym systemie kryptograficznym) lub klucza prywatnego (w asymetrycznym systemie kryptograficznym) osoba przechwytująca szyfrogram nie może odtworzyć tekstu jawnego (przynajmniej taką należy mieć nadzieję). Jest to podstawowy mechanizm zachowania poufności. W przypadku systemu symetrycznego pełni on również funkcję uwierzytelniającą, ponieważ jedynie podmiot posiadający klucz ma możliwość zaszyfrowania tekstu jawnego w taki sposób, aby po rozszyfrowaniu komunikat miał sensowną treść. Odbiorca może rozszyfrować szyfrogram, wyodrębnić z wynikowego zbioru danych fragment przechowujący ustaloną wcześniej wartość i uznać na tej podstawie, że nadawca dysponuje właściwym kluczem, a co się z tym wiąże, jest tym podmiotem, za który się podaje. Większość algorytmów szyfrujących działa w taki sposób, że zmodyfikowanie komunikatu w trakcie przesyłania uniemożliwia wygenerowanie użytecznego teksu jawnego w procesie deszyfracji. Symetryczne systemy kryptograficzne obejmują więc także mechanizmy uwierzytelniania wiadomości oraz weryfikacji ich integralności. Niestety, rozwiązania te nie są dostatecznie

wiarygodne. Z tego względu towarzyszą im zazwyczaj sumy kontrolne, które dają właściwą gwarancję integralności danych. Zagadnienie to zostało omówione w dalszej części rozdziału, za wprowadzeniem do kryptografii.

Algorytmy szyfrowania symetrycznego są zazwyczaj klasyfikowane jako **szyfry blokowe** lub **szyfry strumieniowe**. Szyfry blokowe wykonują operację na ustalonej liczbie bitów jednocześnie (np. na 64 lub 128 bitach danych). Szyfry strumieniowe działają nieustannie niezależnie od tego, ile bitów (lub bajtów) zostanie dostarczonych na wejście modułu szyfrowania. Przez wiele lat najczęściej stosowanym algorytmem szyfrowania był **standard szyfrowania danych** (DES — *Data Encryption Standard*), czyli algorytm blokowy wykorzystujący 64-bitowe porcje danych oraz 56-bitowe klucze. Od pewnego czasu wykorzystanie 56-bitowych kluczy jest uznawane za niewystarczające, przez co w wielu aplikacjach zaczęto stosować **potrójny algorytm DES** (oznaczany jako 3DES lub TDES; zakładający trzykrotne użycie algorytmu DES z dwoma lub trzema różnymi kluczami dla każdego bloku danych). Obecnie mechanizmy DES i 3DES są zastępowane przez **zaawansowany sztandard szyfrowania** (AES — *Advanced Encryption Standard*) [FIPS197], znany również pod pierwotną nazwą **algorytmu Rijandael** (wym. „rajn-dal") powstałą od nazwisk pochodzących z Belgii twórców algorytmu — Vincenta Rijmena i Joan Daemena. Różne warianty mechanizmu AES wykorzystują klucze szyfrowania o długości 128, 192 i 256 bitów, która jest odzwierciedlona w rozszerzeniu nazwy (np. AES-128, AES-192 lub AES-256).

Systemy szyfrowania asymetrycznego mają kilka dodatkowych interesujących cech, niedostępnych w systemach symetrycznych. Załóżmy, że Alicja jest nadawcą wiadomości, a Bob odbiorcą. Przyjmijmy również, że dowolne osoby postronne znają klucz publiczny Boba i mogą do niego przesyłać zaszyfrowane komunikaty. Jedynie Bob będzie mógł rozszyfrować przekazywane informacje, ponieważ tylko on zna klucz prywatny towarzyszący danemu kluczowi publicznemu. Bob nie ma jednak żadnej pewności, że otrzymana wiadomość jest autentyczna, gdyż **dowolny** podmiot może utworzyć i wysłać komunikat, szyfrując go uprzednio kluczem publicznym Boba. Na szczęście, systemy kryptograficzne klucza publicznego pozwalają na użycie dodatkowej funkcji zapewniającej uwierzytelnienie nadawcy. W omawianym przykładzie Alicja może zaszyfrować wiadomość za pomocą własnego klucza prywatnego i przesłać ją do Boba (lub kogokolwiek innego). Wykorzystując znany powszechnie klucz publiczny Alicji, każdy może sprawdzić, czy komunikat został przez nią utworzony oraz czy nie został zmodyfikowany w czasie transmisji. Rozwiązanie to nie gwarantuje jednak poufności, ponieważ każdy ma dostęp do klucza publicznego Alicji. Dlatego Alicja powinna zaszyfrować wiadomość za pomocą własnego klucza prywatnego, a następnie przy użyciu klucza publicznego należącego do Boba, zapewniając autentyczność, integralność **i** poufność informacji. Można autorytatywnie stwierdzić, że przesłany komunikat został utworzony przez Alicję przy zachowaniu poufności wymaganej przez Boba. Proces ten został zilustrowany na rysunku 18.3.

Wykorzystanie szyfrowania asymetrycznego w opisany sposób skutkuje utworzeniem **podpisu cyfrowego**. Podpisy cyfrowe są ważnymi elementami szyfrowania z użyciem klucza publicznego, gdyż gwarantują autentyczność i niezaprzeczalność transmisji. Jedynie podmiot posiadający klucz prywatny Alicji może utworzyć wiadomość lub przeprowadzić transakcję w imieniu Alicji.



System szyfrowania asymetrycznego

Rysunek 18.3. *Symetryczny system kryptograficzny można wykorzystać do zapewnienia poufności (szyfrowanie) i (lub) autentyczności (podpisy cyfrowe) informacji. Jeśli wymagane są obie cechy, wynikiem działania systemu jest podpisany komunikat znany jedynie nadawcy i odbiorcy. Klucze publiczne (zgodnie z nazwą) nie są utajniane*

W **hybrydowych** systemach kryptograficznych wykorzystywane są obie formy kryptografii. Mechanizmy bazujące na kluczach publicznych są w nich stosowane do wymiany losowo generowanych, tajnych **kluczy sesji**, które są następnie używane do szyfrowania symetrycznego w ramach pojedynczej transakcji. Powodem budowania tego rodzaju systemów jest chęć zwiększenia wydajności przetwarzania danych — operacje z użyciem klucza symetrycznego wymagają znacznie mniejszych mocy obliczeniowych niż przetwarzanie danych zaszyfrowanych za pomocą klucza publicznego. Z tego powodu większość działających obecnie systemów ma charakter rozwiązań hybrydowych — szyfrowanie asymetryczne służy do wymiany kluczy szyfrowania symetrycznego, odpowiadającego za szyfrowanie informacji w ramach sesji.

18.4.2. Szyfrowanie RSA — Rivest, Shamir i Adleman

W poprzednim punkcie zostały opisane zasady wykorzystywania mechanizmów szyfrowania asymetrycznego do zapewnienia poufności danych oraz generowania podpisów cyfrowych. W praktyce najczęściej stosowanym algorytmem tego typu jest algorytm RSA, nazwany tak od nazwisk jego autorów — Rivesta, Shamira i Adlemana [RSA78]. Bezpieczeństwo systemu bazuje na założeniu, że podzielenie dużej liczby na składające się na nią liczby pierwsze jest bardzo czasochłonne.

Aby zainicjować działania algorytmu RSA, generowane są dwie duże liczby pierwsze p i q. Zazwyczaj oznacza to konieczność sprawdzenia pewnej liczby dużych wartości nieparzystych, które są losowo wyznaczane do czasu uzyskania dwóch liczb pierwszych. Iloczyn wygenerowanych liczb pierwszych $n = pq$ jest nazywany **modułem**. Długość wartości n, p oraz q jest mierzona w bitach. Najczęściej stosowane są rozwiązania, w których n ma długość 1024 bitów, a pozostałe liczby 512 bitów, choć obecnie zaleca się stosowanie

kluczy o długości co najmniej 2048 bitów. W teorii liczb znana jest **funkcja Eulera** (tocjent) $\Phi(v)$, która operuje na liczbach całkowitych. Jej wynikiem jest liczba liczb całkowitych mniejszych od v, które są względnie pierwsze w odniesieniu do liczby v (tj. są wartościami, których największy wspólny podzielnik wynosi 1). Z uwagi na sposób wyznaczania liczby n w algorytmie RSA $\Phi(n) = (q-1)(p-1)$.

Wykorzystując definicję $\Phi(n)$, możemy wybrać element publiczny (określany jako e od słowa *encryption* oznaczającego szyfrowanie) oraz wyliczyć wartość elementu prywatnego (oznaczanego jako d od słowa *decription* oznaczającego deszyfrowanie). Druga z wymienionych wartości jest wyznaczana ze wzoru $d = e-1 (\text{mod } \Phi(n))$. W praktyce w celu przyspieszenia obliczeń e jest wybierana z niewielkiego zbioru (jest wartością o małej liczbie jedynek logicznych). Może mieć np. wartość 65 537 (10000000000000001 w zapisie binarnym). Aby utworzyć szyfrogram c z jawnego komunikatu m, wyliczana jest wartość $c = m^e (\text{mod } n)$. Z kolei utworzenie wartości m na podstawie c (rozszyfrowanie) sprowadza się do obliczenia wyrażenia $m = c^d (\text{mod } n)$. Klucz publiczny mechanizmu RSA składa się elementu publicznego e oraz modułu n. Natomiast klucz prywatny jest parą obejmującą element prywatny d oraz moduł n.

Zgodnie z wcześniejszymi informacjami algorytmy klucza publicznego, takie jak RSA, znajdują zastosowanie również w procedurach generowania podpisów cyfrowych, które sprowadzają się do wykonania „odwrotnej" operacji RSA. Aby utworzyć podpis RSA wiadomości m, wyznaczana jest wartość $s = m^d (\text{mod } n)$ odpowiadająca podpisanej wersji wiadomości m. Każdy odbiorca wartości s może użyć publicznego elementu e do wygenerowania $m = s^e (\text{mod } n)$. Operacja ta pozwala na sprawdzenie, czy podmiot, który utworzył wartość s, dysponował prywatną wartością d (jeśli nie, odtworzona wartość m nie ma sensu).

Bezpieczeństwo zapewniane przez mechanizm RSA bazuje na założeniu, że podział dużych wartości na odpowiednie elementy składowe jest bardzo czasochłonny. W analizowanym wcześniej przykładzie (przedstawionym na rysunku 18.1) oznaczałoby to, że Ewa może ustalić wartości n i e, ale nie zna liczb p, q oraz $\Phi(n)$. Gdyby mogła wyznaczyć którąkolwiek z trzech pozostałych wartości, mogłaby w łatwy sposób wyliczyć d, wykorzystując przedstawione wcześniej zależności. Jednak wykonanie zadania wymaga rozkładu wartości n na czynniki pierwsze, a poddanie takiej operacji liczb o 1000 bitów (lub dłuższych) wydaje się obecnie leżeć poza możliwościami nawet najlepszych algorytmów faktoryzacji. Najtrudniejsza wydaje się faktoryzacja liczb półpierwszych (liczb, które są iloczynem dwóch liczb pierwszych).

18.4.3. Metoda uzgadniania kluczy Diffie-Hellman-Merkle (znana również jako algorytm Diffiego-Hellmana lub DH)

Działanie protokołów zabezpieczeń często wymaga od stron komunikacji uzgodnienia wspólnego ciągu bitowego, służącego jako klucz szyfrowania symetrycznego. Przeprowadzenie takiej operacji w sieci, w której pracują urządzenia podsłuchowe (np. komputer Ewy) jest sporym wyzwaniem. Uzgodnienie wspólnej tajnej liczby przez dwie strony komunikacji (Alicję i Boba) w taki sposób, żeby Ewa się o niej nie dowiedziała, nie jest zadaniem trywialnym. Jednym z rozwiązań jest użycie **protokołu uzgadniania kluczy Diffiego-Hellmana-Merkle** (nazywanego częściej protokołem Diffiego-Hellmana

lub DH) bazującego na arytmetyce ciał skończonych [DH76][1]. Mechanizm DH jest wykorzystywany w wielu protokołach zabezpieczających wymianę danych w Internecie [RFC2631] i jest blisko związany z algorytmem RSA odpowiedzialnym za szyfrowanie z użyciem klucza publicznego. Przeanalizujmy więc jego działanie.

Wykorzystajmy ten sam przykładowy zbiór użytkowników (Alicję, Boba itd.). Załóżmy, że wszystkie osoby znają dwie liczby całkowite p i g. Niech p będzie (dużą) liczbą pierwszą, a g (takie, że $g<p$) pierwiastkiem pierwotnym modulo p. Wówczas wszystkie liczby całkowite z grupy $Z_p = \{1, \ldots, p-1\}$ można utworzyć, podnosząc g do pewnej potęgi. Innymi słowy, dla każdego n istnieje wartość k taka, że $g^k \equiv n \pmod p$. Wyznaczenie wartości k przy danych g, n i p (nazywane **problemem logarytmu dyskretnego**) okazuje się bardzo trudne, co z kolei pozwala stwierdzić, że mechanizm DH jest bezpieczny. Określenie wartości n przy danych g, k oraz p nie nastręcza trudności. Dzięki temu algorytm jest niezwykle praktyczny.

Uzgodnienie klucza między Alicją i Bobem sprowadza się wykonania następującej procedury: Alicja wybiera pewną losową wartość a, następnie oblicza wartość $A = g^a \pmod p$ i przesyła ją do Boba. Bob w sposób losowy wybiera wartość b i oblicza wartość $B = g^b \pmod p$, po czym przesyła ją do Alicji. W końcu Alicja i Bob wyznaczają tę samą tajną wartość $K = g^{ab} \pmod p$. Alicja oblicza ją w następujący sposób:

$$K = B^a \pmod p = g^{ba} \pmod p$$

Natomiast Bob wykonuje obliczenie:

$$K = A^b \pmod p = g^{ab} \pmod p$$

Ponieważ g^{ba} jest równe g^{ab} (zbiór Z_p charakteryzuje się łącznością wykładników i wszystkie strony znają wykorzystywaną grupę Z_p), zarówno Alicja, jak i Bob znają wartość K. Ewa ma dostęp jedynie do wartości g, p, A oraz B. Nie może więc obliczyć parametru K bez rozwiązania problemu logarytmu dyskretnego [MW99]. Protokół ten jest jednak podatny na atak ze strony Marioli. Mariola może udawać Boba podczas komunikacji z Alicją oraz Alicję w czasie komunikacji z Bobem, podstawiając własne wartości A i B. Nic jednak nie stoi na przeszkodzie, aby podstawowy protokół DH został uzupełniony o ochronę przed atakami typu człowiek pośrodku przez uwierzytelnienie publicznych parametrów A i B [DOW92]. W typowym rozwiązaniu, jakim jest **protokół stacja-stacja** (STS — *Station-to-Station*), Alicja i Bob mają możliwość podpisania generowanych wartości publicznych.

18.4.4. Szyfrowanie z uwierzytelnieniem i kryptografia krzywych eliptycznych (ECC)

Podczas stosowania algorytmu RSA zwiększenie poziomu bezpieczeństwa wynika z wykorzystania wielkich liczb. Jednak stosowane w mechanizmie obliczenia matematyczne (np. potęgowanie) mogą się okazać czasochłonne, szczególnie w przypadku zwiększania liczb. Aby zmniejszyć obciążenie systemów odpowiedzialnych za szyfrowanie

[1] Technika ta została opisana i sklasyfikowana w 1973 roku przez Clifforda Cockesa w dokumencie „A Note on »Non-Secret Encryption«" (patrz *http://www.fi.muni.cz/usr/matyas/lecture/paper2.pdf*).

i generowanie podpisów cyfrowych (w celu zapewniania poufności transmisji), opracowano nową klasę algorytmów nazywanych mechanizmami **szyfrowania z uwierzytelnieniem** (*authenticated encryption* lub *signcryption*) [Z97], które zapewniają wykonanie zadania mniejszym kosztem niż w przypadku rozdzielenia operacji szyfrowania i uwierzytelnienia. Jeszcze większą wydajność algorytmu można niekiedy uzyskać, zmieniając matematyczne podstawy działania mechanizmu klucza publicznego.

Podczas poszukiwania rozwiązań gwarantujących bezpieczeństwo transmisji bez utraty wydajności przetwarzania danych naukowcy opracowali systemy szyfrowania asymetrycznego inne niż RSA. Jedna z alternatywnych technik bazuje na dużej złożoności obliczeniowej operacji obliczania algorytmu dyskretnego i stanowi fundament **kryptografii krzywych eliptycznych** (ECC — *Elliptic Curve Cryptography*; nie należy mylić z kodem korekcji błędów, który również jest oznaczany skrótem ECC) [M85] [K87] [RFC5753]. Systemy ECC zapewniają bezpieczeństwo porównywalne z oferowanym przez mechanizm RSA, ale przy wykorzystaniu znacznie krótszych kluczy (np. sześciokrotnie krótszych w porównaniu z 1024-bitowym modułem RSA). Implementacja algorytmu jest więc znacznie łatwiejsza i zapewnia szybsze działanie wynikowego systemu. Rozwiązania ECC zostały opisane standardami i są gotowe do użycia w wielu aplikacjach, które były dotychczas zdominowane przez mechanizmy RSA. Jednak z uwagi na prawa patentowe (należące do firmy Certicom Corporation) masowe wdrożenie algorytmu jest dość powolne (algorytm RSA również został opatentowany, ale ochrona patentowa zakończyła się w roku 2000).

18.4.5. Wyznaczanie kluczy i doskonała poufność przekazu (PFS)

W systemach komunikacyjnych, w których wymienianych jest wiele wiadomości, stosuje się zazwyczaj krótkookresowe klucze sesji odpowiedzialne za szyfrowanie symetryczne. Klucz sesji jest najczęściej wartością losową (więcej informacji na ten temat znajduje się w kolejnym punkcie) generowaną przez **funkcję wyznaczania kluczy** (KDF — *Key Derivation Function*) na podstawie pewnych danych wejściowych (na podstawie klucza głównego lub wcześniejszych kluczy sesji). Jeśli klucz sesji zostanie przechwycony przez osobę atakującą, wszystkie zaszyfrowane za jego pomocą dane mogą zostać ujawnione. Z tego względu często stosuje się technikę okresowej zmiany klucza szyfrowania w trakcie dłuższej sesji komunikacyjnej. Rozwiązanie gwarantujące poufność informacji mimo ujawnienia jednego klucza sesji jest określane jako rozwiązanie o **doskonałej poufności przekazu** (PFS — *Perfect Forward Secrecy*). Mechanizmy z grupy PFS często wymagają dodatkowej wymiany kluczy lub weryfikacji kluczy, co wprowadza dodatkowy narzut transmisyjny. Przykładem jest protokół STS rozszerzający opisany wcześniej standard DH.

18.4.6. Liczby pseudolosowe, generatory i rodziny funkcji

Liczby losowe służą przede wszystkim jako wartości inicjujące działanie funkcji kryptograficznych lub wartości bazowe w procesie generowania trudnych do odgadnięcia haseł. Ponieważ jednak sposób działania komputerów z założenia nie jest losowy, wyznaczenie tego typu liczb jest dość skomplikowane. Wartości generowane przez komputery w celu zasymulowania zdarzeń losowych są nazywane **liczbami pseudolosowymi**.

Nie są one prawdziwie losowymi wartościami, ale spełniają wiele statystycznych założeń, które pozwalają na uznanie ich za takowe (np. ich rozkład jest równomierny w określonym przedziale).

Za generowanie wartości pseudolosowych odpowiadają algorytmy oraz urządzenia nazywane **generatorami liczb pseudolosowych** (PRNG — *PseudoRandom Number Generator*) lub **generatorami pseudolosowymi** (PRG — *PseudoRandom Generator*). Proste mechanizmy PRNG mają charakter deterministyczny. Oznacza to, że cechują się niewielką liczbą stanów wewnętrznych zależnych od **wartości inicjującej** (*seed*). Ustalenie stanu początkowego pozwala na wyznaczenie wartości pseudolosowych. Przykładowo algorytm **liniowego generatora kongruencyjnego** (LCG — *Linear Congruential Generator*) generuje wartości sprawiające wrażenie losowych, które można precyzyjnie przewidywać, znając parametry początkowe. Mechanizmy LCG doskonale nadają się do użycia w niektórych rodzajach programów komputerowych (np. do wywoływania losowych zdarzeń w grach), ale nie są odpowiednie do stosowania w funkcjach kryptograficznych.

Rodzina funkcji pseudolosowych (PRF — *PseudoRandom Function family*) jest rodziną funkcji, które trudno rozróżnić w sposób algorytmiczny (za pomocą algorytmów o wielomianowej złożoności czasowej) od prawdziwie losowych funkcji [GGM86]. Rozwiązania PRF są mechanizmami silniejszymi niż PRG, ponieważ funkcje PRG można utworzyć na bazie PRF. Algorytmy PRF są podstawą **kryptograficznie silnych** (bezpiecznych) generatorów liczb pseudolosowych (CSPRNG — *Cryptographically Strong PRNG*). Mechanizmy CSPRNG są wykorzystywane w aplikacjach kryptograficznych m.in. do generowania kluczy sesji o dostatecznie dużym zakresie losowości [RFC4086].

18.4.7. Wartości jednorazowe i zaburzające

Wartości jednorazowe (*nonce*) są liczbami używanymi do realizacji jednej transakcji w protokole kryptograficznym. Zazwyczaj są wyznaczane w sposób losowy lub pseudolosowy i służą do uwierzytelniania stron komunikacji (gwarantując odpowiednią niepowtarzalność wymienianych danych). Niepowtarzalność danych jest szczególnie istotna, jeśli dana operacja jest wykonywana okresowo. Przykładowo protokoły typu **hasło-odzew** (*challenge-response*) umożliwiają serwerom przesłanie wartości jednorazowej do stacji klienckiej, która próbuje ustanowić połączenie. Oprogramowanie klienckie musi wówczas odesłać dane uwierzytelniające oraz odebraną wartość jednorazową (zazwyczaj w formie zaszyfrowanej) w określonym czasie. Takie postępowanie zapobiega atakom z odtworzeniem pakietów, ponieważ powtórnie dostarczone komunikaty uwierzytelniające zawierają niewłaściwe wartości jednorazowe.

Wartość **zaburzająca** (*salt*) jest liczbą losową lub pseudolosową, która służy do zabezpieczenia systemu przed atakami realizowanymi **metodą siłową** (*brute-force*) w odniesieniu do haseł. Ataki siłowe polegają zazwyczaj na zgadywaniu haseł, ciągów zabezpieczających, kluczy lub podobnych tajnych wartości i sprawdzaniu, czy pasują do atakowanego konta. Wartości zaburzające utrudniają wykonanie porównania. Znajdują one zastosowanie m.in. w systemie UNIX, który przechowuje hasła użytkownika w pliku dostępnym dla wszystkich osób korzystających z systemu. Podczas logowania

użytkownik podaje hasło, które zostaje wykorzystane do dwukrotnego zaszyfrowania ustalonej wartości. Wynik operacji jest następnie porównywany z odpowiednim wpisem w pliku haseł. Zgodność wartości oznacza, że podano poprawne hasło.

Gdy szyfrowanie z zastosowaniem techniki DES stało się powszechnie znane, pojawiło się zagrożenie **ataku słownikowego** (*dictionary attack*) z użyciem wpisów słownikowych zaszyfrowanych wcześniej za pomocą algorytmu DES (tworzących tęczowe tablice) i porównywanych z elementami pliku haseł. Rozwiązaniem problemu okazało się dodanie pseudolosowej 12-bitowej wartości zaburzającej, która zmieniała sposób działania mechanizmu DES na jeden z 4096 (niestandardowych) sposobów. Po pewnym czasie uznano jednak, że z powodu zwiększającej się mocy obliczeniowej komputerów (pozwalającej na wyznaczanie większej liczby wartości w tym samym czasie) 12-bitowa wartość jest niewystarczająca i trzeba ją wydłużyć.

18.4.8. Kryptograficzne funkcje skrótu

Większość opisanych w książce protokołów (w tym Ethernet, IP, ICMP, UDP i TCP) uwzględnia sekwencje kontrolne (wartości FCS, sumy kontrolne, wartości CRC) w celu sprawdzenia, czy podczas transmisji danej jednostki PDU nie wystąpiły błędy bitowe. Wykorzystane do tego celu funkcje matematyczne zapewniają odpowiedni kompromis między prawdopodobieństwem wykrycia błędu oraz wielkością narzutu transmisyjnego wynikającą z konieczności przesłania wartości FCS. W przypadku zabezpieczenia przekazu istotna jest nie tylko odporność na losowe (nieliczne) przekłamania, ale również na celowe modyfikowanie komunikatów. Trzeba pamiętać, że użytkownik, taki jak Mariola, może zmienić treść wiadomości w czasie przesyłania jej przez sieć. Klasyczne funkcje FCS nie zagwarantują odpowiedniej ochrony.

Suma kontrolna (wartość FCS) może być stosowana do sprawdzenia integralności komunikatu (i zabezpieczenia wymiany danych przed działalnością Marioli), jeśli jest w odpowiedni sposób wygenerowana. Funkcje odpowiadające za wyznaczanie wspomnianych wartości nazywają się **kryptograficznymi funkcjami skrótu** i często wchodzą w skład algorytmów szyfrowania. Wynikiem działania funkcji skrótu H w odniesieniu do komunikatu M jest wartość nazywana **skrótem** lub **odciskiem** wiadomości $H(M)$. Wartość skrótu jest silną wartością FCS, którą można wyliczyć w łatwy sposób. Oto jej cechy.

- **Odporność na wyznaczenie przeciwobrazu** — wyznaczenie wartości M ze znanej wartości $H(M)$ powinno być bardzo trudne.
- **Odporność na wyznaczenie drugiego przeciwobrazu** — przy znanej wartości $H(M1)$ wyznaczenie wartości $M2 \neq M1$ takiej, że $H(M1) = H(M2)$, powinno być bardzo trudne.
- **Odporność na kolizje** — znalezienie takiej pary $M1$, $M2$, że $H(M1) = H(M2)$ przy $M2 \neq M1$, powinno być bardzo trudne.

Jeśli funkcja skrótu ma wszystkie wymienione właściwości, to dwie wiadomości o tym samej wartości skrótu kryptograficznego mają najprawdopodobniej tą samą treść. Dwa najczęściej wykorzystywane algorytmy kryptograficzne to *Message Digest Algorithm 5* (MD5 [RFC1321]), który generuje 128-bitowe (16-bajtowe) wartości, oraz *Secure Hash Algorithm 1* (SHA-1) zwracający wartości 160-bitowe (20-bajtowe). Najnowsza rodzina

funkcji skrótu o nazwie SHA-2 generuje odciski o długości 224, 256, 384 oraz 512 bitów (odpowiednio 28, 32, 48 i 64 bajty). Trwają prace nad innymi mechanizmami.

Funkcje skrótu bazują często na **funkcjach kompresji** *f*, które pobierają dane wejściowe o długości *L* i zwracają odporny na kolizje, ale deterministyczny wynik o rozmiarze mniejszym niż *L*. Jednym z mechanizmów zdolnych do generowania skrótów odpornych na kolizje jest **konstrukcja Merkle-Damgírda**, która pobiera dane wejściowe o dowolnej długości, dzieli je na bloki o rozmiarze *L*, dopełnia, przekazuje do funkcji *f* i sumuje wyniki.

Funkcja MD5 była powszechnie wykorzystywana w rozwiązania internetowych do czasu jej złamania w 2005 roku (wykazano wówczas, że dwie 128-bajtowe sekwencje mogą doprowadzić do wygenerowania jednakowych wartości MD5) [WY05]. Alternatywą okazała się funkcja SHA-1, ale również ona miała pewne słabości. Dlatego opracowano rodzinę algorytmów SHA-2. Choć podobieństwo rozwiązań SHA-2 do SHA-1 uzasadnia obawy, że również te mechanizmy mogą mieć luki. W grudniu 2010 roku Narodowy Instytut Standaryzacji i Technologii (NIST — *National Institute of Standards and Technology*) w Stanach Zjednoczonych ogłosił, że wybrano pięć algorytmów jako rozwiązania rozważane podczas tworzenia nowego algorytmu skrótu oznaczonego jako SHA-3 [CHP]. Ostatecznie w roku 2012 wybrano mechanizm Keccak.

18.4.9. Kody uwierzytelniania wiadomości (MAC, HMAC, CMAC i GMAC)

Kod uwierzytelniający wiadomość (MAC — *Message Authentication Code*; niezwiązany niczym oprócz skrótu z opisanym w rozdziale 3. adresem MAC warstwy łącza danych) pozwala na zapewnienie integralności i autentyczności komunikatu. Wartości MAC są generowane zazwyczaj na podstawie funkcji skrótu wykorzystujących **klucz**. Niektóre funkcje działają na takiej samej zasadzie jak algorytmy wyznaczania skrótu wiadomości (opisane w punkcie 18.4.8), ale w procesie weryfikacji integralności komunikatu wymagają uwzględnienia tajnego klucza. Mogą być również stosowane do potwierdzenia tożsamości (do uwierzytelnienia) nadawcy.

Od kodów MAC wymaga się odporności na różnego rodzaju **fałszerstwa**. Odporność na **selektywne fałszerstwo** oznacza, że dla danej funkcji skrótu $H(M, K)$ pobierającej komunikat wejściowy M oraz klucz K trudno znaleźć wartość wynikową bez znajomości K, mimo dysponowania wartością M. Funkcja $H(M, K)$ jest z kolei odporna na **fałszerstwa egzystencjalne**, jeśli osoba atakująca, nie dysponując kluczem K, nie może ustalić żadnej nieznanej wcześniej kombinacji M i $H(M, K)$. Kody MAC nie realizują tych samych zadań, które są charakterystyczne dla podpisów cyfrowych. Nie zapewniają np. niezaprzeczalności, ponieważ tajny klucz jest znany więcej niż jednej stronie komunikacji.

Standardowy kod MAC wykorzystujący w określony sposób kryptograficzną funkcję skrótu jest nazywany **kodem uwierzytelniającym wiadomość zabezpieczonym kluczem** (HMAC — *keyed-Hash Message Authentication Code*) [FIPS198][RFC2104]. Algorytm HMAC bazuje na podstawowych algorytmach generowania skrótów kryptograficznych — $H(M)$. Aby utworzyć t-bajtową wartość HMAC z wiadomości M na podstawie klucza K i przy zastosowaniu funkcji H (określanej jako *HMAC-H*), wykonywana jest operacja opisana za pomocą poniższej definicji:

$$HMAC\text{-}H(K, M)^t = \Lambda_t(H((K \oplus opad) \parallel H((K \oplus ipad) \parallel M)))$$

W powyższym wyrażeniu występują parametry: *opad* (zewnętrzne dopełnienie) — parametr ten jest tablicą zawierającą wartości 0x5C powtórzone $|K|$ razy — oraz *ipad* (wewnętrzne dopełnienie) będący tablicą składającą się z $|K|$ wartości 0x36. Znak $\oplus$ symbolizuje wektorową operację XOR, natomiast znak $\|$ odpowiada operacji konkatenacji. Standardowy wynik funkcji HMAC składa się z określonej liczby (t) bajtów, operator $\Lambda_t(M)$ służy więc do wyodrębnienia t najbardziej znaczących bajtów z M.

Uważny czytelnik zauważy, że kod HMAC jest skrótem z innego skrótu o ogólnym wzorze $H(K1 \| H(K2 \| M))$ uwzględniającym klucze $K1$ i $K2$. Taka struktura zapewnia odporność na **ataki z wydłużeniem wiadomości** (*extension attack*), w których wybrana wartość dopełnienia może zostać połączona (np. przez Mariolę) z przechwyconą wiadomością i wartością skrótu do sformowania nowej wiadomości i wartości skrótu (która nie została wysłana przez Alicję). Wartości *ipad* i *opad* nie są szczególnie ważne, ale mają tendencję generowania wartości $K1$ i $K2$ o kilku bitach wspólnych (tj. o dużej *odległości Hamminga*). Niektóre ataki tego typu wydają się skuteczne w odniesieniu do niedbale przygotowanych kodów MAC (takich, jakie wynikają ze zależności $H(K \| M)$ lub $H(M \| K)$) oraz nieskuteczne w odniesieniu do konstrukcji HMAC (lub NMAC [BCK96], które wywodzą się z rozwiązań HMAC)[B06].

Ostatnio standaryzacji poddano inną formę kodów MAC oznaczanych jako CMAC [FIPS800-38B] oraz GMAC [NIST800-38D]. W ich tworzeniu zamiast kryptograficznych funkcji skrótu (takich jak HMAC) wykorzystuje się szyfry blokowe, takie jak AES lub 3DES. Algorytm CMAC jest przewidziany do stosowania w środowiskach, w których łatwiejsze jest użycie szyfrów blokowych niż funkcji skrótu. Szczegółowe informacje na temat mechanizmu CMAC bazującego na standardzie AES-128 (o nazwie AES-CMAC) znajdują się w opracowaniu [RFC4493]. W ogólnym ujęciu działanie rozwiązania polega na szyfrowaniu bloku komunikatu za pomocą algorytmu AES-128 z użyciem klucza K, wykonaniu operacji XOR z wynikiem szyfrowania kolejnego bloku, zaszyfrowania wyniku i ponowienia całego zadania w odniesieniu do kolejnego bloku, aż do wyczerpania wiadomości. Wartość końcowa odpowiada wynikowi ostatniej operacji szyfrowania. Jeśli długość ostatniego bloku wiadomości jest parzystą wielokrotnością rozmiaru bloku algorytmu, końcowe szyfrowanie jest wykonywane za pomocą specjalnego **podklucza** (*subkey*), wyznaczonego przez odrębny algorytm [IK03] z klucza K. W przeciwnym przypadku ostatni blok wiadomości zostaje dopełniony i zaszyfrowany z użyciem drugiego podklucza wygenerowanego z klucza K. Kody GMAC są generowane przez specjalną wersję algorytmu AES nazywaną *Galois/Counter Mode* (GCM). Wykorzystują również zabezpieczoną kluczem funkcję skrótu (o nazwie GHASH, która nie jest funkcją kryptograficzną). Więcej informacji na temat trybów pracy systemów kryptograficznych znajduje się w następnym punkcie.

18.4.10. Zestawy algorytmów kryptograficznych

Dotychczas rozważaliśmy działanie mechanizmów, które zapewniają poufność, autentyczność i integralność informacji przesyłanych przez niezabezpieczoną sieć. Jak wiadomo, istnieją również inne korzyści (np. niezaprzeczalność), które można uzyskać, stosując odpowiednie techniki matematyczne lub kryptograficzne. Kombinację konkretnych rozwiązań użytych w danym systemie (szczególnie tych, które są stosowane w Internecie) nazywamy **zestawem algorytmów kryptograficznych** (*cryptographic suite*) lub ze-

stawem algorytmów szyfrujących (*cipher suite*), choć pierwsza nazwa jest bardziej odpowiednia. Definiuje on nie tylko zasady szyfrowania, ale narzuca również algorytm generowania kodów MAC, mechanizm PRF, sposób uzgadniania klucza, technikę podpisywania wiadomości oraz długości i parametry wykorzystywanych kluczy.

Większość zestawów algorytmów kryptograficznych jest przystosowana do współdziałania z protokołami, które zostaną omówione dalej w tym punkcie. Zazwyczaj wykorzystywany w zestawie algorytm szyfrujący jest określany za pomocą nazwy, liczby bitów kluczy (często odpowiadającej wielokrotności 128 bitów) oraz **trybu** pracy. Wśród zatwierdzonych do stosowania w Internecie mechanizmów szyfrujących znajdują się AES, 3DES, NULL [RFC2410] i CAMELLIA [RFC3713]. Algorytm NULL nie zmienia przetwarzanych danych i znajduje zastosowanie w szczególnych przypadkach, w których nie jest wymagana poufność transmisji.

Tryb pracy mechanizmu szyfrującego, szczególnie szyfrów blokowych, określa zasady użycia funkcji szyfrującej do przetwarzania kolejnych bloków informacji (np. kaskadowo) podczas szyfrowania i deszyfrowania wiadomości z użyciem pojedynczego klucza. Obecnie najczęściej stosuje się tryb **wiązania bloków szyfrogramu** (CBC — *Cipher Block Chaining*) lub tryb **licznikowy** (CTR — *CounTeR*), choć zdefiniowano również wiele innych. Szyfrowanie informacji w trybie CBC polega na zaszyfrowaniu bloku tekstu jawnego, a następnie wykonaniu operacji XOR z wcześniejszym blokiem szyfrogramu (pierwszy blok jest poddawany operacji XOR z losowo wybranym **wektorem inicjującym** (IV — *Initialization Vector*). Praca w trybie CTR wymaga wstępnego wygenerowania wartości będącej połączeniem wektora inicjującego i **licznika**, który jest następnie inkrementowany podczas szyfrowania każdego kolejnego bloku wiadomości. W rezultacie powstaje **strumień klucz** (*keystream*), czyli sekwencja (pozornie losowych) wartości bitowych, które po zsumowaniu modulo (XOR) z bitami tekstu jawnego generują szyfrogram. Zastosowanie opisanego mechanizmu pozwala na przekształcenie szyfru blokowego w szyfr strumieniowy, ponieważ nie jest wymagane dopełnianie danych wejściowych.

Rozwiązania CBC wymuszają szeregowe szyfrowanie wiadomości oraz częściowo szeregowe deszyfrowanie. Z kolei algorytmy CTR pozwalają na szyfrowanie i deszyfrowanie w sposób równoległy (znacznie efektywniejszy). Z tego powodu zyskują coraz większą popularność. Ponadto niektóre odmiany trybu CTR (uwzględniające mechanizmy CBC-MAC (CCM) oraz GCM) zapewniają uwierzytelnianie zaszyfrowanych informacji [RFC4309] oraz uwierzytelnianie (ale nie szyfrowanie) dodatkowych danych (nazywane **uwierzytelnionym szyfrowaniem z danymi towarzyszącymi** [AEAD — *Authenticated Encryption with Associated Data*]) [RFC5116]. Wykorzystanie algorytmu uwierzytelnionego szyfrowania sprawia, że generowanie oddzielnych wartości MAC jest najczęściej zbędne. W pewnym szczególnym zastosowaniu algorytmu AEAD, w którym nie jest wymagana poufność przetwarzanych danych, konieczne okazuje się jednak utworzenie pewnych kodów MAC (np. GMAC). Jeśli w nazwie zestawu algorytmów kryptograficznych wymieniony jest algorytm szyfrowania, tryb pracy i długość klucza najczęściej również wynikają z tej nazwy. Przykładowo nazwa ENCR_AES_CTR odnosi się do standardu AES-128 w trybie CTR.

Jeśli zestaw kryptograficzny obejmuje rodzinę PRF, jest to najczęściej rodzina funkcji skrótów kryptograficznych (np. SHA-2 [RFC6234]) lub kodów MAC (np. CMAC [RFC4434][RFC4615]). Nazwy tego typu mechanizmów zwykle uwzględniają nazwę

funkcji bazowej. Przykładowo algorytm AES-CMAC-PRF-128 definiuje rodzina PRF kodów CMAC bazująca na algorytmie AES-128. Inny sposób zapisu nazwy tego samego rozwiązania to PRF_AES128_CMAC. Z kolei algorytm PRF-HMAC_SHA1 odnosi się do rodziny PRF bazującej na mechanizmie HMAC-SHA1.

Jeśli w definicji wykorzystywanego w Internecie zestawu kryptograficznego znajduje się również odniesienie do techniki uzgadniania kluczy, najczęściej dotyczy ono grupy algorytmu DH, ponieważ obecnie żaden inny protokół uzgadniania kluczy nie jest wykorzystywany na skalę masową. Podczas stosowania mechanizmu DH do generowania kluczy algorytmu szyfrującego trzeba zachować szczególną ostrożność, aby zagwarantować utworzenie wartości o dostatecznej długości (sile), które nie pozwolą na złamanie algorytmu szyfrującego. Dlatego zdefiniowano ponad 16 grup mechanizmu DH przeznaczonych do stosowania w różnych kontekstach [RFC5114]. Pierwsze pięć grup jest znanych jako „grupy Oakley", ponieważ zostały zdefiniowane w protokole Oakley [RFC2409] należącym początkowo do stosu IPsec (uznawanym obecnie za przedawniony). Grupy **potęgowania modularnego** (MODP — *modular exponential*) bazują na operacjach potęgowania i wyznaczania wartości modulo. Grupy ECP [RFC5903] są opisywane za pomocą krzywych nad ciałami Galois $GF(P)$ wyznaczanymi dla liczby pierwszej P. Z kolei grupy krzywych eliptycznych modulo potęgi 2 (EC2N) bazują na krzywych nad ciałami $GF(2^N)$ z określoną wartością N.

Zestaw algorytmów kryptograficznych obejmuje czasami również mechanizmy wyznaczania podpisów cyfrowych, które znajdują zastosowanie w generowaniu odcisków różnych danych, kodów MAC oraz wartości DH. Do tego celu najczęściej wykorzystywany jest algorytm RSA (który służy do podpisywania skrótu określonego bloku danych), choć w niektórych przypadkach uzasadnione okazuje się zastosowanie **standardu podpisu cyfrowego** (DSS — *Digital Signature Standard*, oznaczanego również jako DSA — *Digital Signature Algorithm*) [FIPS186-3]. W wielu systemach dodatkowo dostępne są najnowsze rozwiązania z zakresu podpisów cyfrowych — bazujące na krzywych eliptycznych algorytmy ECC (np. ECDSA [X9.62-2005]).

Koncepcja wydzielania zestawów algorytmów kryptograficznych jest podyktowana rozwojem internetowych protokołów zabezpieczeń i koniecznością zdefiniowania niezależnych modułów funkcjonalnych. Wraz ze wzrostem mocy obliczeniowej komputerów wcześniejsze algorytmy kryptograficzne i określone długości kluczy okazują się podatne na ataki, co z kolei wymusza zastępowanie dotychczasowych metod matematycznych i kryptograficznych nowymi rozwiązaniami, mimo że podstawowe założenia dotyczące działania protokołu pozostają niezmienne. Dzięki temu decyzję o wymianie zestawu algorytmów kryptograficznych można podjąć niezależnie od rodzaju stosowanego protokołu komunikacyjnego na podstawie takich parametrów jak wygoda użycia, wydajność czy stopień zabezpieczenia. Protokoły wykorzystują elementy zestawu kryptograficznego w pewien standardowy sposób, więc nic nie stoi na przeszkodzie, aby zestawy te były zastępowane w dowolnym momencie, gdy okażą się potrzebne. Obecnie podczas projektowania protokołów komunikacyjnych często stosuje się zasadę wydzielenia danych do odrębnego przetwarzania (outsourcingu) w zestawie kryptograficznym opracowywanym przez większą społeczność o odpowiedniej wiedzy z zakresu matematyki i kryptografii. Choć możliwość przyłączania nowych zestawów kryptograficznych wydaje się obiecująca, standaryzacja odpowiednich zestawów może zająć nawet kilka lat. Poprawna wymiana informacji między stronami połączenia wymaga zastoso-

wania jednakowego zestawu kryptograficznego przez oba podmioty. Wymóg ten bywa trudny do spełnienia z uwagi na różny sposób implementowania algorytmów kryptograficznych w różnym oprogramowaniu i w odmiennych systemach sprzętowych.

18.5. Certyfikaty, urzędy certyfikacji (CA) i infrastruktura PKI

Rozwiązania dostarczane przez kryptografię i związaną z nią matematykę (podpisy cyfrowe i algorytmy szyfrowania) stanowią podstawę budowy bezpiecznych systemów komunikacyjnych. Jednak opracowanie w pełni funkcjonalnego systemu wymaga jeszcze sporo dodatkowej pracy. Jego projektanci muszą przygotować bezpieczny protokół, który w odpowiedni sposób wykorzysta dostępne techniki kryptograficzne i będzie zdolny do tworzenia, wymiany i unieważniania kluczy (będzie definiował zasady **zarządzania kluczami**). Zarządzanie kluczami pozostaje jednym z największych wyzwań w procesie wdrażania systemów kryptograficznych na wielką skalę (pomiędzy różnymi domenami administracyjnymi).

Jedną z trudności zauważanych w asymetrycznych systemach kryptograficznych jest wybór właściwego klucza publicznego danej strony komunikacji. W przedstawionym wcześniej przykładzie Alicja mogła przesłać klucz do Boba, ale Mariola miała możliwość zmodyfikowania go w trakcie przesyłania przez sieć i przesłania do Boba własnego klucza publicznego. Bob (będący **jednostką zależną**) mógłby wówczas korzystać z klucza Marioli, sądząc, że należy do Alicji. Dzięki temu Mariola mogłaby skutecznie przesłonić Alicję. Aby wyeliminować opisany problem, wykorzystuje się **certyfikaty kluczy publicznych**. Pozwalają one na przypisanie dowodu tożsamości (podpisu cyfrowego) do określonego klucza publicznego. Początkowo wdrożenie takiego rozwiązania wydaje się tak samo niemożliwe do wykonania, jak ustalenie „co było pierwsze — jajko czy kura". W jaki sposób można podpisać klucz publiczny, skoro podpis cyfrowy sam wymaga wiarygodnego klucza publicznego? Istnieją dwa rozwiązania tego zadania.

Jedno z nich o nazwie **sieć zaufania** (*web of trust*) zakłada istnienie certyfikatu (powiązania dowód tożsamości-klucz) **potwierdzanego** przez pewną liczbę wcześniej działających użytkowników. Osoba potwierdzająca podpisuje i dystrybuuje certyfikat. Im więcej osób podpisze certyfikat (z biegiem czasu), tym bardziej jest on wiarygodny. Jednostka weryfikująca certyfikat może zażądać wykazania zaufania do tego certyfikatu ze strony określonej liczby osób lub podpisania go przez określone osoby. Model sieci zaufania ma charakter zdecentralizowany, podobny do społeczności bez określonej władzy centralnej. Ma to pewne wady i zalety. Brak centralnego urzędu certyfikacji oznacza, że system nie załamie się z powodu awarii jednego elementu. Z drugiej strony, nowy członek społeczności musi się liczyć z tym, że jego certyfikat nie zyska w krótkim czasie dostatecznego poziomu zaufania, aby był akceptowany przez większość użytkowników. W celu przyspieszenia tego procesu część grup organizuje „imprezy z podpisywaniem kluczy" (*key signing party*). Sieć zaufania została po raz pierwszy opisana jako element systemu PGP przeznaczonego do szyfrowania poczty elektronicznej [NAZ00], który rozwinął się w standard kodowania o nazwie **OpenPGP** i jest zdefiniowany w dokumencie [RFC4880].

Bardziej sformalizowane podejście, które z teoretycznego punktu widzenia gwarantuje większe bezpieczeństwo w zamian za uzależnienie systemu od centralnego urzędu, wymaga wdrożenia infrastruktury klucza publicznego (PKI — *Public Key Infrastructure*). PKI jest usługą odpowiedzialną za tworzenie, unieważnianie, dystrybucję oraz aktualizację kluczy i certyfikatów. Jej działanie wymaga dostępności **urzędów certyfikacji** (CA — *Certificate Authority*). Urząd certyfikacji jest jednostką i usługą powołaną do zarządzania i potwierdzania powiązań między dowodem tożsamości a odpowiadającym mu kluczem publicznym. W czasie pisania książki na świecie istniało kilkaset komercyjnych urzędów certyfikacji, tworzących w większości przypadków **hierarchiczną** zależność. Dzięki niej klucz publiczny może być podpisywany kluczem rodzica, który z kolei został podpisany kluczem dziadka itd. Urzędy certyfikacji dysponują pojedynczymi lub wieloma **certyfikatami głównych urzędów**, które zapewniają przeniesienie zaufania na wiele certyfikatów podrzędnych. Jednostka, która jest upoważniona do wystawiania certyfikatów i generowania kluczy (np. CA), jest nazywana **kotwicą zaufania** (*trust anchor*), choć tą samą nazwą określa się certyfikaty i inne dane kryptograficzne powiązane z tymi jednostkami [RFC6024] (zagadnienie to zostało opisane w kolejnym punkcie).

18.5.1. Certyfikaty kluczy publicznych, urzędy certyfikacji i standard X.509

Certyfikaty mają wiele różnych form, ale ta, która interesuje nas najbardziej, bazuje na internetowym profilu standardu ITU-T X.509 [RFC5280]. Każdy certyfikat można zapisać lub przesłać w jednym z wielu formatów kodowania. Najpowszechniej stosowane są jednak formaty DER, PEM (wersja formatu DER zakodowana zgodnie z algorytmem Base64), PKCS#7 (P7B) oraz PKCS#12 (PFX). W rozdziale 8. opisane zostało wykorzystanie podobnego formatu — PKCS#1 [RFC3447]. Obecnie internetowe standardy PKI bazują w większości na **składni komunikatów kryptograficznych** [RFC5652], która wywodzi się z wersji 1.5 standardu PKCS#7. W kolejnym analizowanym przykładzie opisane zostało użycie certyfikatu X.509 w formacie PEM, który jest domyślnym formatem większości aplikacji internetowych i ma tę zaletę, że pozwala na przedstawienie treści certyfikatu w kodowaniu ASCII.

Certyfikaty służą do identyfikowania czterech rodzajów podmiotów działających w internecie: użytkowników, serwerów, dostawców oprogramowania oraz urzędów CA. Jeden z najbardziej popularnych urzędów certyfikacji (Verisign) przypisuje każdemu certyfikatowi pewną „klasę" z przedziału od 1 do 5. Certyfikaty klasy 1. są przeznaczone dla użytkowników końcowych. Klasa 2. obejmuje organizacje. Klasa 3. jest przeznaczona dla serwerów i operacji podpisywania oprogramowania. Klasa 4. odnosi się do transakcji sieciowych realizowanych między firmami. Natomiast klasa 5. jest przeznaczona dla organizacji prywatnych i rządów. Klasyfikowanie certyfikatów ma na celu przede wszystkim uproszczenie nazewnictwa oraz grupowanie certyfikatów ze względu na pewne cechy polityki bezpieczeństwa związane z nimi. Ogólnie rzecz ujmując, klasa o wyższym numerze wymusza bardziej rygorystyczne **sprawdzenie tożsamości** podmiotu przed wystawieniem certyfikatu.

Opisywane rozwiązanie nie eliminuje wspomnianego wcześniej problemu „jajka i kury". W praktyce systemy operujące kluczami publicznymi muszą dysponować certyfikatami jednostek CA, które były najczęściej wykorzystywane w czasie tworzenia pakietu in-

stalacyjnego oprogramowania (np. aplikacje, takie jak Microsoft Internet Explorer, Mozilla Firefox i Google Chrome, mają możliwość odwoływania się do wstępnie skonfigurowanej bazy danych certyfikatów głównych urzędów certyfikacji). Aby zobaczyć, w jaki sposób działa system certyfikatów, wystarczy wpisać polecenie odpowiedzialne za wyświetlenie informacji o certyfikatach. Sprawdzenie certyfikatu witryny internetowej w systemie Linux lub Windows sprowadza się do użycia polecenia `openssl` w następującej postaci (niektóre wiersze zostały złamane w celu zwiększenia czytelności listingu):

```
Linux% CDIR=`openssl version -d | awk '{print $2}'`
Linux% openssl s_client -CApath $CDIR \
       -connect www.digicert.com:443 > digicert.out 2>&1
^C (aby przerwać wykonywanie)
```

Pierwsza instrukcja pobiera informacje o katalogu, w którym system przechowuje wstępnie skonfigurowane certyfikaty CA. Katalog ten może być inny w każdym systemie. Niemniej jednak ścieżka dostępu do niego zostanie zapisana w zmiennej powłoki `CDIR`. Kolejne polecenie powoduje ustanowienie połączenia z portem HTTPS (443) serwera *www.digicert.com* i przekierowanie wyniku tej operacji do pliku *digicert.out*. Instrukcja `openssl`[2] zapewnia wyświetlenie informacji o każdym podmiocie identyfikowanym przez poszczególne certyfikaty wraz z danymi na temat poziomu w hierarchii wyznaczanej względem głównego urzędu certyfikacji (poziom 0 odpowiada certyfikatowi serwera, więc wartości poziomu należy liczyć od dołu w górę drzewa). Ponadto polecenie porównuje certyfikaty z zapisanymi lokalnie certyfikatami urzędów CA. W przedstawionym przykładzie weryfikacja zakończyła się powodzeniem, o czym świadczy wartość `0 (ok)` wyniku weryfikacji (`Verify return code`).

```
Linux% grep "return code" digicert.out
    Verify return code: 0 (ok)
```

Plik *digicert.out* zawiera nie tylko informacje na temat połączenia z serwerem, ale również kopię certyfikatu serwera. Aby uzyskać certyfikat w nieco bardziej użytecznej postaci, można wyodrębnić jego dane i przekształcić je w plik zakodowany zgodnie ze standardem PEM:

```
Linux% openssl x509 -in digicert.out -out digicert.pem
```

Dysponując certyfikatem w formacie PEM, możemy skorzystać z różnorodnych funkcji polecenia `openssl` do przetwarzania i weryfikowania danych. Na najwyższym poziomie przedstawione są pewne informacje przeznaczone do podpisania. Po nich prezentowany jest identyfikator algorytmu podpisu cyfrowego oraz wartość podpisu. Aby wyświetlić treść certyfikatu serwera, wystarczy wprowadzić następujące polecenie (niektóre wiersze zostały złamane w celu zwiększenia czytelności listingu):

```
Linux% openssl x509 -in digicert.pem -text
openssl x509 -in digicert.pem -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            04:4d:ff:ee:75:e4:53:ec:29:bf:e5:8a:76:7a:c1:c7
```

[2] W systemie Windows 2003 Server oraz w dodatku Windows Server 2003 Administration Pack znajduje się polecenie `certutil` o podobnym działaniu.

```
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=US, O=DigiCert Inc, OU=www.digicert.com,
        CN=DigiCert High Assurance EV CA-1
Validity
    Not Before: Feb 22 00:00:00 2012 GMT
    Not After : May 17 12:00:00 2014 GMT
Subject: 2.5.4.15=Private Organization/
         1.3.6.1.4.1.311.60.2.1.3=US/
         1.3.6.1.4.1.311.60.2.1.2=Utah/
         serialNumber=5299537-0142/
         streetAddress=Canopy Building II, Suite 200/
         streetAddress=355 South 520 West/
         postalCode=84042, C=US, ST=Utah, L=Lindon, O=DigiCert, Inc.,
         CN=www.digicert.com
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (2048 bit)
        Modulus (2048 bit):
            00:a3:fB:82:aa:eB:9b:56:ce:06:f5:91:c2:81:6c:
            1c:2e:48:a2:b0:fa:2e:70:98:09:92:6e:15:6e:07:
            2d:16:bc:4e:49:42:41:3e:d6:49:29:0b:eB:24:51:
            ...
            55:7a:77:49:81:32:64:B6:7f:fb:77:99:3e:1d:38:
            9a:07
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Authority Key Identifier:
        keyid:4C:58:CB:25:F0:41:4F:52:F4:
        28:CB:81:43:98:A6:A8:A0:E6:92:E5

    X509v3 Subject Key Identifier:
        EB:35:98:DE:FF:72:84:02:F3:98:
        OF:DE:60:B2:71:AC:EE:D4:3A:22
    X509v3 Subject Alternative Name:
        DNS:www.digicert.com, DNS:digicert.com, DNS:content.digicert.com,
        DNS:www.origin.digicert.com
    X509v3 Key Usage: critical
        Digital Signature, Key Encipherment
    X509v3 Extended Key Usage:
        TLS Web Server Authentication, TLS Web Client Authentication
    X509v3 CRL Distribution Points:
        URI:http://crl3.digicert.com/evca1-g1.crl
        URI:http://crl4.digicert.com/evca1-g1.crl

    X509v3 Certificate Policies:
        Policy: 2.16.840.1.114412.2.1
          CPS: http://www.digicert.com/ssl-cps-repository.htm
          User Notice:
            Explicit Text:

    Authority Information Access:
        OCSP - URI:http://ocsp.digicert.com
        CA Issuers - URI:http://cacerts.digicert.com/
        DigiCertHighAssuranceEVCA-1.crt

    X509v3 Basic Constraints: critical
        CA:FALSE
```

```
    Signature Algorithm: sha1WithRSAEncryption
        17:4b:b1:0c:Bd:90:f9:6B:4f:2b:63:60:d9:da:4e:62:15:2B:
        36:07:46:9e:a3:4e:d0:a9:2d:Ba:fd:10:63:25:16:3b:d9:ec:
        ...
        93:3d:Ba:Bd:75:fa:61:29:fe:a3:3b:c0:7a:7e:80:49:6d:14:
        29:47:c0:93
-----BEGIN CERTIFICATE-----
MIIHljCCBn6gAwI8AgIQBE3/7nXkU+wpv+WKdnrBxzANBgkqhkiG9w0BAQUFADBp
MQswCQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMRkwFwYDVQQLExB3
...
lDGtx8rV8GYOYa5EK6Lg0rAup5lgFZrccp8F7Lk2Nc/lGDT/PD4r8HFG/QRiRsAD
LAO01JM9io11+mEp/qM7wHp+gEltFClHwJM=
-----END CERTIFICATE-----
```

Wynik wykonania polecenia zawiera zdekodowaną wersję certyfikatu, po której następuje blok znaków ASCII z treścią certyfikatu zapisaną w formacie PEM (pomiędzy wierszami `BEGIN CERTIFICATE` i `END CERTIFICATE`). W zdekodowanej części widoczny jest blok informacji oraz blok podpisu. W części z informacjami zawarte są pewne metadane, w tym pole `Version` (wersja) wskazujące typ certyfikatu X.509 (najnowsza wersja 3. ma identyfikator o wartości szesnastkowej 0x02), pole `Serial Number`, czyli numer seryjny certyfikatu, niepowtarzalny numer certyfikatu przypisywany przez dany urząd CA oraz pole `Validity` (ważność) odzwierciedlające czas, w którym certyfikat należy uznać za ważny (okres ważności rozpoczyna się od daty zapisanej w polu `Not Before` i trwa do daty zawartej w polu `Not After`). Sekcja metadanych informuje również o tym, jaki algorytm podpisu cyfrowego został użyty do podpisania danych. W tym przypadku wygenerowany został skrót SHA-1, który podpisano za pomocą algorytmu RSA. Sam podpis jest zawarty w końcowej części certyfikatu.

Pole `Issuer` zawiera **jednoznaczną nazwę** (*distinguished name*; termin pochodzący ze standardu ITU-T X.500) jednostki, która wystawiła certyfikat. Wartość ta składa się z kilku pól (zgodnych ze standardem X.501): `C` (kraj [*country*]), `L` (lokalizacja lub miasto [*locality*]) `O` (organizacja [*organization*]), `OU` (jednostka organizacyjna [*organizational unit*]), `ST` (stan lub województwo [*state*]) oraz `CN` (nazwa potoczna [*common name*]). Standard obejmuje również inne pola składowe, z których jedno `EV` (rozszerzona weryfikacja [*extended validation*]) [CABF09] jest widoczne również w prezentowanym przykładzie. Oznacza ono, że do podpisania certyfikatu serwera został wykorzystany certyfikat CA.

Mechanizm EV jest odpowiedzią przemysłu komputerowego na pewne rodzaje ataków związanych z wyłudzaniem danych przez złośliwe witryny internetowe, którym wystawiono certyfikaty bez należytego sprawdzenia tożsamości. Uzyskanie certyfikatu EV wymaga spełnienia wielu surowych kryteriów. Użytkownik odwiedzający witrynę z certyfikatem EV jest informowany o podwyższonym poziomie bezpieczeństwa za pomocą specjalnego zielonego paska wyświetlanego przez większość nowoczesnych przeglądarek internetowych. Jednym z warunków uzyskania uprawnień do wydawania certyfikatów EV jest przedłożenie przez urząd certyfikacji **oświadczenia o zasadach certyfikacji** (CPS — *Certification Practice Statement*), które opisuje praktyki organizacji związane z wydawaniem certyfikatów. Zalecenia dla autorów opracowania CPS (oraz **polityki certyfikacji** [CP — *Certificate Policy*] odnoszącej się do poszczególnych certyfikatów) są podane w dokumencie [RFC5280]. Choć certyfikaty EV zapewniają wyższy poziom bezpieczeństwa (np. w korzystaniu z serwisów internetowych), większość użytkowników nie zwraca szczególnej uwagi na informacje o tym fakcie generowane przez przeglądarki internetowe [BOPSW09].

Pole `Subject` opisuje stronę, której dotyczy dany certyfikat, oraz właściciela klucza publicznego. Sam klucz publiczny jest przekazywany w polu `Subject Public Key Info`. W przedstawionym przykładzie wartość pola `Subject` jest dość złożona, podobnie jak wartości pola `Issuer`, i składa się z wielu **identyfikatorów obiektów** (OID — *Object ID*) [ITUOID]. Część jest zapisana za pomocą nazw (z wykorzystaniem pól `O`, `C`, `ST`, `L` i `CN`), ale nie wszystkie. Przyczyną jest to, że dana wersja narzędzia `openssl` nie interpretuje odpowiednio uzyskanych informacji. Wartość OID 1.3.6.1.4.1.311.60.2.1.3 jest również oznaczana jako `jurisdictionOfIncorporationCountryName` (nazwa kraju firmy), a identyfikator 1.3.6.1.4.1.311.60.2.1.2 jako `jurisdictionOfIncorporationStateOrProvinceName` (nazwa stanu lub województwa). Wartość OID 2.5.4.15 odpowiada rodzajowi działalności (`businessCategory`) — więcej informacji na ten temat można znaleźć w opracowaniu [CABF09]. Podczas identyfikacji strony komunikacji oraz wystawcy certyfikatu najważniejsze wydaje się jednak pole `CN`. W analizowanym przypadku przedstawia ono poprawną nazwę serwera (wraz ze wszystkimi nazwami uwzględnionymi w polu alternatywnych nazw podmiotu — `Subject Alternative Name` (SAN)). Odwołania z użyciem niewymienionych nazw lub ciągów URL (np. pod adres *https://digicert.com* zamiast *https://www.digicert.com*), nawet jeśli odnoszą się do tego samego serwera, powodują zwrócenie błędu. Warto jednak pamiętać, że pole `CN` nie jest przeznaczone do przechowywania nazwy domenowej. Do tego celu służą wartości SAN.

Weryfikacja certyfikatu polega na wykonaniu szeregu rekurencyjnych operacji na drzewie zależności (w kierunku głównego urzędu certyfikacji), w których trakcie porównywana jest każdorazowo nazwa wystawcy (`Issuer`) danego certyfikatu z nazwą podmiotu (`Subject`) zapisaną w certyfikacie wyższym w hierarchii. W omawianym przykładzie certyfikat został wystawiony przez `DigiCert High Assurance EV CA-1` (element `CN` pola `Issuer`). Jeśli wszystkie certyfikaty są ważne (nie upłynął ich termin ważności) oraz są wykorzystywane we właściwy sposób, a jeden z certyfikatów nadrzędnych względem sprawdzanego (rodzic, dziadek itd.; najczęściej certyfikat głównego urzędu certyfikacji) jest uznawany za zaufany, operacja weryfikacji kończy się wynikiem pozytywnym.

Pole `Subject Public Key Info` dostarcza informacji na temat algorytmu oraz klucza publicznego, którymi operuje jednostka wymieniona w polu `Subject`. W tym przypadku kluczem publicznym jest wartość wygenerowana przez algorytm RSA o module 2048 bitów oraz publicznym wykładniku wynoszącym 65537. Dana jednostka dysponuje kluczem prywatnym algorytmu RSA (o właściwym module i prywatnym wykładniku) odpowiadającym wymienionemu kluczowi publicznemu. W przypadku ujawnienia klucza prywatnego lub zmiany z jakichkolwiek powodów klucza publicznego konieczne jest ponowne wygenerowanie pary kluczy (prywatnego i publicznego) i wystawienie nowego certyfikatu. Bieżący certyfikat jest wówczas unieważniany (więcej informacji na ten temat znajduje się w punkcie 18.5.2).

Certyfikaty X.509 w wersji 3. mogą zawierać dodatkowe **rozszerzenia** o charakterze **krytycznym** lub **niekrytycznym**. Mimo że są opcjonalne, część z nich jest wymagana w profilu internetowym [RFC5280]. Rozszerzenia krytyczne muszą zostać przeanalizowane i uznane za zgodne z polityką strony zależnej (terminologia CPS). Rozszerzenia niekrytyczne są przetwarzane, jeśli dana jednostka je obsługuje, ale zignorowanie ich nie powoduje błędu. W przedstawionym przykładzie można wyróżnić dziewięć rozszerzeń X.509v3. Choć opracowano wiele różnych definicji, większość z nich należy do dwóch nieformalnych kategorii. Pierwsza kategoria obejmuje informacje na temat pod-

miotu oraz zasad wykorzystywania certyfikatu. Do drugiej kategorii należą dane na temat wystawcy wraz z identyfikatorami kluczy oraz adresami URI, pod którymi dostępne są dodatkowe informacje na temat urzędu CA wystawiającego certyfikat. Analizowany certyfikat należy do **jednostki końcowej** (a nie do jednostki CA). Certyfikaty jednostek CA zawierają zazwyczaj nieco inne rozszerzenia oraz wartości tych rozszerzeń.

O tym, czy certyfikat należy do jednostki CA, informuje rozszerzenie krytyczne o nazwie `Basic Constraints`. Z zaprezentowanego listingu wynika, że certyfikat nie należy do urzędu CA, więc nie może zostać wykorzystany do podpisania innego certyfikatu. Z kolei certyfikat zawierający informację o przynależności do jednostki CA można uznać za prawidłowy w procesie weryfikacji tylko wtedy, jeśli znajduje się na poziomie innym niż najniższy (nie jest liściem drzewa). Jest to cecha wszystkich certyfikatów głównych urzędów CA oraz certyfikatów służących do podpisywania innych certyfikatów (certyfikatów pośrednich, takich jak `DigiCert High Assurance EV CA-1`).

Rozszerzenie `Subject Key Identifier` identyfikuje klucz publiczny danego certyfikatu. Umożliwia rozróżnianie poszczególnych kluczy tego samego podmiotu. Z kolei krytyczne pole `Key Usage` określa sposób wykorzystania danego klucza. Lista dozwolonych zastosowań obejmuje: generowanie podpisów cyfrowych, zapewnienie niezaprzeczalności, szyfrowanie kluczy, szyfrowanie danych, uzgadnianie kluczy, podpisywanie certyfikatów, podpisywanie list CRL (więcej informacji na ten temat znajduje się w punkcie 18.5.2), tylko szyfrowanie oraz tylko deszyfrowanie. Ponieważ certyfikaty serwerów są zazwyczaj wykorzystywane do identyfikowania dwóch stron połączenia i szyfrowania klucza sesji (patrz punkt 18.9), zakres zastosowań klucza bywa w takich przypadkach bardzo ograniczony. Rozszerzenie `Extended Key Usage` może mieć charakter krytyczny lub niekrytyczny i dostarcza dodatkowych informacji na temat ograniczeń w użyciu klucza. W profilu internetowym zwykle definiuje następujące obszary zastosowań: uwierzytelnienie klienta i serwera TLS, podpisywanie pobieranego kodu, ochrona poczty elektronicznej (niezaprzeczalność i uzgadnianie kluczy bądź szyfrowanie), różnorodne tryby pracy protokołów IPsec (patrz podrozdział 18.8) oraz generowanie znaczników czasu. Rozszerzenie SAN pozwala na użycie pojedynczego certyfikatu do różnych celów (np. w wielu witrynach internetowych o różnych nazwach domenowych), dzięki czemu nie ma potrzeby utrzymywania oddzielnych certyfikatów w każdym serwisie internetowym, co z kolei znacznie zmniejsza koszty i bałagan administracyjny. W analizowanym przypadku certyfikat może zostać wykorzystany w połączeniach z serwerami o adresach domenowych `www.digicert.com`, `digicert.com`, `content.digicert.com`, `www.origin.digicert.com`.

Pozostałe rozszerzenia przedstawionego certyfikatu opisują sposób zarządzania nim i jego status oraz status wystawiającej go jednostki CA. Sekcja `CRL Distribution Points` (CDP) zawiera listę adresów URL, pod którymi dostępne są **listy unieważnionych certyfikatów** (CRL — *Certificate Revocation List*) danego urzędu certyfikacji. Na jej podstawie można ustalić, czy określony certyfikat z łańcucha zależności nie został unieważniony (patrz punkt 18.5.2). Rozszerzenie `Certificate Policies` (CP) przechowuje informację o politykach certyfikacji odnoszących się do danego certyfikatu [RFC5280]. W przykładowym listingu uwzględnione zostały dwa **kwalifikatory**. Element `Policy` o wartości 2.16.840.1.114412.2.1 informuje o tym, że certyfikat spełnia założenia polityki EV. Z kolei element `CPS` przechowuje adres URI dokumentu CPS danej polityki. Pole `User Notice` może natomiast zawierać tekst przeznaczony do wyświetlenia po stronie zależnej.

Rozszerzenie `Authority Key Identifier` jest identyfikatorem klucza publicznego powiązanego z kluczem prywatnym, który został użyty do podpisania niniejszego certyfikatu. Wartość ta jest przydatna szczególnie wtedy, kiedy wydawca certyfikatu dysponuje wieloma kluczami prywatnymi przeznaczonymi do generowania podpisów cyfrowych. Z kolei rozszerzenie `Authority Information Access` (AIA) obejmuje adresy, pod którymi można pobrać z jednostki CA dodatkowe informacje. W prezentowanym przykładzie jest nim wymieniony adres pozwalający na sprawdzenie za pomocą bezpośredniego zapytania, czy dany certyfikat nie został unieważniony (patrz punkt 18.5.2). Sekcja ta umożliwia również pobranie listy urzędów CA, na której występuje adres URL certyfikatu, jaki posłużył do podpisania certyfikatu analizowanego serwera.

Poniżej rozszerzeń znajduje się sekcja podpisu. Obejmuje ona identyfikator algorytmu (w tym przypadku SHA-1 i RSA), który musi odpowiadać wartości omówionego wcześniej pola `Signature Algorithm`. W prezentowanym przykładzie podpis jest wartością składającą się z 256 bajtów, odpowiadającą 2048-bajtowemu modułowi algorytmu RSA.

18.5.2. Walidacja i unieważnianie certyfikatów

We wcześniejszej części rozdziału przewijały się wzmianki o tym, że certyfikat można unieważnić i zastąpić nowym. Zgodnie z zaleceniami IETF opisanymi w dokumencie [RFC5280] w zastosowaniach internetowych wykorzystuje się certyfikaty X.509 w wersji 3. oraz listy CRL opisane standardem X.509 w wersji 2. Nasuwa się więc pytanie, w jaki sposób unieważnia się certyfikat oraz skąd strony zależne wiedzą, że dany certyfikat stracił swoją wiarygodność?

Aby zweryfikować certyfikat, konieczne jest wyznaczenie **ścieżki walidacyjnej** (*validation path*) lub **ścieżki certyfikacji** (*certification path*), która obejmuje pewien zbiór sprawdzanych certyfikatów powiązanych na pewnym poziomie z kotwicą zaufania (np. z certyfikatem głównego urzędu certyfikacji) znaną stronie zależnej. Jednym z najważniejszych zadań w całej operacji jest ustalenie, czy którykolwiek z certyfikatów w łańcuchu nie został unieważniony. Jeśli tak, cała ścieżka walidacyjna jest uznawana za niewłaściwą. Przykład takiej sytuacji został opisany w punkcie 8.5.5.

Certyfikaty są unieważniane z kilku powodów. Jednym z nich może być zmiana afiliacji lub nazwy podmiotu (albo wystawcy). Po unieważnieniu certyfikat staje się bezużyteczny. Problemem jest jednak zapewnienie, że wszystkie jednostki, które chciałyby z niego skorzystać, zostaną poinformowane o utracie ważności. W komunikacji internetowej istnieją dwa sposoby wykonania tego zadania — wykorzystanie list CRL oraz **protokołu weryfikacji statusu certyfikatu** (OCSP — *Online Certificate Status Protocol*) [RFC2560]. Jeśli w rozszerzeniu `CRL Distribution Point` zapisane są adresy URI usług http lub FTP (tak jak w poprzednim przykładzie), odnoszą się one do pliku w formacie DER, który przechowuje listę CRL zgodną ze standardem X.509. Aby pobrać listę CRL wymienioną w opisywanym wcześniej certyfikacie, należy wykonać następujące polecenie:

```
Linux% wget http://crl3.digicert.com/evca1-g1.crl
```

Następnie za pomocą poniższej instrukcji można wyświetlić wynik na ekranie.

```
Linux% openssl crl -inform der -in evca1-g1.crl -text
Certificate Revocation List (CRL):
        Version 2 (0x1)
```

```
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: /C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert High Assurance EV CA-1
        Last Update: Apr  7 17:00:41 2012 GMT
        Next Update: Apr 14 17:00:00 2012 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:4C:58:C8:25:F0:41:4F:52:F4:28:C8:81:43:9B:A6:A8:A0:E6:92:E5

            X509v3 CRL Number:
                132
Revoked Certificates:
    Serial Number: 0CAD2D6AED30F8416FECE2DC4D8BB7EA
        Revocation Date: Dec  9 14:03:23 2011 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Unspecified
...
    Serial Number: 0288E755CBB9999681F79CCC62169C35
        Revocation Date: Apr  6 16:01:10 2012 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Unspecified
    Signature Algorithm: sha1WithRSAEncryption
        a5:24:3a:74:6f:f3:9a:75:80:a9:8a:83:45:be:4c:25:f1:7c:
        81:f2:11:e2:49:b0:ce:84:21:93:60:03:42:d8:05:e7:3a:bb:
        ...
        48:16:c5:43:40:d0:4e:c1:ea:07:90:98:9c:42:90:53:42:68:
        fa:4d:83:ce
-----BEGIN X509 CRL-----
MIILSjCCCjICAQEwDQYJKoZIhvcNAQEFBQAwaTELMAkGA1UEBhMCVVMxFTATBgNV
BAoTDERpZ2lDZXJ0IEluYzEZMBcGA1UECxMQd3d3LmRpZ2ljZXJ0LmNvbTEoMCYG
...
c+m7aXWn8D6Aggi5dIBv7EO3a23ZLrVB33nQAoEkbmwTZ6Az/WHHHEgWxUNAOE7B
6geQmJxCkFNCaPpNg84=
-----END X509 CRL-----
```

Powyższy listing przedstawia listę CRL zgodną ze standardem X.509. Jej format jest bardzo zbliżony do stosowanego w zapisie certyfikatów, a cała treść (podobnie jak w przypadku certyfikatów) jest podpisywana przez urząd CA. Jest to bardzo użyteczne rozwiązanie, ponieważ listy CRL są dystrybuowane na takich samych zasadach jak certyfikaty — za pośrednictwem niezabezpieczonych kanałów komunikacyjnych i serwerów. Jednak w przeciwieństwie do certyfikatów okres ważności zestawienia został zastąpiony informacjami o wcześniejszej i kolejnej aktualizacji wykazu. Nie uwzględniono w nim także nazwy podmiotu oraz klucza publicznego. Włączono natomiast listę numerów seryjnych unieważnionych certyfikatów oraz wyjaśnienie przyczyny unieważnienia. Lista może również obejmować rozszerzenia charakterystyczne dla tego rodzaju zestawień. W analizowanym przypadku widoczne jest rozszerzenie `Authority Key Identifier`, które przechowuje identyfikator klucza użytego przez jednostkę CA do podpisania listy CRL. Z kolei rozszerzenie `CRL Number` udostępnia numer seryjny samej listy. Pozostałe wartości zostały opisane w dokumencie [RFC5280].

Podstawowym sposobem sprawdzenia, czy certyfikat nie został unieważniony, jest wykorzystanie protokołu OCSP. Mechanizm OCSP jest protokołem warstwy aplikacji, który generuje żądania i odpowiedzi HTTP (tzn. bazuje na protokole HTTP z użyciem portu

TCP 80). Żądanie OCSP zawiera dane identyfikujące określony certyfikat oraz ewentualne dodatkowe rozszerzenia. W odpowiedzi z kolei mogą być zawarte informacje o tym, że certyfikat nie został unieważniony, został unieważniony lub jest nieznany. W przypadku problemów z interpretacją żądania po stronie serwera lub jego przetworzenia, zwracany jest komunikat o błędzie. Klucz użyty do podpisania odpowiedzi OCSP nie musi odpowiadać kluczowi wykorzystanemu do podpisania samego certyfikatu, szczególnie jeśli jego wystawca zdefiniował pole `Key Usage`, określając alternatywny serwer OCSP.

Aby przeanalizować wymianę żądań i odpowiedzi w mechanizmie OCSP, wystarczy wykonać poniższe polecenia (zakładamy, że w pliku *DigiCertHighAssuranceEVCA-1.pem* został uprzednio zapisany certyfikat klasy 1.). Niektóre wiersze listingu zostały dodatkowo złamane w celu poprawienia czytelności.

```
Linux% CERT=DigiCertHighAssuranceEVCA-1.pem
Linux% openssl ocsp -issuer $CERT -cert digicert.pem \
       -url http://ocsp.digicert.com -VAfile $CERT -no_nonce -text
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: 88A299F09D061DD5C1588F76CC89FF57092894DD
          Issuer Key Hash: 4C58C825F0414F52F428C8814398A6A8A0E692E5
          Serial Number: 02C71FE01D70414888A7E29E5E584289

OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: 4C58C825F0414F52F428C881439BA6A8A0E692E5
    Produced At: Jan 2 08:03:24 2011 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 88A299F09D061DD5C1588F76CC89FF57092894DD
      Issuer Key Hash: 4C58C825F0414F52F428C8814398A6A8A0E692E5
      Serial Number: 02C71FE01D7041488BA7E29E5E5842B9
    Cert Status: good
    This Update: Jan 2 08:03:24 2011 GMT
    Next Update: Jan 9 08:18:24 2011 GMT

Response verify OK
digicert.pem: good
    This Update: Jan 2 08:03:24 2011 GMT
    Next Update: Jan 9 08:18:24 2011 GMT
```

Jak nietrudno zauważyć, transakcja OCSP została zakończona z wynikiem pozytywnym. W odpowiedzi uwzględnione zostały: identyfikator funkcji skrótu (SHA-1), skrót nazwy wystawcy, numer identyfikujący klucz wystawcy (o takiej samej wartości jaka występuje w rozszerzeniu `Key ID` w certyfikacie) oraz numer seryjny certyfikatu. Odpowiedź została wygenerowana i podpisana przez nadawcę, którego identyfikator znajduje się w polu `Responder ID`. Zawiera skróty i numery przesłane w żądaniu oraz informację o statusie certyfikatu — w tym przypadku `good`, co oznacza, że certyfikat nie został unieważniony. Protokół OCSP ułatwia pracę klienta, ponieważ nie eliminuje

konieczność pobierania bieżącej listy CRL w celu sprawdzenia certyfikatu. Nie zwalnia jednak z obowiązku sprawdzenia całej ścieżki certyfikacji, co niekiedy może być dość uciążliwe dla stacji klienckiej.

Aby ograniczyć pobieranie danych na temat zależności między certyfikatami i usprawnić spoczywający na stacjach klienckich obowiązek ich weryfikacji, opracowano **protokół walidacji certyfikatu po stronie serwera** (SCVP — *Server-based Certificate Validation Protocol*) [RFC5055], który jednak nie cieszy się szczególnie dużą popularnością. Mechanizm SCVP pozwala na zlecenie serwerowi zadania ustalenia ścieżki certyfikacji (DPD — *Delegated Path Discovery*) i opcjonalnie sprawdzenia jej (DPV — *Delegated Path Validation*). Operacja walidacji może zostać zlecona jedynie zaufanym serwerom. Zastosowanie opisywanego protokołu zapewnia nie tylko zmniejszenie obciążenia stacji klienckich, ale pozwala również na wdrożenie jednolitej polityki weryfikacji certyfikatów w całym przedsiębiorstwie.

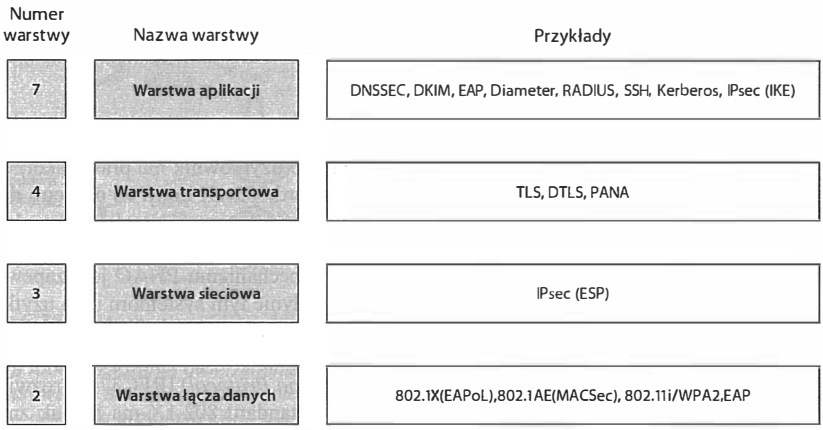
18.5.3. Certyfikaty atrybutów

Poza certyfikatami kluczy publicznych (PKC — *Public Key Certificate*) wykorzystywanymi do wiązania nazw z kluczami publicznymi, standard X.509 definiuje również **certyfikaty atrybutów** (AC — *Attribute Certificate*). Struktura certyfikatów AC jest zbliżona do PKC, ale nie uwzględnia klucza publicznego. Certyfikaty te są używane do przekazywania innego rodzaju danych, w tym informacji o autoryzacji, które mogą mieć inny okres ważności (krótszy) niż odpowiadające im certyfikaty PKC [RFC5755]. W ich treści, podobnie jak w certyfikatach PKC, wyróżnione zostały rozszerzenia i polityki stosowania.

18.6. Protokoły bezpieczeństwa stosu TCP/IP i podział na warstwy

Z dotychczasowych rozważań wynika, że kryptografia zapewnia fundament pod budowę systemów komunikacyjnych o odpowiednich zabezpieczeniach. Protokoły obejmujące elementy kryptograficzne mogą być (i są) implementowane na różnych poziomach stosu protokołów. Znając opisany w 1. rozdziale model odniesienia OSI, możemy stwierdzić, że szyfrowanie i różne inne formy zabezpieczania transmisji można wdrożyć w niemal każdej warstwie tego modelu.

Jak nietrudno się domyśleć, zabezpieczenia na poziomie warstwy łącza danych chronią informacje tylko na pojedynczym odcinku transmisyjnym. Mechanizmy systemu bezpieczeństwa implementowane w warstwie sieciowej obejmują swoim działaniem przekaz między dwiema jednostkami końcowymi. W warstwie transportowej zabezpieczana jest komunikacja między procesami. Natomiast mechanizmy warstwy aplikacji dbają o bezpieczeństwo informacji przetwarzanych przez aplikację. Oczywiście, istnieje również możliwość ochrony danych aplikacji niezależnie od rozwiązań zaimplementowanych w warstwach komunikacyjnych (nic nie stoi na przeszkodzie, żeby zaszyfrować pliki i przesłać je w formie załącznika do listu elektronicznego). Na rysunku 18.4 zostały przedstawione protokoły zabezpieczeń, które są najczęściej używane w połączeniu z protokołami stosu TCP/IP.



Rysunek 18.4. *Protokoły zabezpieczeń występują w niemal każdej warstwie stosu OSI, a także „pomiędzy warstwami". Wybór odpowiednich mechanizmów wymaga starannego przeanalizowania sposobu ich działania*

Z rysunku 18.4 wynika, że istnieje wiele protokołów odpowiedzialnych za bezpieczeństwo transmisji, a wybór właściwego zależy od zakresu funkcjonalnego potrzebnego w danej chwili. Większość z nich została opisana w dalszej części podrozdziału. Szczególnie wnikliwie przedstawione zostały mechanizmy IPsec (bezpieczny transport danych między jednostkami na poziomie warstwy 3.), TLS (zabezpieczenie warstwy transportowej) oraz DNSSEC. Protokoły TLS i IPsec są dominującymi rozwiązaniami — protokół TLS jest wykorzystywany do zabezpieczenia połączeń z serwerami WWW (HTTPS), natomiast IPsec znajduje zastosowanie w ochronie komunikacji na poziomie warstwy sieciowej, w tym w połączeniach VPN. Protokół DNSSEC, odpowiedzialny za ochronę odwołań do serwera DNS (patrz rozdział 11.), jest wdrażany dość wolno, ale przewiduje się, że będzie zyskiwał na popularności, bo zabezpieczanie systemów DNS pozwala na wyeliminowanie ataków z **przejęciem serwera DNS**, które z kolei skutkują odsyłaniem żądań klienckich do fałszywych serwerów DNS, zwracających nieprawdziwe informacje. Nie zostaną tutaj omówione dwa względnie popularne protokoły — Kerberos [RFC4120] (system uwierzytelniania wykorzystujący trzecią zaufaną jednostkę, stosowany w środowisku Windows) oraz SSH [RFC4251] (protokół bezpiecznego logowania zdalnego i tunelowania danych wykorzystywany w systemach uniksowych) — które są rozwiązaniami ściśle powiązanymi z określonymi systemami operacyjnymi (choć nie wynika to z ich budowy). W sposób szczegółowy przedstawione zostały jedynie te protokoły, które najprawdopodobniej z czasem zyskają jeszcze większą popularność w Internecie.

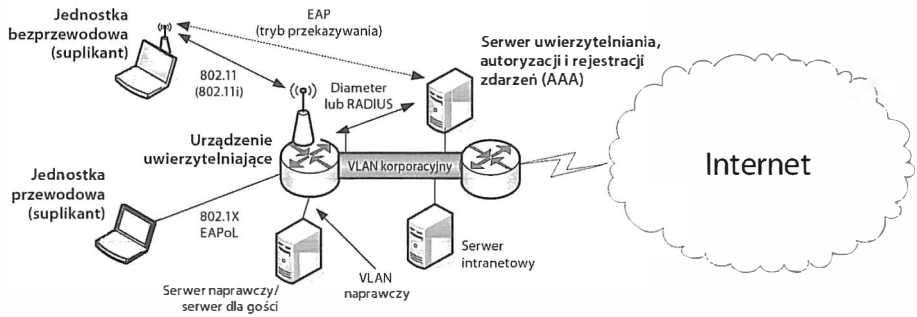
Niemal każda nowoczesna technologia sieciowa uwzględnia pewne rozwiązania gwarantujące bezpieczeństwo transmisji. Omówienie ich rozpoczniemy od dolnych warstw modelu OSI, a dokładniej od warstwy łącza danych. Z informacji zamieszczonych w rozdziale 3. wynika, że protokoły warstwy łącza danych zawierają własne mechanizmy zabezpieczeń (np. w standardzie 802.11-2007 uwzględniono mechanizm WPA2 bazujący na wcześniejszej specyfikacji 802.11i). Szczególnie interesujące są jednak te protokoły, które współdziałają w sieciach o różnym sposobie implementacji warstwy łącza danych.

18.7. Kontrola dostępu do sieci — 802.1X, 802.1AE, EAP i PANA

Kontrola dostępu do sieci (NAC — *Network Access Control*) to grupa metod wykorzystywanych do sprawdzania, czy dany system lub użytkownik ma prawo skorzystania z sieci. Opracowany przez organizację IEEE standard 802.1X **kontroli dostępu do sieci na poziomie portu** (PNAC — *Port-based Network Access Control*) z powodzeniem wspomaga zabezpieczanie komunikacji TCP/IP w przewodowych i bezprzewodowych sieciach LAN wielu przedsiębiorstw. Zadaniem mechanizmu PNAC jest zapewnienie dostępu do sieci (intranetowej lub internetowej) jedynie tym systemom i ich użytkownikom, którzy zostaną uwierzytelnieni przez urządzenie pozwalające na przyłączenie do tej sieci. Z uwagi na częste łączenie mechanizmu 802.1X z **rozszerzalnym protokołem uwierzytelniania** (EAP — *Extensible Authentication Protocol*) [RFC3748] rozwiązanie to jest nazywane EAPoL (*EAP over LAN*). Sam standard 802.1X ma jednak znacznie więcej zastosowań niż tylko formatowanie pakietów EAPoL.

Najczęściej wykorzystywany wariant mechanizmu 802.1X [802.1X-2010] bazuje na standardzie opublikowanym w 2004 roku i jest zgodny z rozwiązaniem 802.1AE (opracowanym przez IEEE standardem szyfrowania w sieciach LAN o nazwie MACSec) oraz ze specyfikacją 802.1AR (opisującą wykorzystanie certyfikatów X.509 do weryfikowania tożsamości jednostek sieciowych). Uwzględnia również dość skomplikowany protokół **uzgadniania kluczy MACSec** (MKA — *MACSec Key Agreement*), którym nie będziemy się zajmować. Zgodnie ze specyfikacją 802.1X uwierzytelniany system musi realizować zadanie **suplikanta** (*supplicant*). Suplikant porozumiewa się z **urządzeniem uwierzytelniającym** (*authenticator*) oraz pośrednio z **serwerem uwierzytelniania** (*authentication server*) w celu uzyskania dostępu do sieci na podstawie przedstawionych danych uwierzytelniających. Podejmowane przez system 802.1X decyzje o dostępie do sieci często wiążą się z wyznaczeniem określonej sieci VLAN (więcej informacji na ten temat znajduje się w rozdziale 3.).

Protokół EAP współdziała z różnymi technologiami warstwy łącza danych i udostępnia wiele metod **uwierzytelniania**, **autoryzacji** i **rejestracji zdarzeń** (AAA — *Authentication*, *Authorization*, *Accounting*). Mechanizm EAP nie realizuje jednak szyfrowania. Musi więc być stosowany w połączeniu z protokołem kryptograficznym, który zapewni bezpieczeństwo komunikacji. W przypadku użycia mechanizmu szyfrowania na poziomie warstwy łącza danych (takiego jak WPA2 w sieciach bezprzewodowych lub 802.1AE w sieciach przewodowych) rozwiązanie 802.1X okazuje się relatywnie bezpieczne. W systemie EAP również wykorzystywane są funkcje suplikanta i serwera uwierzytelnienia (podobnie jak w standardzie 802.1X), ale nieco inaczej nazwane (w specyfikacji EAP stosuje się nazwy **stacja końcowa**, **urządzenie uwierzytelniające** oraz **serwer AAA**; choć w niektórych opracowaniach dotyczących mechanizmu EAP używany jest także termin **serwera uwierzytelniania**). Przykładowa konfiguracja sieci została pokazana na rysunku 18.5.



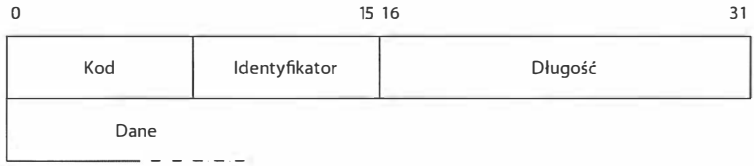
Rysunek 18.5. *Mechanizm EAP w środowisku 802.11i i 802.1X. Pozwala jednostce końcowej (suplikantowi) na uwierzytelnienie się w urządzeniu uwierzytelniającym, które jest oddzielone od serwera AAA. Urządzenie uwierzytelniające może pracować w trybie przekazywania (pass-through), który sprowadza się do powielania niezmienionych pakietów EAP. Może również brać bezpośredni udział w działaniu protokołu EAP. Tryb przekazywania pozwala na uniknięcie implementowania wielu funkcji związanych z uwierzytelnieniem w danym module sieciowym*

Na rysunku 18.5 przedstawiona została hipotetyczna sieć złożona z przewodowych i bezprzewodowych jednostek końcowych, serwera AAA oraz dwóch dodatkowych serwerów — serwera intranetowego działającego we VLAN-ie przeznaczonym dla użytkowników uwierzytelnionych oraz serwera naprawczego pracującego we VLAN-ie, do którego wpisywane są stacji nieuwierzytelnione oraz wymagające aktualizacji oprogramowania. Zadanie urządzenia uwierzytelniającego polega na przekazywaniu informacji między jednostką końcową a serwerem AAA (za pośrednictwem protokołu RADIUS [RFC2865][RFC3126] lub Diameter [RFC3588]) w celu ustalenia, czy dana jednostka powinna otrzymać prawo dostępu do chronionej sieci. Jeśli tak, stosowana jest jedna z wielu technik dalszego postępowania. Najczęstsza sprowadza się do zdefiniowania odwzorowania sieci VLAN w taki sposób, aby uwierzytelnieni użytkownicy otrzymywali dostęp do chronionego VLAN-u lub do VLAN-u zapewniającego routing (w warstwie 3.) do chronionego obszaru sieci. Urządzenie uwierzytelniające może korzystać z łączy trunkowych (lub połączeń agregowanych zgodnie z zaleceniem IEEE 802.1AX; patrz rozdział 3.) i dodawać znaczniki sieci VLAN w zależności od numeru portu lub przekazywać oznaczone ramki generowane przez stację końcową.

W niektórych implementacjach mechanizmu EAP urządzenie uwierzytelniające pracuje bez serwera AAA i musi weryfikować dane stacji końcowej we własnym zakresie. Jednostka wykonująca uwierzytelnianie jest nazywana **serwerem EAP.** Gdy urządzenie uwierzytelniające pracuje w trybie przekazywania, serwerem EAP jest serwer AAA. W przeciwnym przypadku jest nim samo urządzenie.

Zgodnie ze specyfikacją 802.1X, protokół wykorzystywany w komunikacji między suplikantem i urządzeniem uwierzytelniającym jest podzielony na górną i dolną podwarstwę. Dolna podwarstwa jest nazywana **protokołem kontroli dostępu do portu** (PACP — *Port Access Control Protocol*). Górna podwarstwa jest zazwyczaj zgodna z pewną odmianą protokołu EAP. W sieci 802.1AR jest to mechanizm EAP-TLS [RFC5216]. Protokół PACP bazuje na ramkach EAPoL, nawet jeśli mechanizm EAP nie jest wykorzystywany (gdy np. używany jest mechanizm MKA). Pole `Ethertype` w ramkach EAPoL ma wartość 0x888E (więcej informacji na ten temat znajduje się w rozdziale 3.).

Zgodnie z zaleceniami IETF, EAP nie jest pojedynczym protokołem, ale platformą zapewniającą uwierzytelnianie dzięki zastosowaniu innych protokołów, z których część została opisana dalej w tym rozdziale (np. TLS i IKEv2). Podstawowy format pakietu EAP został przedstawiony na rysunku 18.6.

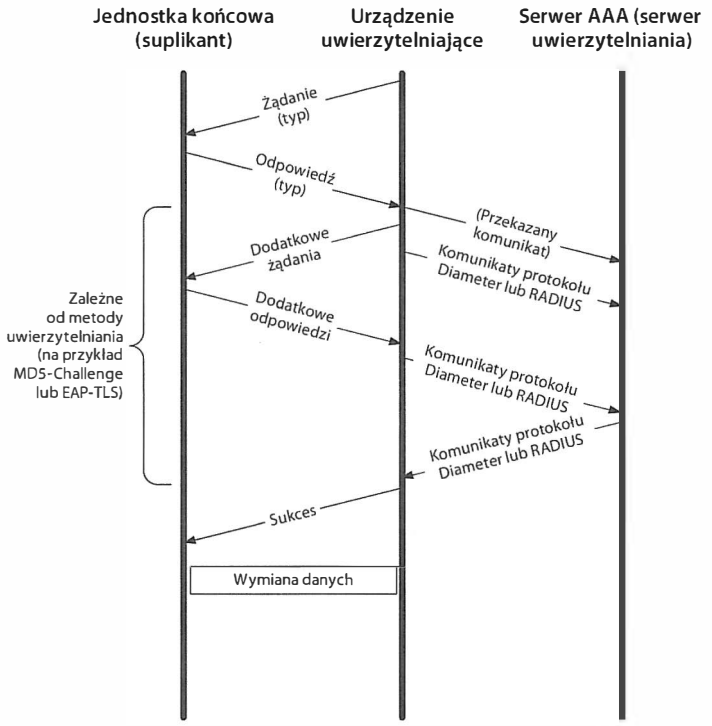


Rysunek 18.6. *Nagłówek EAP zawiera pole kodu (code) umożliwiające wydzielenie pakietów różnych typów (żądania [request], odpowiedzi [response], powiadomienia o sukcesie [success], powiadomienia o błędzie [failure], inicjacji [initiate] oraz zakończenia komunikacji [finish]). Wartość identyfikatora pozwala na powiązanie odpowiedzi z żądaniami. W przypadku żądań i odpowiedzi pierwszy bajt pola danych jest polem typu*

Format pakietu EAP nie należy do szczególnie skomplikowanych. Przedstawione na rysunku 18.6 pole `Kod` zawiera jeden z następujących identyfikatorów typu pakietu EAP: żądanie (1), odpowiedź (2), sukces (3), błąd (4), inicjacja (5), zakończenie (6). Dwa ostatnie kody są opisane w protokole ERP (*EAP Re-authentication Protocol*; omówionym w punkcie 18.7.2). Wartości pola są definiowane przez organizację IANA [IEAP]. Pole `Identyfikator` przechowuje liczbę określoną przez nadawcę, która pozwala powiązać żądania z odpowiedziami. Z kolei pole `Długość` zawiera informację o długości komunikatu EAP wyrażonej w bajtach (z uwzględnieniem pól `Kod`, `Identyfikator` i `Długość`). Do identyfikacji i uwierzytelnienia jednostki końcowej służą żądania i odpowiedzi. Operacja kończy się przesłaniem komunikatu o `Sukcesie` lub `Błędzie`. Sam protokół pozwala na przekazywanie ogólnych komunikatów informacyjnych, umożliwiających powiadomienie użytkownika o tym, co powinien zrobić w przypadku nieudanego uwierzytelnienia. Mechanizm komunikacyjny gwarantuje niezawodność i może bazować na dowolnym protokole niższej warstwy, który zapewnia dostarczanie komunikatów w odpowiedniej kolejności, ale nie musi gwarantować niezawodności. Protokół EAP nie ma zaimplementowanych takich funkcji jak sterowanie przepływem lub obsługa przeciążeń, ale może wykorzystywać do tego celu inne protokoły.

Proces wymiany informacji EAP rozpoczyna się od tego, że urządzenie uwierzytelniające przesyła żądanie do jednostki końcowej. W reakcji na nie stacja końcowa odsyła odpowiedź. Format obu komunikatów jest jednakowy (zgodny z przedstawionym na rysunku 18.6). Sam przepływ informacji odpowiada pokazanemu na rysunku 18.7.

Głównym celem emitowania żądań i odpowiedzi jest wymiana informacji niezbędnych do prawidłowego wykonania operacji uwierzytelnienia w danej *metodzie*. Wiele metod uwierzytelnienia zostało zdefiniowanych w dokumencie [RFC3748]. Część jest również opisana w oddzielnych specyfikacjach. Wyróżnik metody stosowanej w danej chwili jest zapisywany w polu `Typ` żądania i odpowiedzi, i ma wartość 4 lub większą. Inne specjalne wartości pola `Typu` odpowiadają: danym o tożsamości (1) (*identity*), powiadomieniu (2) (*notification*), negatywnego potwierdzenia (3) (*nak*) oraz rozszerzenia (254) (*expanded type*). Pierwszy z wymienionych typów jest stosowany przez urządzenie uwierzytelniające do zażądania od jednostki końcowej przesłania informacji identyfikujących ją oraz

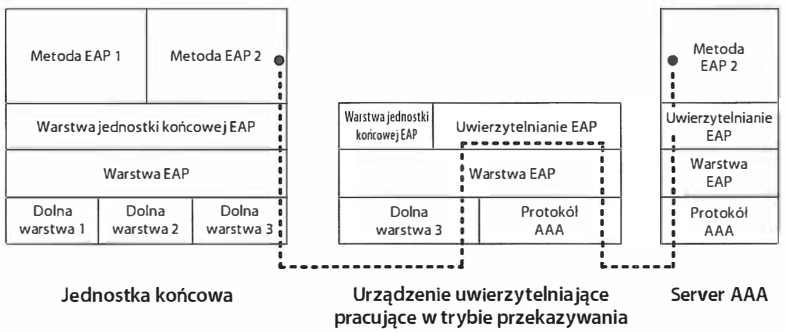


Rysunek 18.7. *Podstawowe komunikaty EAP przenoszą dane uwierzytelniające między jednostką końcową a urządzeniem sieciowym. W wielu implementacjach urządzenie uwierzytelniające jest modułem o nieskomplikowanej budowie, który pracuje w trybie przekazywania informacji. Przetwarzanie informacji jest wówczas przeniesione na jednostkę końcową i serwer AAA. Do przenoszenia danych między urządzeniem uwierzytelniającym a serwerem AAA można również wykorzystywać specjalne protokoły IEEE, takie jak RADIUS lub Diameter*

wskazania metody, zgodnie z którą stacja powinna odpowiedzieć. Typ powiadomienia jest wykorzystywany do przekazywania komunikatów i powiadomień, które są wyświetlane w interfejsie użytkownika lub zapisywane w dzienniku zdarzeń (nie są to błędy, ale informacje uzupełniające). Jeśli jednostka końcowa odeśle odpowiedź z użyciem metody innej niż wskazana przez urządzenie uwierzytelniające, moduł sieciowy reaguje, przesyłając komunikat negatywnego potwierdzenia (w standardowym formacie lub rozszerzonym). Rozszerzone komunikaty potwierdzenia negatywnego zawierają wektor zaimplementowanych metod uwierzytelniania (niedołączany do standardowego komunikatu tego typu).

System EAP jest zbudowany w sposób warstwowy i dysponuje własnym mechanizmem multipleksacji i demultipleksacji. Teoretycznie składa się z czterech warstw: dolnej warstwy (z wieloma protokołami), warstwy EAP, warstwy EAP jednostki końcowej lub urządzenia uwierzytelniającego oraz warstwy metod EAP. Dolna warstwa jest odpowiedzialna za transport ramek EAP z zachowaniem odpowiedniej kolejności. Jak na ironię, wiele protokołów wykorzystywanych do transportu komunikatów EAP to w praktyce protokoły

wyższych warstw modelu OSI, które zostały wcześniej omówione. Do grupy protokołów EAP dolnej warstwy należy zaliczyć: 802.1X, 802.11 (802.11i) (patrz rozdział 3.), UDP z L2TP (patrz rozdział 3.), UDP z IKEv2 (patrz punkt 18.8.1) oraz TCP (patrz rozdziały od 12. do 17.). Sposób implementacji opisanych warstw w urządzeniu pracującym w trybie przekazywania został przedstawiony na rysunku 18.8. Działanie serwera przekazującego miałoby odwrotny charakter, ale nie jest obsługiwane w mechanizmach RADIUS i Diameter.



Rysunek 18.8. *Stos EAP i model implementacyjny. W trybie przekazywania jednostka końcowa i serwer AAA są odpowiedzialne za obsługę metod uwierzytelniania EAP. Moduł sieciowy musi jedynie przetwarzać komunikaty EAP, realizować funkcje właściwe dla urządzenia uwierzytelniającego oraz wymieniać informacje w protokole AAA z serwerem uwierzytelnienia (np. RADIUS lub Diameter)*

W przedstawionym na rysunku 18.8 stosie EAP **warstwa EAP** zawiera mechanizmy niezawodnego dostarczania komunikatów i eliminowania duplikatów. Poza tym demultipleksuje wiadomości na podstawie kodu zawartego w pakietach EAP. **Warstwa jednostki końcowej** (**urządzenia uwierzytelniającego**) jest odpowiedzialna za obsługę komunikatów wymienianych w ramach protokołu komunikacyjnego między oboma komponentami. O sposobie przetwarzania decyduje wartość pola Kod. **Warstwa metod EAP** obejmuje implementację wszystkich metod uwierzytelniania, włączając w to wszystkie operacje niezbędne do obsługi komunikatów o dużym rozmiarze — pozostałe elementy stosu EAP nie obsługują fragmentacji, mimo że część metod wymaga przetwarzania dużych bloków informacji (np. certyfikatów lub łańcuchów certyfikatów).

18.7.1. Metody EAP i wyznaczanie klucza

Przy użyciu opisanej budowy mechanizmu EAP opracowano wiele różnych metod uwierzytelniania i enkapsulacji danych (ponad 50). Niektóre z nich zostały opisane standardami IETF, inne są rozwijane przez poszczególnych producentów (np. Cisco lub Microsoft). Do najczęściej wykorzystywanych należą: TTLS [RFC5281], TLS [RFC5216], FAST [RFC4851], LEAP (własność Cisco), PEAP (EAP w ramach TLS, własność Cisco), IKEv2 (eksperymentalna) [RFC5106] oraz MD5. Ze wszystkich wymienionych metod jedynie MD5 jest wymieniona w dokumencie [RFC3748], ale nie zaleca się już jej stosowania. Niestety, po wyborze odpowiedniej metody komplikacje się nie kończą. Większości z nich odpowiadają różne opcje zestawów kryptograficznych lub zasad weryfikacji tożsamości. Przykładowo niektóre wersje systemu Windows umożliwiają zastosowanie metody PEAP z mechanizmem MSCHAPv2 lub TLS.

Mnogość opcji wynika przede wszystkim z wcześniejszego rozwoju standardu. Niektóre z opracowanych metod zostały z czasem uznane za niedostatecznie bezpieczne lub zbyt mało elastyczne. Część z nich wymaga zbudowania infrastruktury PKI zapewniającej certyfikaty klienckie (np. EAP-TLS), w innych natomiast certyfikaty nie są potrzebne (np. PEAP i TTLS). Starsze protokoły (takie jak LEAP) były stosowane w czasach, gdy inne standardy (np. 802.11, w tym 802.11i) dopiero się rozwijały. W rezultacie wykorzystanie mechanizmu EAP wymaga niekiedy użycia kart inteligentnych, tokenów, haseł lub certyfikatów, zależnie od konkretnego środowiska.

Celem każdej z metod EAP jest przeprowadzenie operacji uwierzytelnienia, a często również autoryzacji w dostępie do sieci. W niektórych przypadkach (np. EAP-TLS) zapewniane jest wzajemne uwierzytelnienie stacji, co oznacza, że oba urządzenia pełnią funkcję jednostki końcowej oraz urządzenia uwierzytelniającego. Sposób działania mechanizmu uwierzytelniającego zależy zazwyczaj od zastosowanych w nim algorytmów kryptograficznych.

Część metod realizuje funkcje wykraczające poza samo uwierzytelnienie. Te, które zapewniają **wyznaczanie kluczy** (*key derivation*), mogą uzgadniać i eksportować klucze zgodnie z hierarchią kluczy [RFC5247] i muszą zagwarantować wzajemne uwierzytelnienie jednostki końcowej EAP i serwera EAP. W operacji wyznaczania kluczy za pomocą funkcji KDF (po stronie jednostki końcowej lub serwera) wykorzystywany jest **nadrzędny klucz sesji** (MSK — *Master Session Key*; nazywany również kluczem AAA). Wartość MSK składa się z co najmniej 64 bajtów i służy do generowania **tymczasowych kluczy sesji** (TSK — *Transient Session Key*), które są używane w procesie kontrolowania dostępu do sieci (często w niższych warstwach stosu). W czasie opisanej operacji powstają również **rozszerzone klucze MSK** (EMSK — *Extended MSK*). Są one jednak dostępne jedynie po stronie serwera EAP lub jednostki końcowej, ponieważ nie można ich przesyłać przez urządzenia uwierzytelniające. Klucze EMSK służą do wyznaczania **kluczy głównych** (*root key*) [RFC5295], które z kolei przynależą do określonych **zastosowań** lub **domen**. Z klucza EMSK powstaje **klucz główny specjalnego przeznaczenia** (USRK — *Usage-Specific Root Key*) używany do realizacji określonych zadań lub **klucz główny domeny** (DSRK — *Domain-Specific Root Key*) przeznaczony do stosowania w danej domenie (grupie komputerów). Klucze potomne generowane na podstawie DSRK są nazywane **kluczami domenowymi specjalnego przeznaczenia** (DSUSRK — *Domain-Specific Usage-Specific Root Key*).

W trakcie wymiany informacji w protokole EAP przedstawionych może zostać kilka tożsamości jednostki końcowej i serwera. Komunikacja ta jest wyróżniana za pomocą specjalnego identyfikatora sesji. Po zakończeniu uwierzytelnienia, w którym realizowane jest zadanie wyznaczenia kluczy, wartości MSK, EMSK, identyfikator (lub identyfikatory) jednostki końcowej, identyfikator (lub identyfikatory) serwera oraz identyfikator sesji są udostępniane niższym warstwom stosu (w analogiczny sposób może zostać przekazany wektor inicjujący, którego użycie jest jednak obecnie uznawane za niewłaściwe). Każdemu kluczowi odpowiada zazwyczaj pewien czas ważności (zalecaną wartością czasu ważności jest 8 godzin), po którego upływie rozpoczyna się procedura ponownego uwierzytelnienia EAP. Szczegółowe omówienie platformy zarządzania kluczami EAP oraz analiza mechanizmów zabezpieczeń znajdują się w dokumencie [RFC5247].

18.7.2. Protokół ponownego uwierzytelnienia (ERP)

Po udanym zakończeniu operacji uwierzytelnienia EAP kolejnym wyzwaniem jest zapewnienie niewielkiej zwłoki czasowej w ponownym uwierzytelnianiu, realizowanym już w trakcie trwania połączenia (np. podczas przenoszenia stacji końcowej z obszaru działania jednego punktu dostępowego do obszaru innego punktu). Zadanie to wykonuje **protokół ponownego uwierzytelnienia EAP** (ERP — *EAP Re-authentication Protocol*) [RFC5296], który działa niezależnie od wybranej wcześniej metody EAP. Stacje końcowe i serwery EAP obsługujące protokół ERP są nazywane jednostkami końcowymi ER lub serwerami ER. W pracy mechanizmu ERP wykorzystywany jest klucz główny ponownego uwierzytelnienia (rRK) generowany na podstawie klucza DSRK (lub EMSK, choć w dokumencie [RFC5295] znajduje się zalecenie, by nie stosować tego klucza) wraz z kluczem zapewnienia integralności (rIK) uzyskiwanym na podstawie klucza rRK (przeznaczonym do tego, aby stacje mogły potwierdzić, że znają klucz rRK).

Protokół ERP wykonuje swoje zadanie w czasie jednej dwukierunkowej wymiany informacji, dzięki czemu istotnie skraca opóźnienie związane z ponownym uwierzytelnieniem. Operacja rozpoczyna się do typowej wymiany komunikatów EAP przy założeniu, że stacja pracuje we własnej domenie. Wygenerowany klucz MSK jest dostarczany w standardowy sposób do urządzenia uwierzytelniającego oraz jednostki końcowej. W tym samym czasie wyznaczane są wartości rIK i rRK, które są znane jedynie jednostce końcowej i serwerowi EAP. Mogą one być wykorzystywane we własnej domenie wraz z kluczami rMSK przeznaczonymi dla każdego z urządzeń uwierzytelniających. Gdy jednostka końcowa ER zmienia domenę, zaczyna korzystać z innych wartości kluczy (DS-rIK oraz DS-rRK, które są kluczami typu DSUSRK). Domena serwera ER jest zapisana w obszarze TLV każdego komunikatu ERP, co pozwala jednostkom końcowym na ustalenie domeny serwera, z którym się komunikują. Szczegółowy sposób działania protokołu jest opisany w dokumencie [RFC5296].

18.7.3. Protokół przenoszenia danych uwierzytelniających w dostępie do sieci (PANA)

Mimo że od lat wykorzystuje się mechanizmy EAP, 802.1X i PPP do zapewnienia uwierzytelniania jednostek klienckich (a w niektórych przypadkach również sieci), protokoły te są w pewnym stopniu zależne od rodzaju łącza. Systemy EAP są implementowane przede wszystkim w łączach dedykowanych określonym użytkownikom, rozwiązania 802.1X znajdują zastosowanie jedynie w sieciach IEEE 802, natomiast protokół PPP sprawdza się w połączeniach typu punkt-punkt. Aby wyeliminować tę niedogodność, opracowano **protokół przenoszenia danych uwierzytelniających w dostępie do sieci** (PANA — *Protocol for Carrying Authentication for Network Access*). Jego specyfikacja jest zawarta w dokumentach [RFC5191], [RFC5193] oraz [RFC6345], a podstawowe wymagania dotyczące sposobu funkcjonowania zostały zdefiniowane w opracowaniach [RFC4058] i [RFC4016]. Protokół PANA pełni rolę dolnej warstwy stosu EAP, czyli służy do przenoszenia informacji EAP. Wykorzystuje do tego celu protokoły UDP/IP (oraz port 716). Zgodnie z założeniami jego stosowanie nie jest ograniczone do żadnego rodzaju łącza, a także nie wymaga pracy w modelu punkt-punkt. W rezultacie protokół PANA pozwala na implementowanie dowolnych metod EAP w każdej technologii warstwy łącza danych.

W środowisku PANA wyróżnione zostały trzy elementy funkcjonalne: klient PANA (PaC — *PANA Client*), agent uwierzytelniania PANA (PAA — *PANA Authentication Agent*) oraz element przekazujący PANA (PRE — *PANA Relay Element*). W typowej instalacji używane są dodatkowo serwer uwierzytelniania (AS — *Authentication Server*) oraz punkt egzekwowania polityki (EP — *Enforcement Point*). Serwera AS może być tradycyjnym serwerem AAA pracującym zgodnie z protokołem RADIUS lub Diameter. Moduł PAA jest odpowiedzialny za przekazywanie danych uwierzytelniających z jednostki PaC do serwera AS oraz za pobieranie z modułu EP informacji o konfiguracji łącza dostępowego, po uzyskaniu lub odmowie prawa do korzystania z sieci. Niektóre z wymienionych komponentów mogą funkcjonować w ramach jednego urządzenia. Przykładami są chociażby moduł PaC i jednostka końcowa EAP oraz urządzenie uwierzytelniające EAP i moduł PAA. Jeśli bezpośrednia komunikacji między komponentami PaC i PAA nie jest możliwa, wspomaga ją element PRE.

Protokół PANA definiuje zbiór komunikatów typu żądanie-odpowiedź wraz z rozszerzalnym zbiorem par atrybut-wartość standaryzowanym przez organizację IANA [IPANA]. Podstawowym kontenerem transmisyjnym są komunikaty EAP wysyłane w formie datagramów UDP/IP w ramach sesji PANA. Sama sesja PANA składa się z czterech faz: uwierzytelnienie (autoryzacja), korzystanie z dostępu, ponowne uwierzytelnienie, zakończenie. Faza ponownego uwierzytelnienia jest w istocie elementem fazy korzystania z dostępu, w trakcie której czas ważności sesji jest okresowo przedłużany przez ponawianie operacji uwierzytelniania EAP. Faza zakończenia jest realizowana jawnie lub w wyniku wygaśnięcia sesji (spowodowanego upływem czasu ważności sesji lub wykryciem niedostępności urządzeń). Sesje PANA są identyfikowane za pomocą 32-bitowej wartości dołączanej do każdej wiadomości.

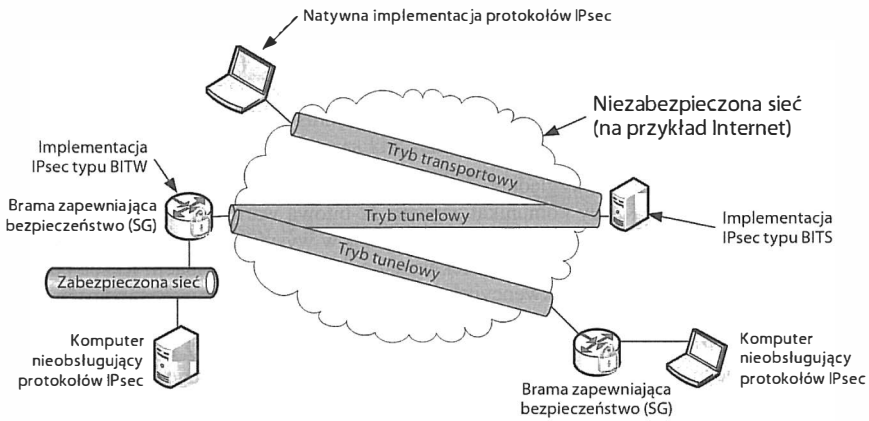
Mechanizm PANA uwzględnia dodatkowo pewną formę niezawodnego protokołu transportowego. Każdy komunikat zawiera 32-bitową wartość numeru sekwencyjnego. Nadawca informacji monitoruje kolejne numery w wysyłanych wiadomościach. Z kolei odbiorca śledzi kolejne numery w sekwencji odbieranych informacji. Każda odpowiedź zawiera ten sam numer sekwencyjny, który został zapisany we wcześniejszym żądaniu. Początkowe wartości są generowane w sposób losowy przez nadawcę komunikatu (tj. PaC lub PAA). W protokole PANA zaimplementowano również retransmisję czasową. Jest to jednak niezbyt efektywny mechanizm — działa na zasadzie algorytmu start-stop i nie wykorzystuje adaptacyjnego zegara retransmisji ani przebudowywania pakietów. Obsługuje natomiast mechanizm wykładniczego odczekiwania w przypadku utraty większej liczby pakietów.

18.8. Bezpieczeństwo warstwy 3. (IPsec)

IPsec jest nazwą architektury i kolekcją standardów, które zapewniają uwierzytelnienie źródła danych, integralność i poufność informacji oraz kontrolę dostępu na poziomie warstwy sieci w protokołach IPv4 oraz IPv6 [RFC4301], w tym również Mobile IPv6 [RFC4877]. Mechanizm IPsec odpowiada także za wymianę kluczy kryptograficznych między stronami komunikacji, wybór zestawów algorytmów kryptograficznych oraz metod sygnalizacji stosowania kompresji. Stroną komunikacji może być pojedyncza stacja lub **brama zapewniająca bezpieczeństwo** transmisji (SG — *security gateway*), która wyznacza granicę między zabezpieczoną i niezabezpieczoną częścią sieci. Rozwiązania

IPsec są stosowane w aplikacjach zapewniających zdalny dostęp do korporacyjnej sieci LAN (tworzenie połączeń VPN), przy łączeniu w bezpieczny sposób różnych obszarów sieci firmowej przez Internet lub też w zabezpieczaniu komunikacji między komputerami bądź routerami realizującymi zadania w sposób charakterystyczny dla stacji końcowych (np. podczas przesyłania informacji o routingu). Mechanizm IPsec jest najczęściej wybieranym rozwiązaniem do zabezpieczania nowo opracowywanych protokołów [RFC5406].

Na rysunku 18.9 zostało przedstawionych kilka sposobów wykorzystania rozwiązań IPsec. Po stronie komputera kod IPsec może zostać zintegrowany ze stosem IP lub funkcjonować jako sterownik „poniżej" istniejącego stosu sieciowego (ten sposób implementacji jest określany jako „guz na stosie" [BITS — *Bump In The Stack*]). Alternatywnie obsługa mechanizmu IPsec może zostać zaimplementowana w module SG, co niekiedy nazywa się „guzem na przewodzie" (BITW — *Bump In The Wire*). W rozwiązaniu BITW brama musi realizować funkcje stacji końcowej oraz urządzenia SG — urządzeniami tego typu trzeba zazwyczaj zarządzać zdalnie. Z tego też powodu konieczne jest implementowanie w routerach protokołów warstw aplikacji i transportowej (bez uruchamiania opisanych funkcji routery byłyby urządzeniami warstwy 3. [więcej informacji na ten temat znajduje się w rozdziale 1.]). Przeanalizujmy zatem działanie mechanizmu IPsec w typowej komunikacji między dwoma jednostkami (choć obsługuje on również multiemisję).

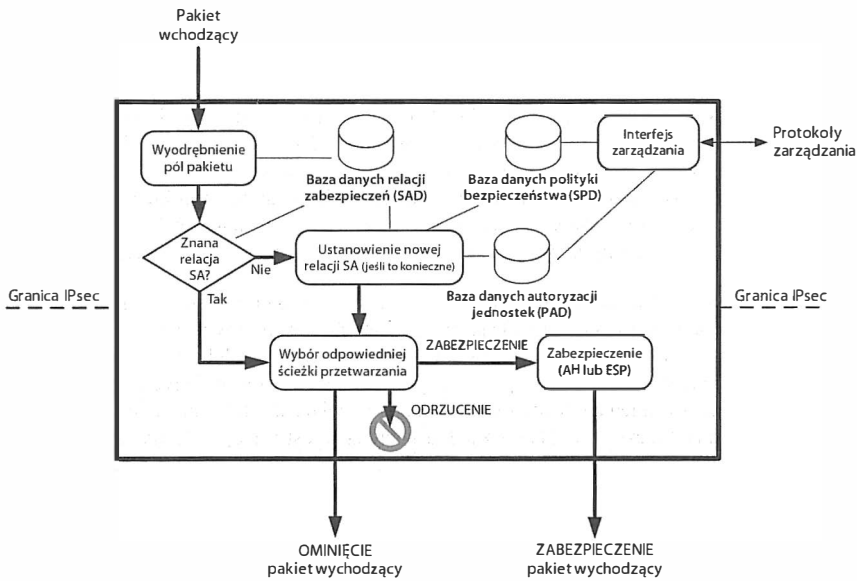


Rysunek 18.9. *Mechanizm IPsec znajduje zastosowanie w zabezpieczaniu komunikacji między dwoma komputerami, między komputerem i bramą oraz między dwoma bramami. Realizuje także zadania multiemisji i pozwala na pracę w środowiskach z urządzeniami mobilnymi*

W działaniu protokołu IPsec wyróżnia się dwie następujące po sobie fazy — fazę nawiązania połączenia, w której wymieniane są dane uwierzytelniające i ustanawiane są **relacje zabezpieczeń** (SA — *Security Association*), oraz fazę wymian danych, w której do ochrony informacji wykorzystuje się różne rodzaje enkapsulacji (protokoły **nagłówka uwierzytelniającego** [AH — *Authentication Header*] lub **zabezpieczenia pola danych** [ESP — *Encapsulating Security Payload*]) oraz różne tryby pracy (**tunelowy** lub **transportowy**). Każdy z komponentów środowiska IPsec wymaga użycia odpowiedniego zestawu algorytmów kryptograficznych, a samo środowisko IPsec może współdziałać z wieloma różnymi zestawami kryptograficznymi. Pełna implementacja mechanizmu IPsec obejmuje protokół ustanawiania relacji SA, protokół AH (opcjonalnie),

protokół ESP oraz zbiór zestawów algorytmów kryptograficznych, parametrów konfiguracyjnych oraz narzędzi wspomagających ustanawianie połączeń. Omówienie dotychczasowego rozwoju środowiska IPsec oraz bieżące specyfikacje wszystkich komponentów znajdują się w dokumencie [RFC6071].

Mimo że rozwiązania IPsec są domyślnie uwzględniane w wielu systemach operacyjnych (obsługa protokołów IPsec jest warunkiem implementacji stosu IPv6), działanie mechanizmu jest bardzo selektywne i odnosi się jedynie do pewnej grupy pakietów zdefiniowanych przez administratora w ramach obowiązującej polityki bezpieczeństwa. Ustawienia polityki są przechowywane w **bazie danych polityki bezpieczeństwa** (SPD — *Security Policy Database*) stanowiącej element środowiska IPsec. Działanie protokołów IPsec jest dodatkowo zależne od dwóch innych baz danych — **bazy danych relacji zabezpieczeń** (SAD — *Security Association Database*) oraz **bazy danych autoryzacji jednostek sieciowych** (PAD — *Peer Authorization Database*). Ich zawartość decyduje o sposobie przetwarzania pakietów. Stosowny przykład został przedstawiony na rysunku 18.10.



Rysunek 18.10. *W przypadku bramy sieciowej przetwarzanie pakietów IPsec odbywa się w warstwie 3. w module logicznym oddzielającym sieć zabezpieczoną od niezabezpieczonej. Baza danych polityki bezpieczeństwa określa sposób postępowania z pakietami: ominięcie modułu, odrzucenie lub zabezpieczenie. Zabezpieczenie wymaga wdrożenia lub weryfikacji integralności oraz poufności danych. Administrator konfiguruje bazę SPD tak, aby uzyskać zamierzony poziom bezpieczeństwa transmisji*

Schemat zamieszczony na rysunku 18.10 w nieco uproszczony sposób opisuje działanie bramy SG. Zgodnie z nim urządzenie sprawdza wybrane pola nadchodzących pakietów (**wyróżniki ruchu**) i ustala, czy należy je przetwarzać jako ruch IPsec oraz czy istnieją związane z nimi wartości SA. Jeśli tak, dalsze postępowanie nie jest skomplikowane. Sprowadza się do wykonania operacji wymaganych przez protokoły ESP lub AH, opisane

w punktach 18.8.2 i 18.8.3. W przeciwnym przypadku urządzenie ustala, jakiego rodzaju relacja SA powinna zostać ustanowiona (jeśli w ogóle jakakolwiek jest potrzebna), a następnie zapisuje w bazie SAD informacje na jej temat. Jeżeli trzeba ustanowić nową relację SA, najłatwiejszym sposobem realizacji zadania jest użycie jednego z protokołów automatycznego wyznaczania kluczy. Choć specyfikacja IPsec dopuszcza ręczne wprowadzanie kluczy, metoda ta nie nadaje się do zastosowania na szerszą skalę i jest podatna na błędy. Z tego względu do nawiązywania relacji SA zaleca się stosowanie protokołów wyznaczania kluczy. Najnowsza wersja jednego z tego rodzaju protokołów została opisana w kolejnym punkcie.

18.8.1. Protokół wymiany kluczy w Internecie (IKEv2)

Pierwszy etap w ustanawianiu połączenia IPsec polega na zdefiniowaniu relacji SA. Relacja SA jest nieskomplikowanym (jednokierunkowym), uwierzytelnionym wiązaniem między dwiema komunikującymi się stronami lub nadawcą i wieloma odbiorcami (w multiemisji). W przeważającej liczbie przypadków komunikacja ma charakter dwukierunkowy i jest realizowana jedynie między dwoma urządzeniami. Wówczas zastosowanie mechanizmu IPsec wymaga utworzenia dwóch relacji SA. Zadanie to wykonuje specjalny protokół nazywany **protokołem wymiany kluczy internetowych** (IKE — *Internet Key Exchange*). Bieżąca wersja protokołu jest oznaczana jako IKEv2 [RFC5996], choć często używa się po prostu określenia IKE. Protokół IKE jest jednym z najbardziej skomplikowanych elementów mechanizmu IPsec. Dlatego po zrozumieniu zasad jego funkcjonowania pozostałe operacje wydają się bardzo łatwe. W dalszej części omówione zostaną jednak tylko najważniejsze z realizowanych zadań. Szczegółowy opis poszczególnych komponentów protokołu (np. różnorodnych zestawów algorytmów kryptograficznych lub parametrów konfiguracyjnych) został zamieszczony w dokumencie [RFC5996].

Relacja SA jest ustanawiana w czasie wymiany pary komunikatów żądanie-odpowiedź. W żądaniu zawarte są propozycje następujących parametrów: algorytmu szyfrowania, algorytmu ochrony integralności danych, grupy Diffiego-Hellmana oraz rodziny PRF zapewniającej uzyskanie przypadkowo wyglądającego zbioru danych wyjściowych niezależnie od strumienia bitów wejściowych. W rozwiązaniu IKE rodzina PRF odpowiada za generowanie kluczy sesji. Na początek ustanawiana jest relacja SA dla samego protokołu IKE (o nazwie IKE_SA), a następnie możliwe jest określenie parametrów SA dla protokołów AH lub ESP (nazywanych CHILD_SA). W ramach opisywanego mechanizmu możliwe jest również negocjowanie **kompresji pola danych IP** (IPComp — *IP Payload Compression*) [RFC3173], ponieważ zastosowanie kompresji w innych warstwach (po wykonaniu szyfrowania) okazuje się nieefektywne. Szczegółowe omówienie protokołów AH i ESP znajduje się w punktach 18.8.2 i 18.8.3.

Działanie mechanizmu IKE bazuje na przesyłaniu par komunikatów nazywanych **wymianami** (*exchange*). Komunikaty te są przekazywane pomiędzy **inicjatorem** (*initiator*) i **odpowiadającym** (*responder*). Pierwsze dwie wymiany, nazywane `IKE_SA_INIT` oraz `IKE_AUTH`, ustanawiają relację `IKE_SA` oraz pojedyncze powiązanie `CHILD_SA`. Kolejne wymiany typu `CREATE_CHILD_SA` umożliwiają ustanawianie dodatkowych relacji `CHILD_SA`. Natomiast wymiany typu `INFORMATIONAL` pozwalają na wprowadzanie zmian lub pozyskiwanie informacji na temat obowiązujących powiązań SA. W większości przypadków wystarczające okazują się pojedyncze wymiany `IKE_SA_INIT` oraz `IKE_AUTH`

(wymagające przekazania razem czterech komunikatów). Komunikaty transmitowane w ramach poszczególnych wymian składają się z **pól danych** (*payloads*) wyróżnianych za pomocą identyfikatorów typu. W jednym komunikacie może występować wiele pól danych. Wynikowy pakiet IP często okazuje się więc relatywnie duży i wymaga fragmentacji.

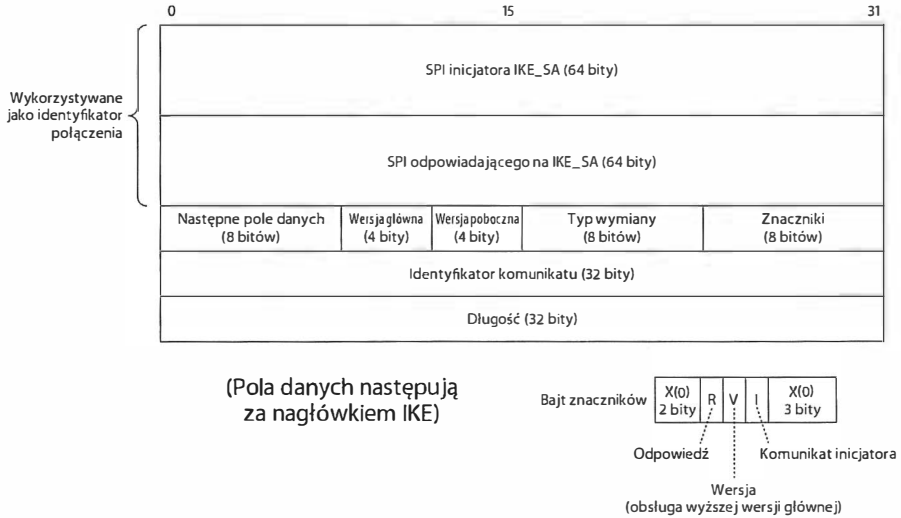
Komunikaty IKE są przesyłane w datagramach UDP z wykorzystaniem portów 500 i 4500. Jednak z uwagi na możliwość wykonania operacji NAT podczas przekazywania ruchu przez sieć (w wyniku której port ulega zmianie), odbiorca IKE musi być przygotowany na przetwarzanie datagramów pochodzących z dowolnego portu. Port 4500 jest zarezerwowany do obsługi danych IKE i ESP transportowanych w protokole UDP [RFC3948]. Komunikaty IKE odbierane na porcie 4500 muszą mieć ustawione pierwsze cztery bajty danych na zera, aby można było odróżnić je od innych pakietów (np. ESP lub WESP).

W przypadku utraty pakietów inicjator IKE retransmituje komunikaty w odpowiednich interwałach. Odpowiadający wykonują retransmisję jedynie w reakcji na nadchodzące żądanie. Okres ponawiania transmisji jest opisany funkcją wykładniczą, ale całkowita liczba retransmisji nie została sprecyzowana. Zarówno inicjator, jak i odpowiadający rejestrują ostatnio wysłane komunikaty oraz odpowiadające im numery sekwencyjne. Zadaniem numerów sekwencyjnych jest powiązanie żądań z odpowiedziami oraz obsługa retransmisji. Użycie numerów sekwencyjnych dodatkowo uzupełnia protokół IKE o funkcję obsługi okien transmisyjnych o maksymalnym rozmiarze ustalonym przez odpowiadającego podczas pierwszej wymiany SA (rozmiar okna może być zmieniony również na późniejszym etapie komunikacji). Maksymalny rozmiar okna wyznacza całkowitą liczbę niepotwierdzonych żądań.

18.8.1.1. Formaty komunikatów IKEv2

Komunikat IKE składa się z nagłówka, po którym następuje zero lub więcej **pól danych IKE**. Struktura nagłówka została pokazana na rysunku 18.11.

Widoczne na rysunku 18.11 pole SPI (**indeks parametru zabezpieczeń** — *Security Parameter Index*) przechowuje 64-bitową liczbę identyfikującą określoną relację IKE_SA (w innych protokołach IPsec wykorzystywana jest 32-bitowa wartość SPI). Zarówno inicjator, jak i odpowiadający dysponują wartościami SA powiązanymi ze stacjami znajdującymi się po drugiej stronie połączenia. Zatem każda z jednostek operuje wartościami SPI, które wraz z adresami IP stanowią doskonały identyfikator połączenia. Pole *Następne pole danych* zostało omówione w dalszej części tego punktu. Pola *Wersja główna* i *Wersja poboczna* przechowują odpowiednio wartości 2 i 0 (w przypadku opisywanej wersji protokołu IKE). Wartość wersji głównej jest zmieniana wówczas, gdy nie można zagwarantować współdziałania protokołów o różnych wersjach. Pole *Typ wymiany* określa rodzaj wymiany danych, której elementem jest dany komunikat. Dopuszczalne wartości to `IKE_SA_INIT` (34), `IKE_AUTH` (35), `CREATE_CHILD_SA` (36), `INFORMATIONAL` (37) oraz `IKE_SESSION_RESUME` (38; patrz [RFC5723]). Pozostałe wartości są wyłączone z użycia. Zakres od 240 do 255 jest natomiast zarezerwowany do użytku prywatnego. W polu *Znaczniki* istotne są trzy wartości bitowe (indeksowane od 0 w kierunku od prawej do lewej strony): `I` (komunikat inicjatora, bit 3.), `V` (wersja, bit 4.) oraz `R` (odpowiedź, bit 5.). Pole `I` jest ustawiane przez jednostkę inicjującą wymianę danych



Rysunek 18.11. *Nagłówek IKEv2. Wszystkie wiadomości IKE zawierają nagłówek, po którym następuje zero lub więcej pól danych. W protokole IKE wykorzystywane są 64-bitowe wartości SPI. Pole Typ wymiany wskazuje cel wymiany komunikatów oraz określa rodzaj pól danych spodziewanych w komunikacie. Identyfikator komunikatu kojarzy żądania z odpowiedziami i zapobiega atakom z odtworzeniem pakietów*

oraz zerowane przez odbiorcę w odpowiedzi. Bit `V` informuje, że nadawca komunikatu obsługuje protokół o wersji wyższej niż podana w polu *Wersja główna*. Ustawienie bitu `R` stanowi informację, że komunikat jest odpowiedzią na wcześniejszą wiadomość o tym samym identyfikatorze.

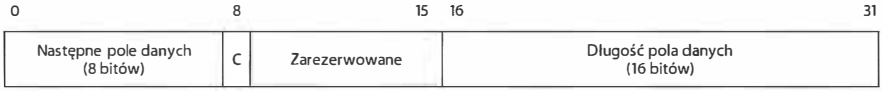
Pole *Identyfikator komunikatu* w protokole IKE pełni podobną rolę jak *Numer Sekwencyjny* w mechanizmie TCP (patrz rysunek 12.3 w rozdziale 12.). Jedyna różnica polega na tym, że wyznaczanie identyfikatora rozpoczyna się od zera w przypadku inicjatora oraz od 1 w jednostce odpowiadającej. Wartość licznika jest zwiększana o jeden w każdej transmisji. Ponieważ w odpowiedziach zapisywane są takie same wartości identyfikatora, jakie są przesyłane w żądaniach, **detekcja powtórzeń** (*replay detection*) nie stanowi problemu. Wiadomości o przedawnionych identyfikatorach nie są po prostu przetwarzane. Przewinięcie licznika identyfikatorów (bardzo mało prawdopodobne, choć możliwe w przypadku przesłania ponad 4 miliardów komunikatów IKE) jest obsługiwane przez wymianę `IKE_SA_INIT`.

Pozostałe pola (*Następne pole danych* oraz *Długość*) opisują zawartość komunikatu IKE. Każdy komunikat może przenosić dowolną liczbę pól danych (może również nie przenosić żadnych pól danych). Każde z pól danych ma niezależną strukturę. Wartość *Długość* odzwierciedla rozmiar nagłówka oraz wszystkich uwzględnionych w komunikacie pól danych. Wartość *Następne pole danych* zawiera informację o typie następującego po nim pola danych (wartość 0 odpowiada brakowi pola danych). Aktualnie zdefiniowanych jest 16 typów wartości. Zostały one przedstawione w tabeli 18.2. Bieżąca lista typów jest publikowana w dokumencie [IKEPARAMS] i obejmuje wszystkie identyfikatory typów stosowane w protokole IKEv2.

Tabela 18.2. *Typy pól danych protokołu IKEv2. Wartość 0 oznacza brak pola danych*

Wartość	Notacja	Przeznaczenie	Wartość	Notacja	Przeznaczenie
33	SA	Relacja zabezpieczeń	41	N	Powiadomienie
34	KE	Wymiana kluczy	42	D	Usunięcie
35	IDi	Identyfikacja (inicjatora)	43	V	Identyfikator dostawcy
36	IDr	Identyfikacja (odpowiadającego)	44	TSi	Selektor ruchu (inicjatora)
37	CERT	Certyfikat	45	TSr	Selektor ruchu (odpowiadającego)
38	CERTREQ	Żądanie certyfikatu (wyznacza kotwicę zaufania)	46	SK { }	Zaszyfrowane i uwierzytelnione dane (przechowuje inne pola danych)
39	AUTH	Uwierzytelnienie	47	CP	Konfiguracja
40	Ni, Nr	Wartości jednorazowe (inicjatora, odpowiadającego)	48	EAP	Protokół EAP

Wartości w przedziałach od 1 do 32 oraz od 49 do 255 są zarezerwowane. Przedział od 128 do 255 jest przeznaczony do użytku prywatnego. Każde pole danych IKE rozpoczyna się od **ogólnego nagłówka pola danych** IKE, którego struktura została przedstawiona na rysunku 18.12.



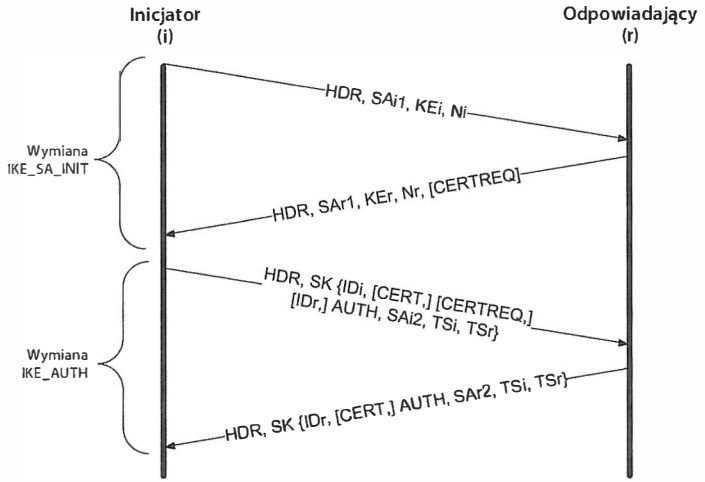
Rysunek 18.12. *Ogólny nagłówek pola danych IKEv2 przesyłany na początku każdego pola danych*

Ogólny nagłówek pola danych ma stały rozmiar 32 bitów, a zawarte w nim wartości *Następne pole danych* oraz *Długość pola danych* definiują łańcuch bloków o zmiennym rozmiarze (z których każdy może mieć maksymalny rozmiar 65 535 bajtów, włącznie z nagłówkiem) zawartych w pojedynczym komunikacie IKE. Każdemu polu danych odpowiada odrębny zbiór nagłówków specjalnych. Bit C (dane krytyczne) wskazuje, że określone pole danych (nie to, które jest opisane za pomocą wartości *Następne pole danych*) jest uznawane za „krytyczne" do poprawnego przeprowadzania wymiany IKE. Odbiorcy krytycznych pól danych, którzy nie mają możliwości odpowiedniego zinterpretowania zawartości (wskazanej w sekcji *Następne pole danych* poprzedniego pola danych lub w sekcji *Następne pole danych* nagłówka IKE), mają obowiązek przerwać wymianę IKE. Dzięki temu można dodawać do protokołu funkcje, które nie będą poprawnie obsługiwane we wszystkich implementacjach.

18.8.1.2. Wymiana IKE_SA_INIT

W dokładniejszym zrozumieniu zasad działania protokołu IKE bardzo pomocne jest poznanie sposobu wykonywania operacji IKE_SA_INIT, czyli pierwszej z dwóch wymian (IKE_SA_INIT i IKE_AUTH) inicjujących komunikację i przedstawionych na rysunku

18.13. Wymiany inicjujące są nieformalnie nazywane fazą 1. komunikacji IKE. Pozostałe wymiany (CREATE_CHILD_SA oraz INFORMATIONAL) mogą zostać zapoczątkowane przez dowolną stronę dopiero po zakończeniu etapu wymian inicjujących. Ponadto dane przekazywane w ramach kolejnych wymian zawsze podlegają szyfrowaniu i ochronie integralności zgodnie z parametrami wyznaczonymi w 1. fazie komunikacji.



Rysunek 18.13. *Wymiany IKE_SA_INIT oraz IKE_AUTH przekazują pola danych niezbędne do ustanowienia dwóch pierwszych relacji zabezpieczeń (IKE_SA oraz CHILD_SA). Mogą obejmować żądania certyfikatów oraz same certyfikaty, a także pola danych powiadomień i konfiguracji (które nie zostały tutaj pokazane)*

Zgodnie z rysunkiem 18.13, wymiana IKE_SA_INIT służy do negocjowania zestawu kryptograficznego, przekazywania wartości jednorazowych oraz uzgadniania kluczy DH. Może również przenosić dodatkowe informacje, zależnie od konkretnej implementacji oraz sposobu wdrożenia systemu. Operacja rozpoczyna się od wysłania przez inicjatora komunikatu IKE zawierającego wykaz obsługiwanych zestawów kryptograficznych, informacje dotyczące algorytmu DH oraz wartości jednorazowe. Transport wymienionych danych wymaga zdefiniowania trzech pól danych (*SA*, *KE* oraz *Ni*). Szczegółowy opis poszczególnych pól znajduje się w sekcji 3. dokumentu [RFC5996]. Część z nich została również omówiona w podpunkcie 18.8.1.3. Trzeba jednak pamiętać, że w różnych implementacjach możliwe jest dołączanie dodatkowych pól danych. Brak odpowiedzi na ten komunikat jest powodem rozpoczęcia retransmisji po stronie inicjatora.

Wraz z odbiorem pierwszej wiadomości odpowiadający uzyskuje informację o tym, że inicjator żąda rozpoczęcia transakcji IKE, przedstawiając jednocześnie listę obsługiwanych zestawów kryptograficznych oraz parametrów konfiguracyjnych. Odpowiadający wybiera akceptowalne zestawy i zgłasza ten fakt inicjatorowi za pomocą pola danych *SAr1* (patrz podpunkt 18.8.1.3). Dodatkowo przekazuje wartości parametrów do procedury uzgadniania kluczy DH (w polu *KEr*), wartości jednorazowe (w polu *Nr*) oraz opcjonalne żądanie dostarczenia certyfikatu inicjatora (w polu danych *CERTREQ*). Pole *CERTREQ* obejmuje również wykaz urzędów CA, które odbierający uznaje za odpowiednie do weryfikacji certyfikatów używanych ewentualnie w kolejnych wymianach (przedstawia

w ten sposób własną kotwicę zaufania). Komunikat zawierający nagłówek IKE oraz wszystkie pola danych jest przesyłany w formie odpowiedzi do inicjatora, co kończy wymianę IKE_SA_INIT. W niektórych implementacjach dołączane są również dodatkowe pola (np. powiadomienia lub konfiguracji; więcej informacji na ich temat znajduje się w podpunkcie 18.8.1.5). Aby w pełni zrozumieć funkcje wymiany IKE_SA_INIT, warto wnikliwiej przeanalizować najważniejsze z jej pól danych: *SA*, *KE*, *Ni* oraz *Nr*.

18.8.1.3. Pola danych i propozycje relacji zabezpieczeń (SA)

Pola danych komunikatów SA zawierają wartość SPI oraz zestaw **propozycji** (często składający się z jednej tylko propozycji), które są opisane za pomocą dość złożonych struktur. Każda propozycja ma odpowiedni numer oraz identyfikator protokołu IPsec. Wspomniane identyfikatory odpowiadają następującym protokołom stosu IPsec: IKE, AH lub ESP (więcej informacji na ich temat zostało przedstawionych w punktach 18.8.2 oraz 18.8.3). Kilka struktur o tym samym numerze propozycji jest uznawanych za kilka części tej samej propozycji (oraz tego samego protokołu). Struktury o różnych numerach są uznawane za różne propozycje (lub propozycje różnych protokołów).

Każda struktura propozycji (protokołu) zawiera jedną strukturę **transformaty** (lub wiele tego typu struktur) opisującą algorytm przeznaczony do wykorzystania ze wskazanym protokołem. Protokołowi AH towarzyszy zazwyczaj pojedyncza transformata (algorytm sprawdzania integralności), protokół ESP wykorzystuje dwie transformaty (algorytmy sprawdzania integralności oraz szyfrowania), z kolei protokół IKE definiuje cztery transformaty (odpowiadające za przynależność do grupy DH, wyznaczanie rodziny PRF, sprawdzanie integralności oraz szyfrowanie). Algorytmy łączące zadania szyfrowania i weryfikacji integralności są przedstawiane jedynie jako mechanizmy szyfrujące, bez odrębnego definiowania funkcji związanych z zapewnianiem integralności danych. Jeden ze specjalnych numerów transformaty (odpowiadający wartości logicznej) informuje o tym, czy numery sekwencyjne relacji SA (tj. numery stosowane w protokołach AH i ESP) powinny być wyznaczane jako wartości 32-bitowe czy 64-bitowe.

W przypadku zdefiniowania wielu transformat tego samego typu dana propozycja jest sumą wszystkich transformat (o ile jest do dopuszczalne). W przypadku przedstawienia wielu transformat o różnych typach propozycją jest ich część wspólna. Pojedyncza transformata może się składać z dowolnej liczby **atrybutów** (może również nie mieć żadnych atrybutów). Atrybuty są niezbędne w przypadkach, w których daną transformatę można wykonać na więcej niż jeden sposób (jeśli np. do szyfrowania można użyć kluczy o różnych długościach, w opisie danej transformaty zostanie uwzględniony atrybut z informacją o długości klucza obowiązującej w danej propozycji). Większość transformat nie wymaga definiowania atrybutów, ale jedna z częściej stosowanych transformat — szyfrowanie AES — jest od niego zależna.

18.8.1.4. Pola danych wymiany kluczy (KE) oraz wartości jednorazowych (Ni, Nr)

Poza polami danych *SA* komunikaty IKE_SA_INIT zawierają często pole *KE* (wymiany kluczy) oraz pola *Ni* i *Nr* (wartości jednorazowych; zapisywanych czasami jako *No*). W polu *KE* zapisane są numery grupy DH oraz dane potrzebne do wymiany kluczy

odpowiadające publicznym wartościom liczbowym wykorzystywanym do utworzenia tymczasowego klucza DH (początkowego wspólnego hasła). Numer grupy DH określa grupę, z której pochodzą wyznaczone wartości publiczne. Pole danych wartości jednorazowej zawiera natomiast ostatnio wygenerowaną wartość jednorazową o długości od 16 do 256 bajtów. Jest ona wykorzystywana do utworzenia niepowtarzalnego klucza i eliminowania tym samym ataków z odtwarzaniem danych.

Po zakończeniu wymiany DH każda ze stron może wygenerować wartość SKEYSEED, która jest używana do wyznaczania kolejnych kluczy w ramach wymiany IKE_SA (o ile do powoływania kluczy nie została wykorzystana odpowiednia metoda protokołu EAP; więcej informacji na ten temat znajduje się w podpunkcie 18.8.1.9). Dodatkowo każda ze stron dysponuje siedmioma tajnymi wartościami: SK_d, SK_ai, SK_ar, SK_ei, SK_er, SK_pi oraz SK_pr, obliczanymi zgodnie z poniższą zależnością.

```
SKEYSEED = prf(Ni | Nr, g^ir)
{SK_d|SK_ai|SK_ar|SK_ei|SK_er|SK_pi|SK_pr} = prf+ (SKEYSEED, Ni|Nr|SPIi|SPIr)
```

Symbol | oznacza operację konkatenacji. Kaskadowe wywołanie funkcji PRF odpowiada operacji prf+ (K, S) = T1 | T2 | ..., gdzie T1 = prf(K, S|0x01), T2 = prf(K, T1|S|0x03), T3 = prf(K, T2|S|0x03), T4 = prf(L, T3|S|0x04), ... Wartość g^ir jest wspólnym hasłem określonym w czasie wymiany DH. Parametry *Ni* oraz *Nr* odpowiadają wartościom jednorazowym (zaczerpniętym z nagłówków pól danych). Warto też wspomnieć, że w poszczególnych kierunkach relacjom SA odpowiadają różne klucze (co wyjaśnia, dlaczego aż tyle jest ich potrzebnych). Klucz SK_d służy do wyznaczania kluczy relacji CHILD_SA. Klucze SK_a oraz SK_e są stosowane w uwierzytelnianiu oraz szyfrowaniu. Klucz SK_p odpowiada za generowanie pól danych *AUTH* w wymianie IKE_AUTH.

18.8.1.5. Pola danych powiadomień (N) oraz konfiguracji (CP)

Pole danych oznaczane symbolem *N* służy do przenoszenia powiadomień. Choć nie zostało przedstawione na rysunku 18.13, jest obecne w omawianych dalej przykładach. Służy często do przenoszenia komunikatów o błędach oraz informacji o zdolności jednostek do realizacji różnych wymian IKE. Zawiera pole SPI o zmiennej długości oraz 16-bitowe pole identyfikujące rodzaj powiadomienia [IKEPARAMS]. Wartości niższe niż 8192 opisują standardowe błędy, natomiast wyższe niż 16383 przekazują informacje statusowe. Przykładowo podczas tworzenia relacji SA właściwej dla trybu transportowego zamiast domyślnego trybu tunelowego pole danych powiadomień jest wykorzystywane do przenoszenia wartości 16391 (USE_TRANSPORT_MODE). Jeśli jedna ze stron obsługuje kompresję danych IP [RFC3173], sygnalizuje ten fakt za pomocą wartości 16387 (IPCOMP_SUPPORTED). Analogicznie stacje obsługujące mechanizm efektywnej kompresji nagłówków (ROHC — *Robust Header Compression*) [RFC5857] przesyłają wartość 16416 (ROHC_SUPPORTED) wraz z parametrami ROHC niezbędnymi do ustanawiania relacji SA oznaczanej jako ROHCo-IPsec. Z kolei chęć wykorzystania komunikacji trybu WESP (patrz podpunkt 18.8.3.2) jest sygnalizowana za pomocą wartości 16415 (USE_WESP_MODE). Pola danych powiadomień mogą zawierać dane o różnych rozmiarach, zależne od rodzaju powiadomienia.

Pola danych konfiguracji (*CP*) również przenoszą dodatkowe informacje (podobnie jak pola *N*), ale są wykorzystywane przede wszystkim do wstępnej konfiguracji systemu, np. w celu uzyskania w ramach komunikacji IKE parametrów dostarczanych standardowo przez protokół DHCP (patrz rozdział 6.). Pola danych konfiguracji należą do czterech

kategorii: CFG_REQUEST, CFG_REPLAY, CFG_SET oraz CFG_ACK. Z kolei przenoszone przez nie dane są zapisywane w formie par atrybut-wartość (ATV — *Attribute-Value*). W dokumencie [IKEPARAMS] zdefiniowano ok. 20 par tego typu. Większość dotyczy pozyskiwania informacji o adresach IPv4 i IPv6, a także maskach podsieci i adresach serwerów DNS. Są one szczególnie istotne w przypadku konfiguracji protokołu IPv6 (z uwagi na użycie protokołu ICMPv6 w procesie bezstanowej automatycznej konfiguracji oraz protokołu wykrywania sąsiedztwa — patrz rozdział 8.). Konfigurację węzła IPv6 w ramach połączenia VPN bazującego na protokole IPsec zdefiniowano w eksperymentalnej specyfikacji [RFC5739].

18.8.1.6. Wybór i użycie algorytmów

Transformaty odpowiedzialne za definiowanie zestawów kryptograficznych zostały w specyfikacji IKE podzielone na cztery rodzaje: szyfrowanie (typ 1., wykorzystywany w protokołach IKE i ESP), PRF (typ 2., wykorzystywany w protokole IKE), ochrona integralności (typ 3., wykorzystywany w protokołach IKE i AH oraz opcjonalnie w ESP) i grupa DH (typ 4., wykorzystywany w protokole IKE oraz opcjonalnie w AH i ESP). Mimo że mechanizm IKE ma możliwość negocjowania zestawu kryptograficznego używanego w relacji SA w danym kierunku, istnieje pewien zbiór algorytmów (transformat) obowiązkowy niezależnie od danej implementacji. Ponadto kilka algorytmów wskazano jako zalecane, co oznacza, że mogą się stać obowiązkowe w najbliższej przyszłości. Wszystkie ze wspomnianych algorytmów zostały zestawione w dokumencie [RFC4307] (oraz w tabeli 18.3).

Tabela 18.3. *Algorytmy obowiązkowo implementowane wraz z protokołem IKEv2 (pogrupowane według numeru typu)*

Przeznaczenie	Nazwa	Numer	Status	Oryginalna specyfikacja RFC
Transformata IKE typu 1. (szyfrowanie)	ENCR_3DES	3	Wymagana	[RFC2451]
	ENCR_NULL	11	Opcjonalna	[RFC2410]
	ENCR_AES_CBC	12	Zalecana	[RFC3602]
	ENCR_AES_CTR	13	Zalecana	[RFC3686]
Transformata IKE typu 2. (funkcje PRF)	PRF_HMAC_MD5	1	Opcjonalna	[RFC2104]
	PRF_HMAC_SHA1	2	Wymagana	[RFC2104]
	PRF_AES128_CBC	4	Zalecana	[RFC4434]
Transformata IKE typu 3. (integralność)	AUTH_HMAC_MD5_96	1	Opcjonalna	[RFC2403]
	AUTH_HMAC_SHA1_96	2	Wymagana	[RFC2404]
	AUTH_AES_XCBC_96	5	Zalecana	[RFC3566]
Transformata IKE typu 4. (grupy DH)	1024 MODP (Grupa 2)	2	Wymagana	[RFC2409]
	2048 MODP (Grupa 14)	14	Zalecana	[RFC3526]

Oficjalny rejestr wartości [IKEPARAMS] utrzymuje organizacja IANA. Choć przedstawiona lista zawiera wszystkie obowiązkowe algorytmy, w czasie pisania książki opracowano i opublikowano wiele kolejnych algorytmów, grup i technik, włącznie z podpisami cyfrowymi bazującymi na mechanizmach ECC (patrz [RFC4754]).

18.8.1.7. Wymiana IKE_AUTH

Zgodnie z przedstawionymi wcześniej informacjami wartość SKEYSEED służy do wyznaczania kluczy szyfrowania i uwierzytelniania, które z kolei umożliwiają zabezpieczanie pól danych podczas wymiany `IKE_AUTH`. Klucze te są nazywane odpowiednio `SK_e` oraz `SK_a`. Zapis `SK{P1, P2, ..., PN}` informuje, że pola danych `P1, ..., PN` są szyfrowane i objęte ochroną integralności właśnie za pomocą wspomnianych kluczy. Podstawowym celem wymiany `IKE_AUTH` jest potwierdzenie tożsamości każdej ze stron komunikacji. Dodatkowo jednak zapewnia ona wymianę informacji potrzebnych do ustanowienia pierwszej relacji `CHILD_SA`.

Rozpoczynając wymianę `IKE_AUTH`, inicjator wysyła pole danych `SK {IDi, AUTH, SAi2, TSi, TSr}`. Jeśli klucz deszyfrowania jest poprawny, zapewnia ono dostarczenie identyfikatora inicjatora, danych uwierzytelniających potwierdzających tożsamość inicjatora, pola danych kolejnej relacji *SA* (pierwszej relacji `CHILD_SA`) o nazwie *SAi2* oraz pary **selektorów ruchu** (pól *TSi* i *TSr* omówionych w podpunkcie 18.8.1.8). Inicjator może również przekazać własny certyfikat (w polu *CERT*), żądanie dostarczenia certyfikatu (pole *CERTREQ*) określające kotwicę zaufania oraz identyfikator odpowiadającego (w polu *IDr*). Przesłanie identyfikatora strony odpowiadającej okazuje się bardzo użyteczne w przypadkach, w których odpowiadający posługuje się większą liczbą identyfikatorów skojarzonych z tym samym adresem IP i trzeba wskazać właściwy do ustanowienia określonej relacji zabezpieczeń. Standard pozwala na wykorzystanie także wielu innych typów danych identyfikacyjnych, w tym adresu IP, kwalifikowanej nazwy domenowej (FQDN), adresu e-mail oraz niepowtarzalnej nazwy (stosowanej w połączeniu z certyfikatami X.509). Pełna lista typów znajduje się w rejestrze IKEv2 Identification Payload ID Types [IKEPARAMS].

Wiadomość kończąca wymianę zawiera identyfikator odpowiadającego (*IDr*), dane uwierzytelniające potwierdzające tożsamość odpowiadającego (*AUTH*), dane innej relacji *SA* (`CHILD_SA` — *SAr2*) oraz zbiór selektorów ruchu (*TSi* i *TSr*), który może być podzbiorem pierwotnych wartości *TSi* i *TSr*. Wszystkie pola danych transmitowane w ramach wymiany `IKE_AUTH` są szyfrowane i objęte mechanizmem ochrony integralności. Na tym etapie dopuszczalne jest również przesyłanie certyfikatu lub certyfikatów (w polu danych *CERT*). Jednak wymaga to poprzedzenia listy certyfikatów kluczem publicznym, który umożliwia sprawdzenie pola *AUTH*. Sama treść zależy od rodzaju wybranego wcześniej zestawu kryptograficznego. W trakcie wymiany obie strony są zobowiązane do sprawdzania wszystkich dostępnych sygnatur w celu wyeliminowania ewentualnych prób ujawnienia informacji i wykluczenia ataków typu MITM.

18.8.1.8. Selektory ruchu i pola danych TS

Selektory ruchu wskazują pola oraz wartości datagramu IP, które są „selekcjonowane" do przetwarzania z użyciem mechanizmu IPsec. Wraz z informacjami pozyskiwanymi z bazy SPD pozwalają na ustalenie, czy dany datagram powinien zostać zabezpieczony

za pomocą protokołu IPsec. Zgodnie z wcześniejszymi informacjami, datagramy nie objęte ochroną pomijają moduł IPsec lub są usuwane.

Zawartość pola danych TS może obejmować zakresy adresów IPv4 lub IPv6, zakresy numerów portów, identyfikator protokołu nagłówka IPv4 lub IPv6. Wartości zakresów są najczęściej zapisywane z użyciem symboli wieloznacznych. Przykładowo zapis 192.0.2.* odpowiada notacji 192.0.2.0/24 reprezentującej zakres 192.0.2.0 – 192.0.2.255. Przy użyciu selektorów ruchu możliwe jest wdrożenie polityki określającej, który zestaw kryptograficzny jest wymagany do ustanowienia relacji *SA* z określoną jednostką na portach z podanego zakresu. Większość z wymienionych parametrów jest przekazywana za pomocą interfejsu zarządzania do bazy SPD. Przekazanie wartości *TS* następuje natomiast w czasie wymiany `IKE_AUTH` w polach danych *TSi* i *TSr*. Jeśli jeden zakres jest mniejszy od drugiego, wybierany jest mniejszy zakres (zgodnie z zasadą „zawężania").

18.8.1.9. EAP i IKE

Mimo że protokół IKE dysponuje własnymi metodami uwierzytelniania (patrz sekcja 2.15 dokumentu [RFC5996]), może również współdziałać z mechanizmem EAP (patrz sekcje 2.16 i 3.16 dokumentu [RFC5996]). Rozwiązanie to okazuje się bardzo użyteczne, ponieważ system EAP udostępnia znacznie więcej metod uwierzytelniania, niż zostało zdefiniowanych w relatywnie niewielkim zbiorze funkcji obsługi współdzielonych kluczy i publicznych certyfikatów protokołu IKE. W praktyce ograniczone możliwości operowania kluczami są jednym z powodów niezbyt intensywnego upowszechniania się mechanizmów IPsec.

Zamiar skorzystania z rozwiązania EAP jest sygnalizowany drugiej stronie przez pominięcie pola danych *AUTH* w trzecim komunikacie wymiany `IKE_AUTH` (patrz rysunek 18.1). Dołączając pole danych *IDi*, ale bez pola *AUTH*, inicjator dostarcza informacje o swojej tożsamości, ale nie może ich potwierdzić. Jeśli mechanizm EAP zostanie zaakceptowany, odpowiadający odsyła pole danych EAP i wstrzymuje się z przesłaniem pól *SAr2*, *TSi* oraz *TSr* do czasu zakończenia procesu uwierzytelniania w ramach mechanizmu EAP. Wspomniana operacja kończy się z chwilą przesłania przez inicjatora pola danych *AUTH* właściwego dla komunikacji EAP, które można zweryfikować po stronie odpowiadającego (wymaga to przeprowadzenia przynajmniej jednej wymiany pól EAP).

Jedną z niedogodności wynikających z zastosowania protokołu EAP w komunikacji IKE jest podwojenie procedury uwierzytelniania. Dotyczy to przede wszystkim starszych metod EAP, które zapewniają jednokierunkowe potwierdzenie autentyczności (stacji końcowej w urządzeniu uwierzytelniającym) i wymagają od mechanizmu IKE przeprowadzenia uwierzytelnienia dla drugiego kierunku komunikacji na podstawie certyfikatu. Ponieważ wdrażanie systemu kluczy publicznych bywa niekiedy kłopotliwe, a nowsze metody EAP zapewniają wzajemne uwierzytelnianie jednostek oraz generowanie kluczy, w dokumencie [RFC5998] opisano również techniki uwierzytelniania bazujące *jedynie* na rozwiązaniach EAP. Wysłanie przez inicjatora powiadomienia `EAP_ONLY_AUTHENTICATION` powoduje wstrzymanie wysyłania pól danych *AUTH* i *CERT* w komunikacie 4. (zgodnie z rysunkiem 18.1). Kolejne pola *AUTH* wykorzystują natomiast klucze wygenerowane przez mechanizm EAP (zamiast wartości `SK_pi` oraz `SK_pr`).

Uwierzytelnienie bazujące całkowicie na protokole EAP wymusza zastosowanie jedynie metod gwarantujących odpowiedni poziom bezpieczeństwa, a w konsekwencji eliminuje potrzebę przeprowadzenia uwierzytelniania właściwego dla protokołu IKE. Wspomniane metody są nazywane **bezpiecznymi** metodami EAP. Muszą one zapewniać wzajemne uwierzytelnienie, generowanie kluczy oraz zabezpieczenie przed atakami słownikowymi. W dokumencie [RFC5998] zamieszczono listę 13 metod spełniających wymienione kryteria, wśród których są: EAP-TLS, EAP-FAST oraz EAP-TTLS.

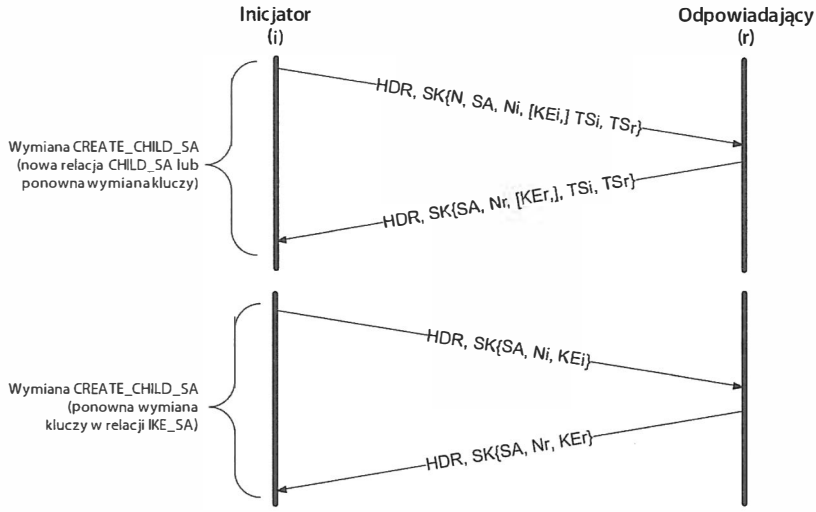
18.8.1.10. Zabezpieczenia typu „lepsze niż nic" (BTNS)

Ostatnie prace nad rozwiązaniami IKE i IPsec doprowadziły do opracowania rozwiązań określanych jako „lepsze niż nic" (BTNS — *Better-Than-Nothing Security*). Mają one zapewnić użyteczność i łatwość implementacji rozwiązań IPsec, szczególnie w zakresie wdrażania infrastruktury PKI oraz systemu uwierzytelniania na podstawie certyfikatów [RFC5387]. Z technicznego punktu widzenia BTNS jest formą nieuwierzytelnionego połączenia IPsec [RFC5386] wymagającego ustanowienia relacji SA za pomocą protokołu IKE. W rozwiązaniu tym wykorzystuje się klucze publiczne, ale nie są sprawdzane ich powiązania z głównym urzędem certyfikacji. W rezultacie dana relacja SA zapewnia, że w komunikacji przez cały czas bierze udział to samo urządzenie, ale nie może potwierdzić tożsamości jednostki podczas ustanawiania relacji SA. Taka forma uwierzytelnienia jest sposobem na **zachowanie ciągłości skojarzenia**. Niestety, jest jednak mniej wiarygodna niż obowiązująca w zasadniczym mechanizmie IPsec zasada gwarantowania **autentyczności źródła danych**. Wprowadzenie rozwiązania BTNS nie oznacza żadnych dodatkowych zmian w stosie IPsec. Nie wpływa na format komunikatów IKE, AH ani ESP.

18.8.1.11. Wymiana CREATE_CHILD_SA

Wymiana `CREATE_CHILD_SA` jest wykorzystywana do utworzenia relacji `CHILD_SA` niezbędnej do działania protokołów ESP lub AH bądź do ponownej wymiany kluczy w ramach istniejącej relacji SA (`IKE_SA` lub `CHILD_SA`) po zakończeniu pierwotnej wymiany. Opisywana operacja bazuje na pojedynczej wymianie pakietów, która może zostać rozpoczęta przez dowolną ze stron relacji `IKE_SA`, ustanowionej w ramach początkowej wymiany. Wykonanie zadania jest różne, zależnie od tego, czy dotyczy modyfikacji relacji `IKE_SA`, czy `CHILD_SA`. Oba warianty zostały przedstawione na rysunku 18.14 prezentującym sytuację, w jakiej wymianę `CREATE_CHILD_SA` rozpoczyna strona, która niekoniecznie musi być inicjatorem relacji `IKE_SA`.

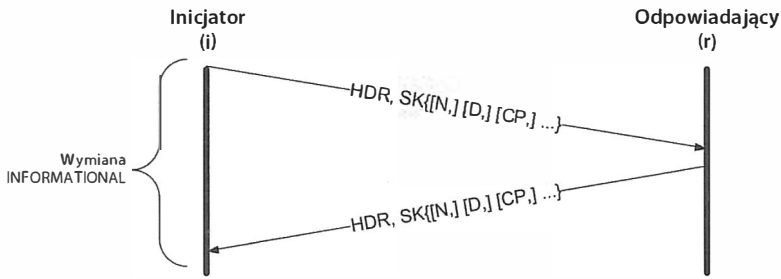
W pierwszym przypadku zaprezentowano wymianę `CREATE_CHILD_SA` odpowiadającą za ustanowienie nowej relacji `CHILD_SA` lub ponowną wymianę kluczy w istniejącej relacji. Ponowna wymiana kluczy jest sygnalizowana obecnością powiadomienia `N(REKEY_SA)` wysyłanego przez inicjatora. Realizacja zadania polega na utworzeniu nowej relacji *SA*, a następnie usunięciu dotychczasowej (więcej informacji na ten temat znajduje się w kolejnym podpunkcie). Wyznaczenie nowej relacji *SA* oraz selektorów ruchu (TS) pozwala na zmianę większości parametrów połączenia. Jeśli jest to konieczne, strony mogą również wymienić się nowymi wartościami DH (wykorzystując pola danych KE). Dzięki temu uzyskuje się większe bezpieczeństwo przekazywania danych w ramach nowej relacji *SA*. Ponowna wymiana kluczy w relacji `IKE_SA` sprowadza się do przeprowadzenia analogicznej operacji. Różnica polega jedynie na tym, że wymagane jest dołączenie pól KE oraz pominięcie pól TS, co można zauważyć w drugiej części rysunku 18.14.



Rysunek 18.14. *Wymiana CREATE_CHILD_SA służy do utworzenia nowej relacji CHILD_SA lub ponownej wymiany kluczy w ramach tej relacji bądź też do ponownej wymiany kluczy w relacji IKE_SA. Zmiana relacji CHILD_SA wymaga przesłania pola powiadomienia, które zawiera identyfikator SPI modyfikowanej relacji SA*

18.8.1.12. Wymiana informacyjna (INFORMATIONAL)

Wymiana typu `INFORMATIONAL` umożliwia przekazanie informacji statusowych oraz komunikatów o błędach. Bazuje zazwyczaj na polach danych powiadomień (*N*). Jest również wykorzystywana do usuwania relacji *SA*, do czego służy pole danych usunięcia (*D* — *Delete*), przesyłane także w procedurze ponownej wymiany kluczy. Przebieg operacji został przedstawiony na rysunku 18.15.



Rysunek 18.15. *Wymiana INFORMATIONAL pozwala na przekazanie informacji statusowych oraz na usunięcie relacji SA. Wykorzystuje pola danych powiadomienia (N), usuwania (D) oraz konfiguracji (CP)*

Wymiana `INFORMATIONAL` może zostać wykonana jedynie po poprawnym zakończeniu wymian inicjujących komunikację. Obejmuje opcjonalny zestaw powiadomień, pole danych usuwania (*D*) wyznaczające relację *SA* do usunięcia (na podstawie wartości *SPI*) oraz pole danych konfiguracji (*CP*). Część komunikatów odebranych od inicjatora wymaga

odesłania odpowiedzi nawet w przypadku dostarczenia pustej wiadomości IKE (zawierającej jedynie nagłówek). Brak odpowiedzi może spowodować niepotrzebną retransmisję komunikatu. W takich (nietypowych) przypadkach wykorzystuje się komunikaty INFORMATIONAL (mimo że nie jest realizowana wymiana INFORMATIONAL). Najczęściej strony sygnalizują w ten sposób odbiór wiadomości IPsec o nierozpoznanej wartości *SPI* lub o nieobsługiwanej wyższej wersji IKE.

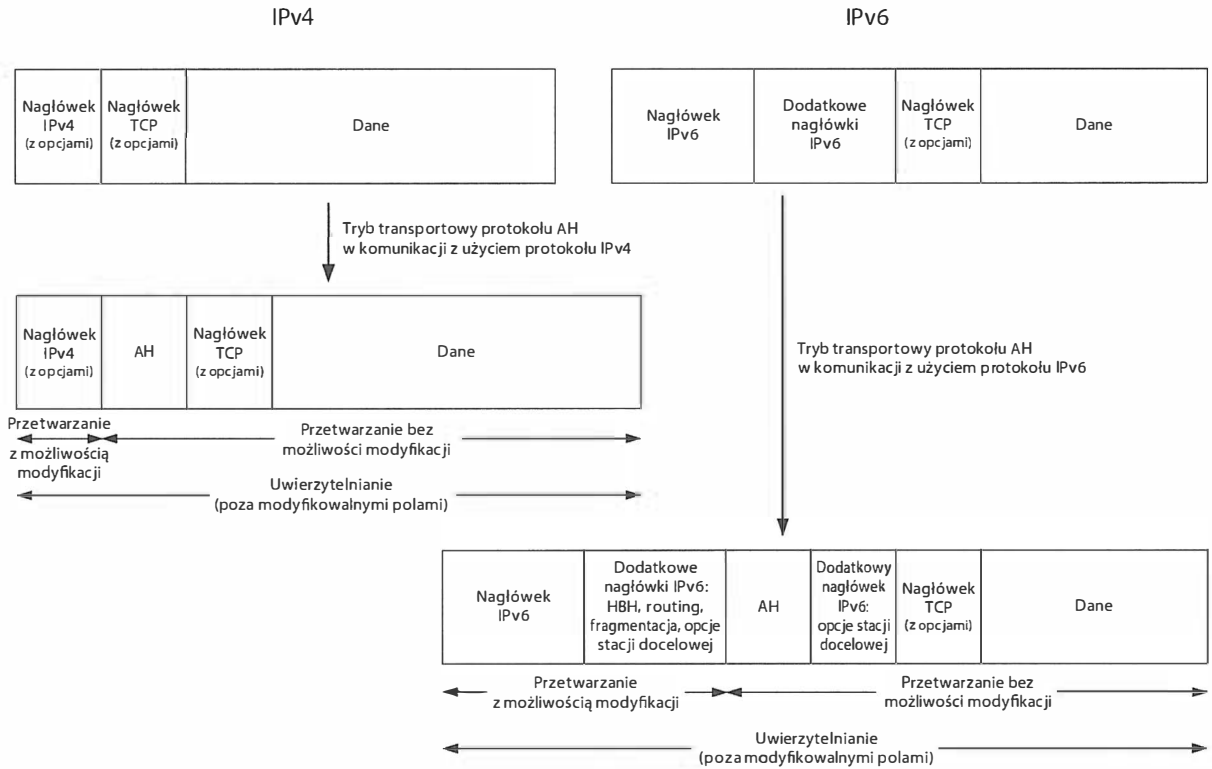
18.8.1.13. Protokół IKE dla urządzeń mobilnych (MOBIKE)

Relacja IKE_SA po ustanowieniu jest utrzymywana przez dłuższy czas. Jednak w przypadku pracy w środowiskach, w których adresy IP ulegają zmianie z uwagi na przemieszczanie się urządzeń (lub z powodu uszkodzenia interfejsu), konieczne jest wykorzystanie pewnej odmiany protokołu IKE, która została opisana w zaleceniu [RFC4555] i nazwana MOBIKE. Rozwiązanie MOBIKE rozszerza podstawowy protokół IKEv2 o opcję „zmiany adresu" uzgadnianą za pomocą wymiany INFORMATIONAL. W specyfikacji opisano sposób postępowania, gdy adresy są znane. Nie sprecyzowano natomiast zasad pozyskiwania wspomnianych adresów.

18.8.2. Protokół AH

Zdefiniowany w dokumencie [RFC4302] protokół AH jest jednym z podstawowych elementów systemu IPsec (choć jest rozwiązaniem opcjonalnym w stosie protokołów IPsec), który gwarantuje zachowanie autentyczności i integralności (ale nie poufności) datagramów IP. Ponieważ protokół AH zapewnia jedynie integralność danych, a nie ich poufność, i nie współdziała z funkcją NAT (więcej informacji na ten temat znajduje się w końcowej części punktu), jest znacznie mniej popularny niż drugi z mechanizmów ochrony danych wchodzących w skład systemu IPsec. Działając w trybie transportowym, protokół AH umieszcza dodatkowy nagłówek pomiędzy nagłówkiem warstwy 3. (IPv4, IPv6 lub rozszerzeniem IPv6) a nagłówkiem kolejnego protokołu (np. UDP, TCP lub ICMP). Podczas pracy w środowisku IPv6 informacje AH są zapisywane bezpośrednio przed nagłówkiem rozszerzenia przenoszącego opcje stacji docelowej (*destination options*), o ile takowy został zdefiniowany. W trybie tunelowym „wewnętrzny" nagłówek IP opisuje oryginalny datagram IP — określa właściwe źródło pakietu IP oraz wskazuje ostateczną stację docelową. Natomiast „zewnętrzny" nagłówek IP przechowuje parametry jednostek IPsec. Pracując w tym trybie, protokół AH zabezpiecza cały wewnętrzny datagram IP. Można zatem stwierdzić, że tryb transportowy służy do komunikacji między jednostkami, które są połączone bezpośrednio, a tryb tunelowy znajduje zastosowanie w połączeniach między bramami SG lub między stacją końcową i bramą SG (np. w połączeniach VPN). Przykład zastosowania protokołu AH do zabezpieczenia segmentów TCP przenoszonych w datagramach IPv4 i IPv6 został przedstawiony na rysunku 18.16.

W przypadku użycia protokołu AH (zgodnie z rysunkiem 18.16) nagłówek IPv4 musi przechowywać w polu numeru protokołu wartość 51. W transmisji IPv6 pakiety zawierają dodatkowy nagłówek AH umiejscowiony między opcjami stacji docelowej i pozostałymi opcjami protokołu. W obu rozwiązaniach wynikowy datagram składa się z części **modyfikowalnej** oraz **niemodyfikowalnej**. Część modyfikowalna jest zmieniana podczas przesyłania pakietu przez sieć. Modyfikowane są takie parametry jak *TTL* w protokole



Rysunek 18.16. *Wykorzystanie protokołu AH do zapewnienia autentyczności i integralności datagramów IPv4 i IPv6. W trybie transportowym standardowy pakiet IP jest uzupełniany danymi mechanizmu AH*

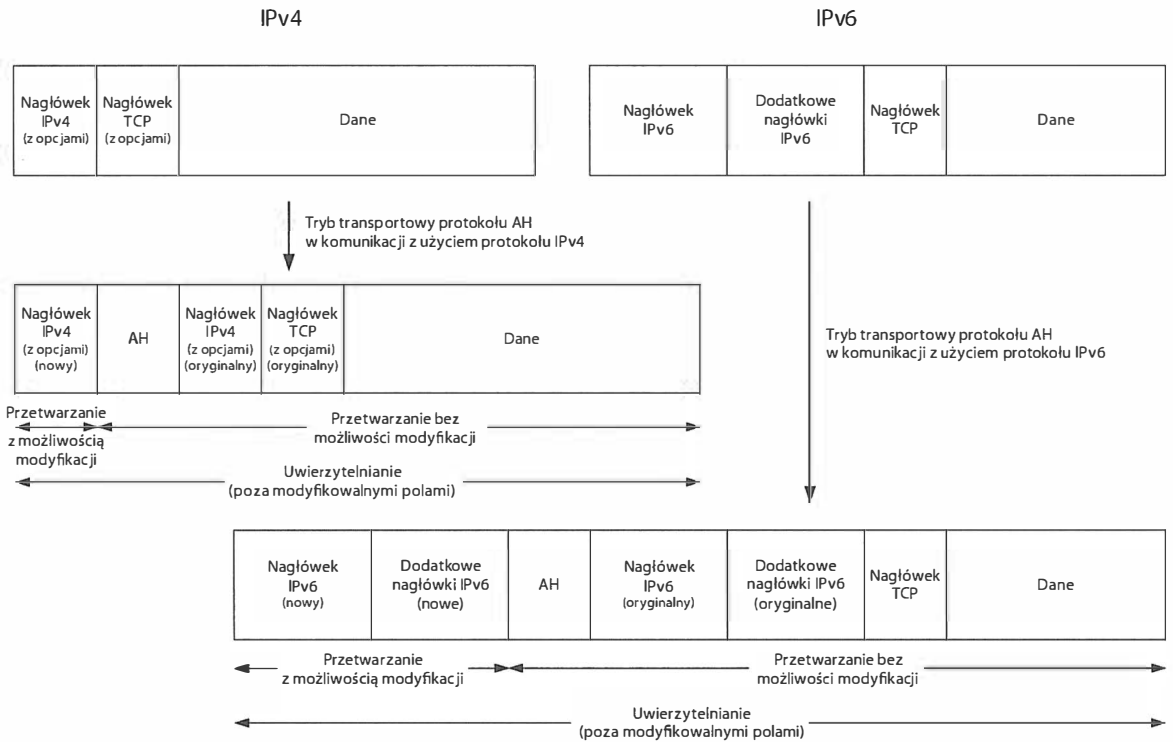
IPv4 lub *Limit węzłów* w protokole IPv6, pole *etykiety strumienia* IPv6, pole *DS* oraz bity *ECN*. Niezmienna część, obejmująca źródłowy i docelowy adres IP, nie może być modyfikowana podczas przekazywania pakietów przez sieć, ponieważ jest objęta ochroną integralności zapewnianą przez protokół AH. Z tego powodu tryb transportowy nie może być stosowany do przekazywania datagramów AH przez routery z uruchomioną funkcją NAT, co jest przyczyną wielu problemów z wdrażaniem opisywanego systemu. Tryb transportowy nie pozwala również na przekazywanie fragmentów pakietów (IPv4 lub IPv6).

Rozwiązaniem problemu jest wykorzystanie trybu tunelowego, którego działanie zostało zilustrowane na rysunku 18.17. W tym trybie oryginalny datagram jest transmitowany przez sieć w niezmienionej formie dzięki umieszczeniu go w nowym datagramie IP (objętym ochroną integralności).

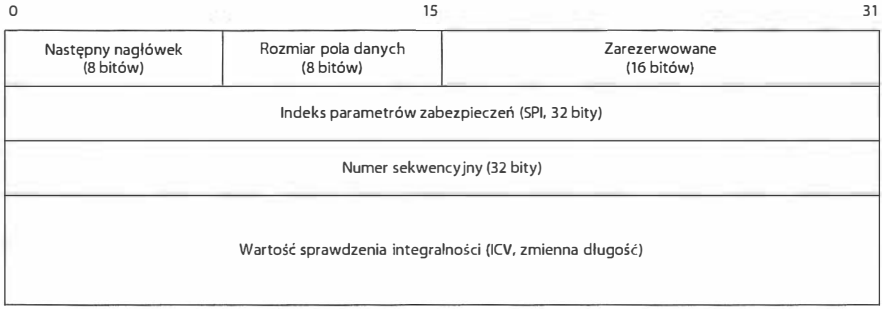
Praca w trybie tunelowym polega na zapisywaniu całego oryginalnego datagramu IP w polu danych kolejnego pakietu i zabezpieczeniu go za pomocą mechanizmu AH. Nagłówek wewnętrznego pakietu pozostaje wówczas niezmieniony. Natomiast nagłówek zewnętrzny otrzymuje adresy IP źródłowy i docelowy właściwe dla bramy SG lub stacji końcowej. W takich przypadkach mechanizm AH chroni cały pierwotny datagram oraz niektóre elementy nowego nagłówka (uniemożliwiając wykonywanie operacji NAT).

W obu trybach pracy protokołu AH wykorzystywany jest taki sam format nagłówka (pokazany na rysunku 18.18). Uwzględnia on informacje na temat długości datagramu, skojarzonych z nim relacji *SA* oraz dane niezbędne do sprawdzenia integralności. Pole *Długość pola danych* określa rozmiar pakietu AH wyrażony w 32-bitowych słowach i pomniejszony o 2. Wartość *Indeks parametrów zabezpieczeń* (SPI — *Security Parameters Index*) jest 32-bitowym identyfikatorem relacji *SA*, która pozwala odbiorcy na odczytanie parametrów skojarzonych z daną relacją *SA*. W przypadku multiemisji wartości SPI są przetwarzane w nieco odmienny sposób (więcej informacji na ten temat znajduje się w punkcie 18.8.4). Parametr *Numer sekwencyjny* jest 32-bitową wartością, inkrementowaną o 1 podczas wysyłania każdego pakietu w ramach określonej relacji *SA*. Pole to zabezpiecza komunikację przed odtworzeniem pakietu, o ile odbiorca z niego korzysta (nadawca zawsze wyznacza odpowiednią wartość, nawet jeśli nie jest ona przetwarzana przez odbiorcę). Standard protokołu uwzględnia również wykorzystanie **rozszerzonego numeru sekwencyjnego** (*ESN — Extended Sequence Number*). Praca w trybie uwzględniającym funkcję *ESN* jest rozwiązaniem zalecanym i negocjowanym w trakcie wymiany `IKE_SA_INIT`. Jeśli zostanie zaakceptowana, numer sekwencyjny jest wyliczany z użyciem wartości 64-bitowych, ale tylko 32 mniej znaczące bity są zapisywane w polu *Numeru sekwencyjnego*. Ostatnia wartość — *Wartość sprawdzenia integralności* (*ICV — Integrity Check Value*) — jest wyznaczana w sposób zależny od rodzaju zastosowanego zestawu kryptograficznego. Pole to zawsze przechowuje wartość dodatnią o rozmiarze odpowiadającym wielokrotności liczby 32.

Algorytm używany do ochrony integralności danych jest określany w odpowiedniej relacji *SA* jako transformata typu 3. i może zostać wskazany ręcznie lub z wykorzystaniem metod automatycznych (takich jak IKE). Wykaz opcjonalnych, zalecanych i wymaganych algorytmów (dla protokołów AH oraz ESP) znajduje się w dokumencie [RFC4835] i obejmuje rozwiązania: HMAC-MD5-96 (opcjonalne), AES-XCBC-MAC-96 (zalecane) oraz HMAC-SHA1-96 (obowiązkowe). Ochroną integralności objęte są następujące elementy datagramu: te pola nagłówka poprzedzające dane *AH*, które są niezmienne lub znane po stronie docelowej, nagłówek *AH* oraz wszystkie dane za nagłówkiem *AH*, najbardziej znaczące bity *ESN* (jeśli zostały wykorzystane) i ewentualne dopełnienie.



Rysunek 18.17. *Tryb tunelowy protokołu AH zapewnia uwierzytelnienie i ochronę integralności danych w datagramach IPv4 oraz IPv6. W rozwiązaniu tym standardowy datagram IP (przenoszący w tym przykładzie segment TCP) jest zapisywany w nowym, „zewnętrznym" datagramie IP*



Rysunek 18.18. *Protokół AH jest stosowany do uwierzytelniania i ochrony integralności datagramów IPv4 i IPv6 w trybach pracy tunelowym oraz transportowym. Wartość SPI identyfikuje relację SA, z którą powiązana jest dana komunikacja AH. Pole o nazwie Numer sekwencyjny zabezpiecza transmisję przed ponownym przesłaniem tych samych danych. Wartość ICV jest skrótem MAC wygenerowanym na podstawie niemodyfikowalnej części danych*

W praktyce w trybie tunelowym nieco kłopotliwe okazuje się przetwarzanie pewnych zmiennych pól, takich jak bity ECN, które sygnalizują przeciążenie połączenia (patrz rozdziały 5. i 16.). W dokumencie [RFC4301] zapisano zalecenie, aby były one po prostu kopiowane do nowo tworzonego zewnętrznego nagłówka IP. Z drugiej strony, w specyfikacji [RFC6040] zdefiniowano dwa tryby formatowania pakietów — **tryb normalny** oraz **tryb zgodności**. W trybie normalnym bity *CE* i *ECT* są kopiowane do nagłówka nowego datagramu. Z kolei w trybie zgodności bity te są zerowane, co prowadzi do utworzenia pakietu zewnętrznego w formacie uniemożliwiającym transport z wykorzystaniem bitów *ECN*. Jeśli podczas rozpakowywania pakietów zewnętrzny lub wewnętrzny nagłówek zawierają znacznik *CE*, jest on kopiowany do pakietu wynikowego, chyba że oryginalny pakiet nie zawiera informacji o obsłudze wartości *ECT* (w takim przypadku pakiet jest usuwany). Jeżeli znacznik *ECT* jest ustawiony w wewnętrznym lub zewnętrznym nagłówku, jest również ustawiany w odtworzonym pakiecie.

18.8.3. Protokół ESP

Protokół ESP jest jednym z elementów mechanizmu IPsec. Został opisany w dokumencie [RFC4303] (w którym jest określany jako ESP v3, mimo że formalnie numeracja wersji nie jest prowadzona) jako rozwiązanie zapewniające poufność, integralność i autentyczność datagramów IP oraz odporność na ich powtarzanie. Zakres realizowanych funkcji podlega konfiguracji. Jeśli w danej komunikacji istotna jest tylko ochrona integralności, wystarczy ustawić metodę szyfrowania na NULL [RFC2410] (obsługa tej metody jest obowiązkowa). Analogicznie możliwe jest wyłączenie ochrony integralności i skorzystanie jedynie z szyfrowania gwarantującego poufność informacji. Rozwiązanie to jednak jest podatne na ataki pasywne, więc nie zaleca się jego stosowania. W protokole ESP ochrona integralności oznacza potwierdzenie autentyczności źródła danych. Dzięki wyjątkowej elastyczności i długiej liście funkcji mechanizm ten jest znacznie bardziej popularny niż protokół AH.

18.8.3.1. Tryby transportowy i tunelowy

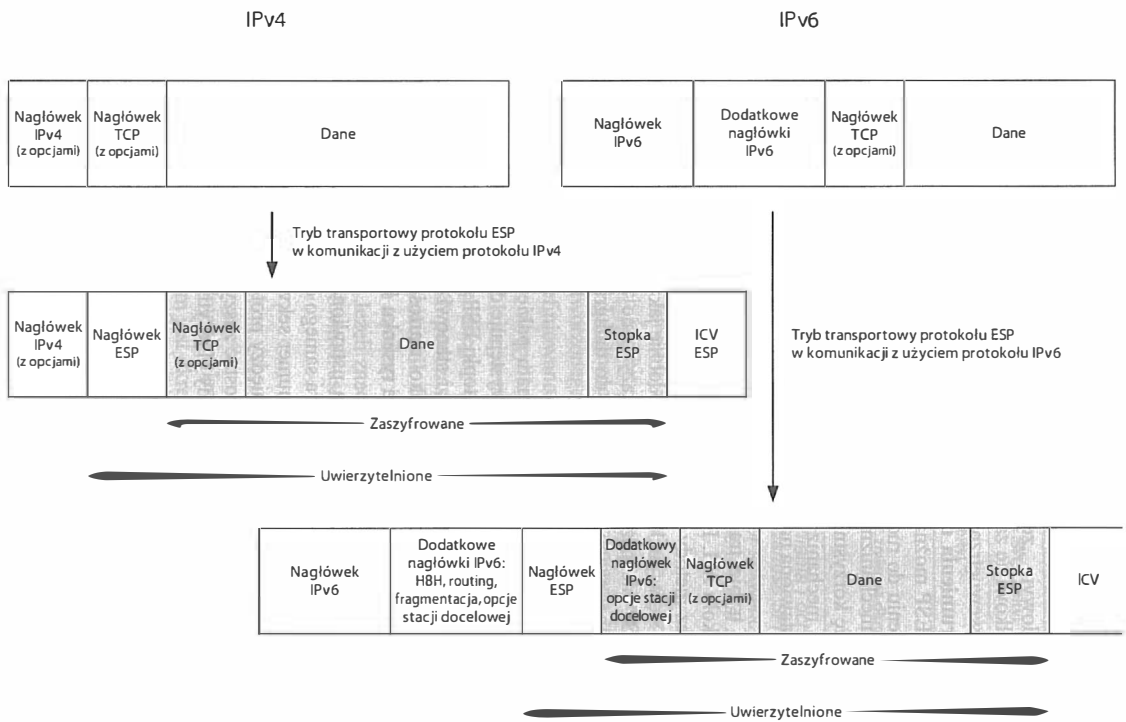
Podobnie jak AH, protokół ESP również pracuje w trybach transportowym i tunelowym. W trybie tunelowym „zewnętrzny" pakiet IP zawiera „wewnętrzny" pakiet IP, który może zostać całkowicie zaszyfrowany. Ukrycie za pomocą szyfrowania rozmiaru i zawartości pakietu wewnętrznego pozwala więc na wdrożenie w ograniczonej formie zasady **poufności strumienia danych** (TFC — *Traffic Flow Confidentiality*). Jeśli to konieczne, protokół ESP można stosować łącznie z protokołem AH. Oba znajdują zastosowanie w odniesieniu do ruchu IPv4 i IPv6. W niektórych wdrożeniach, z uwagi na efektywność, użycie mechanizmu ESP z ograniczeniem jedynie do ochrony integralności danych okazuje się korzystniejsze niż użycie protokołu AH. Dlatego jest wymaganą opcją konfiguracyjną mechanizmu IPsec. Enkapsulacja danych ESP w trybie transportowym została przedstawiona na rysunku 18.19.

Struktura pakietu w trybie transportowym jest zbliżona do obowiązującej w trybie transportowym protokołu AH. Różnica wynika jedynie z dodania stopki ESP, która jest wykorzystywana przez metody szyfrowania i ochrony integralności (więcej informacji na ten temat znajduje się w punkcie 18.8.3). Podobnie jak w protokole AH, tryb transportowy mechanizmu ESP nie pozwala na przekazywanie fragmentów pakietów. Budowa datagramów trybu tunelowego jest podobna do datagramów protokołu AH i została przedstawiona na rysunku 18.20.

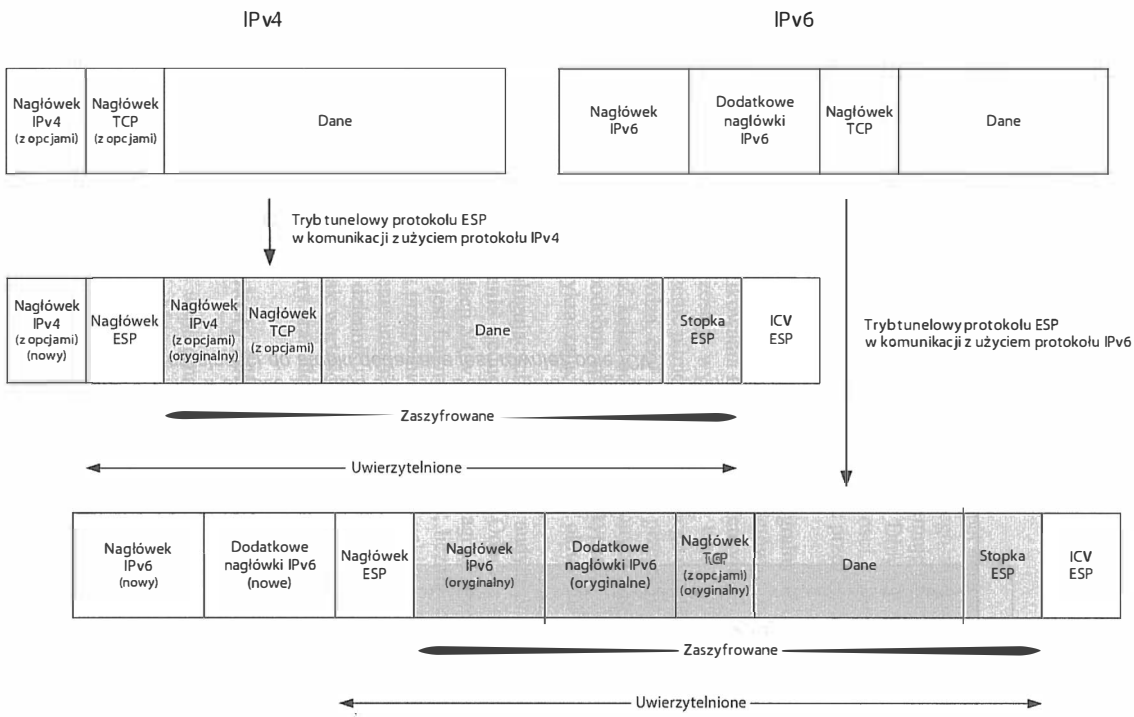
Specyfikacja ESP nie definiuje stałego formatu pakietu, tak jak w przypadku AH. Opisuje natomiast ogólną strukturę, w której występuje część nagłówka i stopki. Dodatkowo zdefiniowana została opcjonalna część stopki ESP. Jest ona dodawana w komunikacji wymagającej uwierzytelniania pakietów i stanowi przestrzeń do zapisu dodatkowych bitów wartości kontrolnej (opisanej jako wartość ICV protokołu ESP). Pełna struktura pakietu ESP została przedstawiona na rysunku 18.21.

Datagramy IP przenoszące pakiety ESP mają ustawioną wartość 50 w polu *Protokół* (IPv4) lub *Następny nagłówek* (IPv6). Struktura samego datagramu ESP (przedstawiona na rysunku 18.21) obejmuje indeks SPI oraz numer sekwencyjny (o takim samym znaczeniu, jakie ma w protokole AH). Różnica między protokołami uwidacznia się przede wszystkim w polu danych. Obszar ten może zostać zaszyfrowany oraz dopełniony bitami do rozmiaru wymaganego przez zastosowany algorytm szyfrowania.

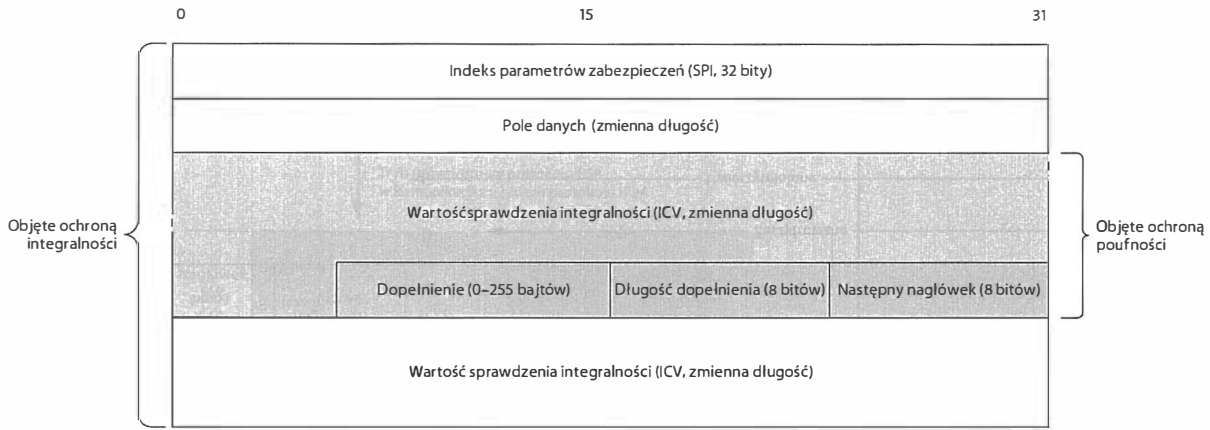
Całe pole danych nie może przekroczyć rozmiaru opisywanego za pomocą 32 bitów (64 bitów w protokole IPv6), w którym są uwzględnione dwa końcowe pola, odzwierciedlające *Długość dopełnienia* oraz identyfikator protokołu *Następnego nagłówka*. Wartości *Dopełnienia*, *Długości dopełnienia* i *Następnego nagłówka* stanowią stopkę pakietu ESP (zgodnie z rysunkami 18.19 i 18.20). W niektórych algorytmach kryptograficznych używany jest wektor IV. Jeśli jego przesłanie jest konieczne, wartość IV zapisuje się na początku pola danych (wartość ta nie została przedstawiona na rysunku). Dodatkowe dopełnienie, wymagane przez mechanizm TFC, może zostać zapisane w obszarze pola danych, przed stopką ESP (stosowny przykład został przedstawiony na rysunku numer 2 w dokumencie [RFC4303]). Jego użycie okazuje się konieczne do utajnienia długości datagramu i wyeliminowania ataków bazujących na analizie ruchu. W praktyce jednak technika ta nie jest zbyt często stosowana. Pole *Następny nagłówek* przechowuje wartości ze zbioru zdefiniowanego do użycia w polu *Protokół* mechanizmu IPv4 lub *Następny*



Rysunek 18.19. *Protokół ESP zapewnia poufność (dzięki szyfrowaniu), uwierzytelnienie oraz ochronę integralności datagramów IPv4 i IPv6. W trybie transportowym (przedstawionym na rysunku) standardowy datagram IP jest modyfikowany w taki sposób, aby obejmował również nagłówek ESP. Transportowane pole danych podlega szyfrowaniu, uwierzytelnianiu i ochronie integralności*



Rysunek 18.20. *W trybie tunelowym (przedstawionym tutaj podczas transmisji segmentu TCP) protokół ESP zapisuje standardowy datagram IP wewnątrz nowego „zewnętrznego" datagramu IP. Mechanizm ESP pozwala na modyfikowanie zewnętrznego datagramu (np. podczas operacji NAT), o ile wewnętrzny pakiet pozostanie nienaruszony. Rozwiązanie ESP jest znacznie częściej stosowane niż AH*



Rysunek 18.21. *Zaszyfrowane pole danych znajduje się w środku komunikatu ESP. Nagłówek ESP obejmuje indeks SPI, Numer sekwencyjny. Z kolei w stopce znajdują się pola Dopełnienia, Długości dopełnienia oraz Następnego nagłówka. Po włączeniu ochrony integralności do stopki dodawane jest również pole ICV*

nagłówek w protokole IPv6 (zapisany w nim identyfikator może mieć np. wartość 4 w przypadku datagramu IPv4 lub 41 w przypadku datagramu IPv6). Wartość 59 oznacza brak kolejnego nagłówka (taki pakiet nie niesie żadnych informacji i jest usuwany). Wysyłanie pustych pakietów jest kolejnym sposobem obrony przed atakami bazującymi na analizie ruchu.

Pole ICV jest obszarem stopki ESP o zmiennej długości, używanym jedynie w połączeniu z mechanizmem kontroli integralności. Jego wartość jest wyznaczana na podstawie nagłówka ESP, pola danych i stopki. Obejmuje również wartości nieokreślone bezpośrednio (np. bardziej znaczące bity ESN). Długość pola ICV wynika z rodzaju zastosowanego algorytmu weryfikacji integralności. Z tego powodu jest określana w chwili ustanawiania relacji SA i nie ulega zmianie przez cały czas obowiązywania tej relacji.

Jeśli zostało włączone zabezpieczenie integralności danych, aktywny jest również mechanizm ochrony przed ponawianiem pakietów. Jego działanie sprowadza się do wyznaczania i weryfikowania numerów sekwencyjnych. W chwili ustanowienia relacji *SA* licznik pakietów zostaje wyzerowany. Od tego momentu każda transmisja datagramu powoduje zwiększenie numeru sekwencyjnego o jeden. W czasie działania zabezpieczenia nadawca sprawdza, czy licznik się nie przepełnia, i w razie konieczności ustanawia nową relację *SA*. Z kolei odbiorca chronionych danych zarządza oknem numerów sekwencyjnych (w podobny sposób, jak w przypadku okna TCP). Datagramy o numerach sekwencyjnych wykraczających poza ramy okna są usuwane.

W systemach obsługujących mechanizmy audytu przetwarzanie ESP może generować **audytowe zdarzenia** (jedno lub więcej). Odnoszą się one do następujących sytuacji: brak poprawnej relacji *SA* w danej sesji, datagram przekazany do przetwarzania w ramach mechanizmu ESP jest fragmentem pakietu, licznik zapobiegający odtwarzaniu pakietów ma wartość bliską przepełnienia, odebrany pakiet jest spoza okna dopuszczalnych datagramów, sprawdzenie integralności zakończyło się niepowodzeniem. Zdarzenia audytowe są rejestrowane w dzienniku zdarzeń systemowych. Wraz z nimi zapisywane są dodatkowe metadane, takie jak wartość SPI, bieżąca data i czas, adresy IP źródłowy i docelowy, numer sekwencyjny oraz identyfikator strumienia IPv6 (jeśli jest dostępny).

18.8.3.2. Protokoły ESP-NULL, WESP oraz widoczność ruchu

Jak już wcześniej wspominaliśmy, za pomocą szyfrowania protokół ESP zapewnia poufność informacji. Może jednak pracować w trybie samego sprawdzania integralności. Wykorzystuje wówczas algorytm szyfrowania o nazwie NULL. Tryb samej ochrony integralności (ESP-NULL) bywa potrzebny w środowiskach (szczególnie korporacyjnych), w których implementowane są wyrafinowane mechanizmy inspekcji pakietów, a poufność jest zapewniana w inny sposób. Przykładowo w niektórych rozwiązaniach korporacyjnych urządzenia sieciowe analizują pakiety, poszukując w nich zabronionych treści (choćby sygnatur wirusów), i w razie wykrycia odstępstw od przyjętej polityki alarmują administratorów lub blokują ruch sieciowy. Działanie wspomnianych jednostek jest niemożliwe, gdy w transmisji wykorzystywany jest protokół ESP zapewniający szyfrowanie między urządzeniami końcowymi. Innymi słowy, aby inspekcja pakietów była możliwa, konieczne jest zapewnienie **widoczności ruchu**.

Gdy urządzenie analizujące ruch otrzyma pakiety strumienia ESP, musi ustalić, czy dany ruch jest szyfrowany (czy zastosowano algorytm NULL, czy nie). Biorąc pod uwagę to, że negocjacja zestawu kryptograficznego jest realizowana poza mechanizmem ESP (ręcznie lub z użyciem protokołu IKE), istnieją tylko dwie metody wykonania zadania. Pierwsza polega na przeprowadzeniu kilku niestandardowych testów i zgadywaniu [RFC5879]. Zaletą rozwiązania jest brak konieczności wprowadzania jakichkolwiek modyfikacji do protokołu ESP, które zapewniłyby widoczność ruchu. Inna technika sprowadza się do dołączenia do pakietu ESP znacznika informującego o zastosowaniu (lub nie) szyfrowania. Służy do tego algorytm opakowywania pakietów ESP — WESP (*Wrapped ESP*) [RFC5840]. Jego zadanie polega na dołączaniu specjalnego nagłówka przed samym pakietem ESP. Mechanizm WESP wykorzystuje niezależny numer protokołu (141), a jego użycie podlega negocjacji w ramach protokołu IKE (z użyciem pola danych powiadomień `USE_WESP_MODE` o wartości `16415`). Nagłówek WESP (o zmiennej długości) obejmuje pola, które wyznaczają położenie przenoszonych danych, oraz pole **znaczników** (definiowanych przez organizację IANA [IWESP]) wskazujących na wykorzystanie algorytmu ESP-NULL. Choć protokół WESP ułatwia klasyfikowanie ruchu w urządzeniach sieciowych, jego użycie zależy od obsługi mechanizmu w jednostce końcowej. Ponieważ jest to relatywnie nowe rozwiązanie, nie wszystkie obecnie funkcjonujące systemy pozwalają na jego stosowanie. Z drugiej strony, dzięki rozszerzalności formatu WESP zaimplementowanie go umożliwia zaadaptowanie rozwiązania również do innych celów w przyszłości.

18.8.4. Multiemisja

Protokół IPsec umożliwia opcjonalnie realizację zadań związanych z multiemisją [RFC5374], choć funkcja ta nie jest często wykorzystywana. W najprostszej formie wymaga ręcznej konfiguracji kluczy, chociaż istnieją również rozwiązania zapewniające automatyczne wyznaczanie kluczy w grupie multiemisji, nazywane **zarządzaniem kluczem grupy** (GKM — *Group Key Management*). Działanie mechanizmów GKM jest nadzorowane przez **kontrolery grupy/serwery kluczy** (GCKS — *Group Controller/Key Server*). Jednostki te są wykorzystywane do ustanawiania **grupowych relacji zabezpieczeń** (GSA — *Group Security Associations*), które obejmują co najmniej jedną relację SA protokołu IPsec oraz co najmniej jedną relację GKM SA zapewniającą dostarczenie parametrów niezbędnych do ustanowienia relacji SA protokołu IPsec [RFC3740]. Z uwagi na możliwość dynamicznego dołączania do grupy lub jej opuszczania protokoły GKM muszą znacznie częściej realizować operacje ponownego wyznaczania kluczy niż w mechanizmach dwustronnej wymiany danych. Z tego względu są one ulubionymi obiektami badań osób zajmujących się bezpieczeństwem sieci [AKNT04]. Szczegółowe omówienie zasad działania protokołu GKM byłoby niezwykle długie, dlatego czytelnicy zainteresowani tą problematyką powinni zapoznać się z dokumentacją GDOI [RFC3547] lub GSAKMP [RFC4535].

Obecnie multiemisja w ramach mechanizmu IPsec wymaga od wszystkich uczestników komunikacji stosowania jednakowych algorytmów i protokołów. Obsługiwane są operacje transmisji z dowolnych adresów multiemisji oraz z pojedynczych adresów multiemisji (ASM i SSM; więcej informacji na ten temat znajduje się w rozdziale 9.). Te same procedury dotyczą lokalnych adresów rozgłoszeniowych IPv4 i adresów transmisji dowolnej IPv6. Stacje IPsec mogą niezależnie wybierać pracę w trybach transportowym i tunelowym. Jednak działanie bram SG jest ograniczone do trybu tunelowego z użyciem adresów multiemisji jako adresów docelowych.

Obsługa datagramów multiemisji w trybie tunelowym IPsec okazuje się sporym wyzwaniem, ponieważ zewnętrzny datagram IP musi zawierać multiemisyjny adres docelowy. Jedynie wówczas zostanie poprawnie przekazany w infrastrukturze zdolnej do obsługi multiemisji. Wymaga to zastosowania specjalnej procedury nazywanej pracą w **trybie tunelowym z zachowaniem adresu** podczas wprowadzania datagramów AH lub ESP do tuneli. W skrócie zadanie polega na dobraniu zewnętrznego źródłowego i zewnętrznego docelowego adresu IP w taki sposób, aby odpowiadały one adresom wewnętrznym (przy założeniu, że wykorzystywana jest ta sama wersja protokołu IP). Celem jest zapewnienie, że (1) do przekazywania datagramu zostanie wykorzystany routing multiemisji oraz że (2) podczas wyznaczania tras multiemisji poprawnie zadziała mechanizm przekazywania na podstawie trasy powrotnej (PRF) (więcej informacji na ten temat znajduje się w rozdziale 9.).

Wprowadzenie multiemisji wymaga wykonania pewnych zmian w niskopoziomowych mechanizmach stosu IPsec przedstawionego na rysunku 18.10. Przykładowo bazy SPD i SAD muszą zostać zmodyfikowane w taki sposób, aby obsługiwały znaczniki „zachowania adresów" w trybach tunelowych. Ponadto w bazie SPD musi zostać obsłużony znacznik kierunkowości, który umożliwi ustalenie, w jakich okolicznościach relacja *SA* powinna zostać utworzona automatycznie. Zapobiega to tworzeniu relacji *SA* z użyciem jako adresu źródłowego niedozwolonego adresu mutliemisji (przez zwykłą zamianę adresów docelowego ze źródłowym, tak jak jest to realizowane w przypadku klasycznej transmisji z użyciem relacji *SA*). Baza SPD może wymagać określenia stanów definiujących przypadki, w których należy wykorzystać protokół GKM (np. w celu pozyskania klucza grupy). Z kolei grupowa baza danych PAD (GPAD) powinna przechowywać informacje na temat każdej jednostki GCKS wraz ze specyfikacją selektorów ruchu, które mogą posłużyć do utworzenia relacji *SA* oraz danych uwierzytelniających pozwalających na użycie określonego protokołu GKM z daną jednostką GCKS. Dane GPAD nie są analizowane przez protokoły spoza grupy GKM, takiej jak IKE. Niemniej jednak struktury PAD i GPAD mogą być implementowane w ramach jednego systemu.

18.8.5. Protokoły L2TP/IPsec

Protokół L2TP (więcej informacji na jego temat znajduje się w rozdziale 3.) odpowiada za tunelowanie ruchu warstwy 2., takiego jak PPP, w sieciach IP oraz połączeniach innego typu. Uwzględnia pewne metody uwierzytelniania w chwili ustanawiania połączenia, ale nie zapewnia uwierzytelniania, ochrony integralności i poufności każdego z kolejnych transmitowanych pakietów. Aby wyeliminować tę niedogodność, można połączyć działanie mechanizmów L2TP i IPsec [RFC3193]. Połączenie określane jako L2TP/IPsec jest zalecanym sposobem realizacji zdalnego dostępu do korporacyjnych (lub domowych sieci) — wyznacza standard połączeń VPN. Zabezpieczenie strumienia L2TP za pomocą mechanizmów IPsec można zrealizować przez zastosowanie bezpośredniej enkapsulacji L2TP w IP (o numerze protokołu 115) lub przez użycie w protokołach UDP i IP enkapsulacji, która ułatwia przekazywanie pakietów przez urządzenia NAT.

Standardowo w połączeniach L2TP/IPsec wykorzystuje się protokół IKE, choć istnieją również inne metody wymiany kluczy. Działanie systemu bazuje na relacjach *SA* ESP w trybie transportowym (wymagana obsługa) lub w trybie tunelowym (opcjonalna obsługa). Relacja *SA* zabezpiecza ruch L2TP, który jest odpowiedzialny za ustanowienie tunelu

przeznaczonego do transportu ruchu warstwy 2. Ponieważ opisywane rozwiązanie jest połączeniem dwóch protokołów, z których każdy dysponuje mechanizmami uwierzytelniania, ustanawianie połączeń L2TP/IPsec wymaga zazwyczaj przeprowadzenia dwóch niezależnych procedur potwierdzania tożsamości — jednej uwierzytelniającej urządzenie (z wykorzystaniem protokołu IPsec i współdzielonego hasła lub certyfikatów) oraz jednej uwierzytelniającej użytkownika (za pomocą nazwy i hasła lub tokenu).

Mechanizm L2TP/IPsec jest obsługiwany na większości nowoczesnych platform. W systemie Windows można z niego skorzystać, wybierając opcję utworzenia nowego „połączenia z miejscem pracy". Opcja L2TP jest również dostępna w narzędziach konfiguracji sieci większości smartfonów (pracujących pod kontrolą systemu Android lub iPhone). W systemie Mac OS X włączenie obsługi protokołów L2TP/IPsec sprowadza się do dodania odpowiedniego interfejsu sieciowego (w ramach ustawień systemowych). W systemie Linux konieczne może się okazać niezależne skonfigurowanie usług IPsec i L2TP. Jeżeli na danej platformie nie jest możliwe skorzystanie z protokołu L2TP, warto rozważyć użycie samego mechanizmu IPsec.

18.8.6. IPsec i funkcja NAT

Wykonanie operacji NAT w protokole IPsec sprawia dość sporo kłopotów. Przede wszystkim dlatego, że identyfikacja punktów końcowych bazuje na adresach IP, które nie powinny się zmieniać. To założenie projektowe nie zostało uzupełnione o rozwiązania, które sprawiłyby, że translacja adresów przestałaby stanowić problem. Z tego też powodu przez długi czas wdrażanie rozwiązań IPsec postępowało tak powolnie. Obecnie jednak mechanizm IPsec umożliwia zarówno zmianę adresów (z protokołem MOBIKE), jak i ich tłumaczenie (za pomocą funkcji NAT Traversal).

Aby zapewnić całościowe rozwiązanie problemu translacji adresów, trzeba rozważyć działanie protokołów IKE, AH i ESP w obu trybach — transportowym i tunelowym. Jak się okazuje, nie wszystkie warianty komunikacji IPsec umożliwiają pracę z urządzeniami NAT. Wymagania stawiane poszczególnym rozwiązaniom zostały przedstawione w opracowaniu [RFC3715]. Analizę zagadnienia rozpoczniemy więc od wyróżnienia podstawowych niezgodności między funkcjami NAT i mechanizmem IPsec, a następnie zaprezentujmy metody rozwiązania opisywanych problemów.

Jedna z największych trudności wynika ze sposobu działania protokołu AH oraz zasad modyfikowania adresów podczas wykonywania operacji NAT na transmitowanych datagramach. Praca mechanizmu AH bazuje na założeniu, że podczas wyliczania wartości MAC uwzględniane są również adresy IP pakietu. Oznacza to, że urządzenie realizujące funkcję NAT nie może zmienić adresu w taki sposób, który nie doprowadzi do odrzucenia datagramu przez moduł AH. Warto tutaj zauważyć, że problem ten nie dotyczy protokołu ESP, który nie obejmuje ochroną integralności pól adresów IP.

Kolejna trudność wynika ze stosowania w warstwie transportowej protokołów UDP i TCP wraz z sumą kontrolną pseudonagłówka, która jest wyliczana na podstawie adresów IP. Jeśli na poziomie warstwy transportowej wykorzystywany jest mechanizm ochrony integralności obejmujący sumę kontrolną lub stosowane jest szyfrowanie, nie można wykonać operacji NAT bez unieważnienia pakietu. Ten sam problem występuje w przypadku zmiany portów w ramach funkcji NAPT lub stosowania innych protokołów weryfikujących poprawność danych w różnych warstwach stosu.

Trzecia istotna trudność wiąże się z zapisywaniem identyfikatorów pól danych w protokole IKE. Do identyfikacji jednostki końcowej IKE może służyć wiele parametrów, jednym z nich jest adres IP. Jednak adresy zapisywane w polu danych IKE podlegają szyfrowaniu i nie można ich zmienić w trakcie operacji NAT bez uszkodzenia pakietu. W takich przypadkach trzeba polegać na innych technikach identyfikowania jednostek końcowych (np. na ciągach FQDN lub nazwach pozyskiwanych z certyfikatów X.509).

Czwarty problem wynika ze sposobu demultiplekacji ruchu w modułach NAT i NAPT wymaganej do ustalenia odbiorcy danych. W protokołach TCP i UDP służy do tego celu numer portu. Jednak protokoły AH i ESP, działając w sposób zbliżony do protokołów transportowych, nie wykorzystują numerów portów, lecz wartości SPI. Mimo że niektóre implementacje NAT mogą rozdzielać strumienie na podstawie wartości SPI, trzeba pamiętać, że wartości te są wybierane lokalnie przez jednostkę odpowiadającą w protokole IPsec i istnieje ryzyko powtórzenia się tej samej wartości w większej liczbie komputerów. Ponieważ moduł NAT nie może jej zmodyfikować, trzeba się liczyć z ewentualnością niewłaściwego rozdzielenia powracającego strumienia danych i wystąpienia błędów transmisyjnych.

Translacja NAT w komunikacji IPsec jest problematyczna z jeszcze jednego powodu. Niektóre protokoły warstwy aplikacji również przenoszą adresy IP w zaszyfrowanym lub objętym ochroną integralności polu danych (przykładem jest chociażby SIP). Zastosowanie mechanizmów zabezpieczających dodatkowo utrudnia analizę ruchu, gdyż uniemożliwia dekodowanie pakietów. Na szczęście, niektóre narzędzia monitorowania sieci (np. Wireshark) są w stanie przetwarzać zaszyfrowane datagramy, jeśli użytkownik dostarczy odpowiednie informacje na temat kluczy.

Rozwiązaniem większości problemów wynikających z konieczności wykonania operacji NAT jest enkapsulowanie ruchu ESP i IKE w datagramach protokołów UDP/P, które mogą być modyfikowane w tradycyjnych urządzeniach NAT (nie opracowano jednak mechanizmu umożliwiającego translację adresów w protokole AH). Inicjator IKE wykorzystuje port UDP o numerze 500 lub 4500 do wysłania początkowego komunikatu IKE, a następnie rozpoczyna wymianę datagramów UDP z pakietami ESP lub IKE na porcie 4500 (niezależnie od tego, czy funkcja NAT jest realizowana, czy nie). Zapisywanie komunikatów ESP w datagramie UDP wysyłanym z portu 500 jest zabronione [RFC5996]. Wybór portu 4500 wynika z konieczności ominięcia pewnych implementacji NAT, które w niewłaściwy sposób przetwarzają ruch IPsec na porcie 500.

Obsługa NAT w protokole IKE jest funkcją opcjonalną. Jeśli jest zaimplementowana, do wymiany `IKE_SA_INIT` dołączane są dwa pola danych powiadomień: `NAT_DETECTION_DESTINATION_IP` oraz `NAT_DETECTION_SOURCE_IP`. Są one umieszczane za polami *Ni* i *Nr*, a przed polem danych *CERTREQ*. Wśród informacji zawartych w wymienionych polach znajdują się: skrót SHA-1 indeksu SPI danej relacji *SA*, źródłowy lub docelowy adres IP oraz źródłowy lub docelowy numer portu. Dane te są nie podlegają modyfikacjom, ponieważ wchodzą w skład komunikatów IKE. Jeśli urządzenie odbiorcze wykryje wykonanie operacji NAT w czasie transmisji, dalsza komunikacja w ramach protokołu IKE odbywa się na porcie 4500 w protokole UDP. Dzięki temu translacja adresów przebiega bez zakłóceń.

Pakiety IKE odpowiedzialne za ustanowienie relacji *SA*, które zostały poddane co najmniej jednej translacji adresów, zawierają selektory ruchu (pola *TS*) o adresach nieistotnych dla odbiorcy (są to prywatne adresy IP obowiązujące „za" urządzeniem NAT). Nie odpowiadają one adresom, które są zapisane w polach adresowych datagramu IKE odbieranego przez odpowiadającego. Rozwiązanie problemu polega na zapisaniu wartości pozyskanych z pól IKE *TSi* i *TSr* do przyszłego wykorzystania oraz zastąpieniu ich źródłowym i docelowym adresem IP z odebranego datagramu. W praktyce oznacza to „odroczoną translację adresów" wykonaną przez odbiorcę na podstawie pól *TS*. Wynikowy datagram i pola danych TS są wykorzystywane w zapytaniach kierowanych do bazy danych SPD, która dostarcza informacji na temat polityki bezpieczeństwa obowiązującej w ramach tworzonej relacji *SA*. W przypadku trybu transportowego odpowiadający kończy wymianę, a inicjator wykonuje analogiczną podmianę adresów po swojej stronie (więcej informacji na ten temat znajduje się w sekcji 2.23.1 dokumentu [RFC5996]).

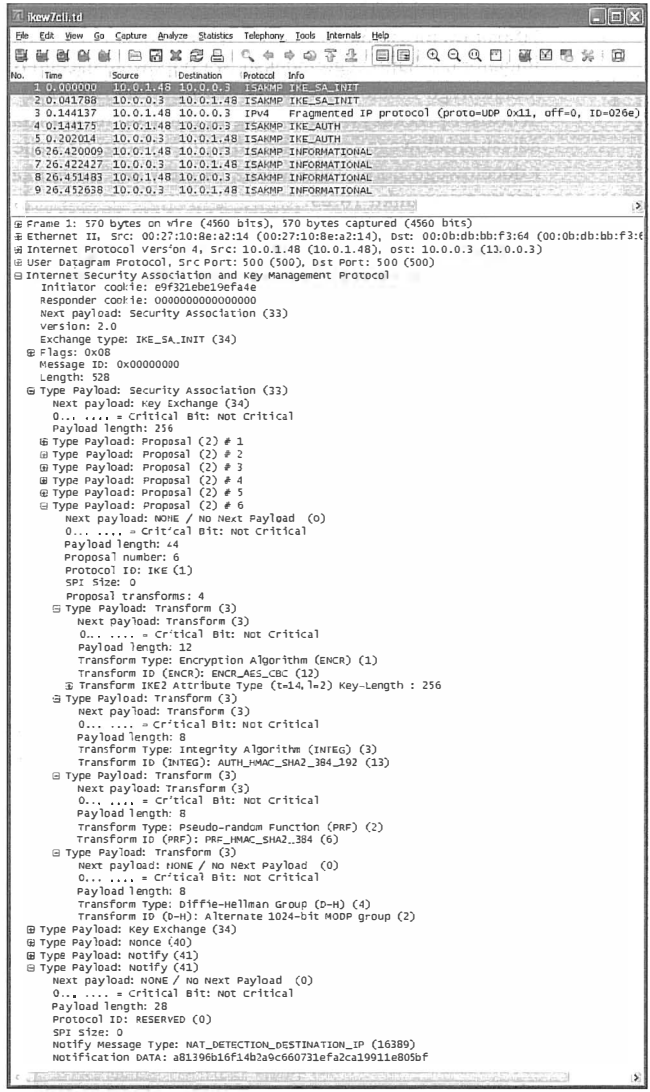
18.8.7. Przykład

Mechanizm IPsec jest implementowany w formie otwartego i firmowego oprogramowania w różnych systemach operacyjnych. Standardowy podsystem VPN Agile platformy Windows 7 obsługuje zarówno protokół IKE, jak i protokół MOBIKE. W systemach Linux obsługa komunikacji IPsec jest wbudowana w jądro od wersji 2.6. Ponadto dostępne są pakiety OpenSwan oraz StrongSwan, które pozwalają na pełne wdrożenie sieci VPN. W prezentowanym dalej przykładzie wykorzystano system Linux z usługą StrongSwan (o adresie IPv4 10.0.0.3) oraz jednostkę kliencką pracującą pod kontrolą systemu Windows 7 (o adresie IPv4 10.0.1.48). Do zademonstrowania protokołu IKE wykorzystano certyfikaty komputerów utworzone za pomocą algorytmu RSA. Początkowa wymiana IKE została przedstawiona na rysunku 18.22.

Analizując pierwszy rysunek, możemy zauważyć, że program Wireshark przedstawił wymianę IKE jako komunikację w protokole ISAKMP. Jest to przedawniona nazwa **internetowego protokołu zarządzania kluczami i relacjami zabezpieczeń** (ISAKMP — *Internet Security Association and Key Management Protocol*), który jest obecnie określany jako protokół IKE. Nagłówek IKE zawiera indeks SPI inicjatora (oznaczony jako *Initiator cookie*) oraz analogiczny indeks odpowiadającego (niewyznaczony w tym przypadku). Numer wersji ma wartość 2, co oznacza, że pakiet zawiera dane protokołu IKEv2, a typ wymiany to `IKE_SA_INIT`.

Po dokładniejszym przyjrzeniu się zarejestrowanym informacjom można stwierdzić, że przechwycony komunikat `IKE_SA_INIT` zawiera pięć pól danych: jedno pole *SA*, jedno pole *KE*, jedno pole wartości jednorazowych (*Nonce*) oraz dwa pola powiadomień (*Notify*). Pole *SA* składa się z sześciu propozycji, z których każda obejmuje wykaz transformat. Propozycje te odpowiadają zestawowi algorytmów, z których inicjator chciałby skorzystać. Propozycja 6. (ostatnia) została zaprezentowana bardziej szczegółowo. Zakłada ona wykorzystanie algorytmu AES w trybie CBC z kluczem szyfrowania o długości 256 bitów, ochronę integralności na bazie mechanizmu HMAC SHA-256, generowanie wartości losowych za pomocą funkcji PRF z grupy SHA-384 oraz wykorzystanie alternatywnej 1024-bitowej grupy MODP do uzgadniania kluczy DH. W innych propozycjach (nieprzedstawionych szczegółowo na rysunku) zawarte są sugestie użycia algorytmów 3DES i AES (o różnych długościach kluczy) do szyfrowania, SHA-1 do zapewnienia

Rysunek 18.22. *Na rysunku przedstawiamy początkową wymianę danych IKE z zaznaczonym pierwszym pakietem. Wymiana IKE_SA_INIT jest realizowana za pośrednictwem portu UDP o numerze 500 i obejmuje indeks SPI inicjatora, zestaw proponowanych algorytmów kryptograficznych, dane niezbędne do uzgodnienia klucza DH, wartość jednorazową oraz pola powiadomień przechowujące adresy mechanizmu NAT Traversal. Każda propozycja zapisana w polu danych SA jest żądaniem ustanowienia relacji IKE_SA z użyciem określonych transformat (odpowiadających za szyfrowanie, ochronę integralności, generowanie liczb losowych z zastosowaniem wskazanych funkcji PRF i uzgadnianie kluczy DH zgodnie z podanymi parametrami)*



integralności oraz inne warianty SHA jako funkcje PRF. Za polem danych *SA* znajduje się pole danych wymiany kluczy (*Key Exchange*), które zawiera publiczne dane potrzebne do uzgodnienia kluczy DH w ramach 1024-bitowej grupy MODP. Kolejne pole przechowuje 48-bajtowy ciąg bitów o losowej wartości jednorazowej. Dwa ostatnie pola danych są powiadomieniami wykorzystywanymi przez funkcję obsługi translacji adresów. Pierwszym z powiadomień jest pole typu `NAT_DETECTION_SOURCE_IP`, natomiast

drugie to NAT_DETECTION_DESTINATION_IP. Wartość zapisana w pierwszym odpowiada 20-bajtowemu skrótowi SHA-1, wyznaczonemu na podstawie: 8-batjtowego indeksu SPI inicjatora, 8-bajtowego indeksu SPI odpowiadającego (o wartości 0 w tym przypadku), 4-bajtowego źródłowego adresu IPv4 oraz 2-bajtowego numeru portu źródłowego UDP. Wartości przekazywane w drugim powiadomieniu różnią się jedynie tym, że zamiast adresu źródłowego i portu źródłowego wykorzystywane są wartości docelowego adresu i docelowego portu. Odpowiedź na pierwszy komunikat IKE_SA_INIT została przedstawiona na rysunku 18.23.

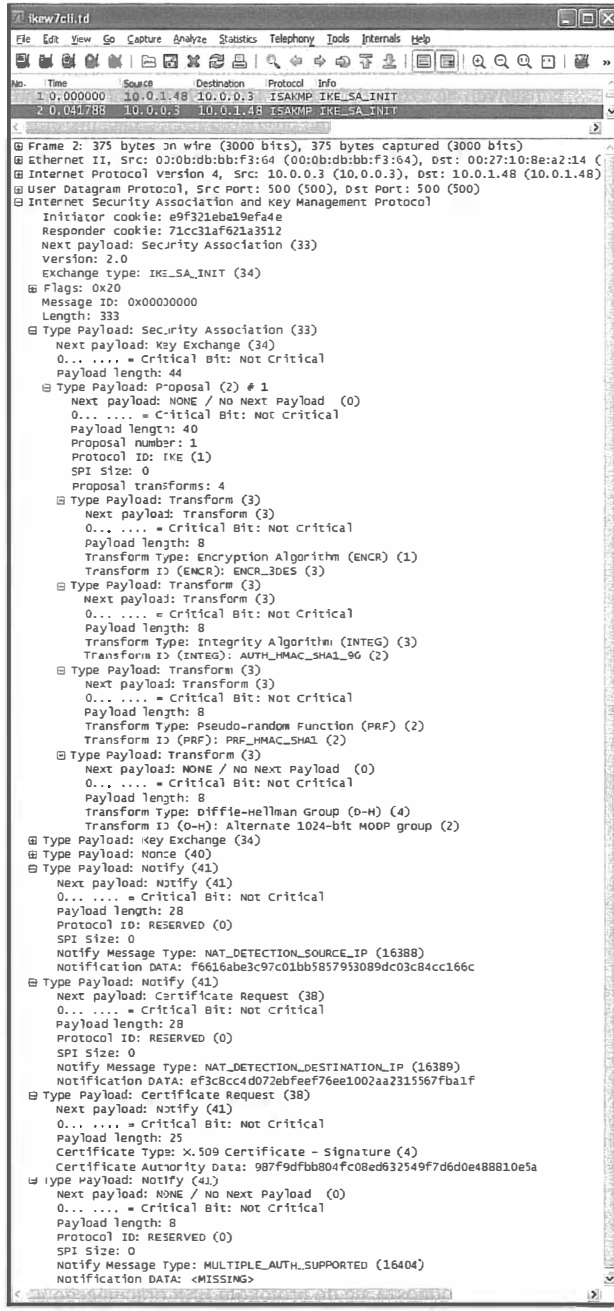
Przedstawiony na rysunku komunikat IKE_SA_INIT składa się z następujących pól danych: SA, KE, wartości jednorazowej (*Nonce*), trzech powiadomień (*Notify*) oraz żądania dostarczenia certyfikatu (*Certificate Request*). Pole *SA* przedstawia tylko jedną propozycję obejmującą następujące transformaty: 3DES (szyfrowanie), HMAC_SHA1_96 (kontrola integralności), HMAC_SHA1 (PRF) oraz grupa 2 algorytmu DH. Pole *KE* zawiera 128-bajtową wartość z 1024-bitowej grupy MODP. W polu wartości jednorazowej zapisany jest 32-bajtowy losowy ciąg bitów gwarantujących niepowtarzalność szyfrogramu. Dwa kolejne pola przenoszą powiadomienia NAT_DETECTION_SOURCE_IP oraz NAT_DETECTION_DESTINATION_IP (opisane wcześniej). Na końcu dołączone zostały dwa nowe pola danych: *CERTREQ* oraz *MULTIPLE_AUTH_SUPPORTED*.

Pole żądania certyfikatu (*CERTREQ*) wskazuje certyfikaty preferowane przez odpowiadającego. W tym przypadku odpowiadający informuje, że certyfikaty dostarczone później przez inicjatora powinny być podpisane przez określony urząd certyfikacji. Kodowanie wykorzystane do określenia jednostki CA jest jednym z kilku opisanych w sekcji 3.6 zalecenia [RFC5996]. Trzeba jednak pamiętać, że obecnie zestandaryzowane zostały jedynie formaty odpowiadające wartościom 4, 12 i 13. W omawianym przypadku przesyłana jest wartość 4, co oznacza, że pole składowe *Certificate Authority Data* przechowuje skróty SHA-1 kluczy publicznych (wartości *Subject Public Key Info* z certyfikatu X.509) zaufanych urzędów CA. Na podstawie długości pola (20 bajtów) można wnioskować, że został wskazany tylko jeden urząd certyfikacji. Wartość ta jest skrótem SHA-1 z klucza publicznego zapisanego w formacie DER o nazwie „Test CA" (przygotowanego na potrzeby omawianego przykładu).

Format **sprecyzowanych reguł kodowania** (DER — *Distinguished Encoding Rules*) jest jednym z elementów standardu **podstawowych reguł kodowania** (BER — *Basic Encoding Rules*) zdefiniowanego w notacji ASN.1. Zapis DER gwarantuje zakodowanie wartości w jeden niebudzący wątpliwości sposób. Jest jednocześnie jednym z formatów, które są najczęściej wykorzystywane do kodowania certyfikatów X.509. Innymi stosowanymi formatami zapisu są: PEM i ASCII (prezentowany wcześniej). Przekształcanie treści certyfikatów należy do zadań wielu narzędzi przetwarzania certyfikatów, w tym programu openssl.

Ostatnie z pól danych widocznych na rysunku 18.23 to powiadomienie (*Notify*) o treści *MULTIPLE_AUTH_SUPPORTED* bez dodatkowych danych. Jest to eksperymentalne rozszerzenie protokołu IKE, zdefiniowane w dokumencie [RFC4739], które sygnalizuje gotowość urządzenia do obsługi więcej niż jednej metody uwierzytelniania. Znajduje ono zastosowanie np. w przypadku wymiany IKE_AUTH z serwerem dostawcy usług internetowych, który umożliwia ustanowienie relacji IKE SA na podstawie certyfikatów oraz wymaga dodatkowo uwierzytelnienia użytkownika za pośrednictwem protokołu EAP.

Rysunek 18.23. *W komunikacie kończącym wymianę IKE_SA_INIT zapisane są: indeks SPI odpowiadającego (oznaczony jako cookie), pojedyncza propozycja ze stosownymi transformatami, parametry algorytmu DH, wartość jednorazowa oraz dane dotyczące procedury translacji adresów. Dodatkowo w komunikacie występuje pole danych CERTREQ żądania dostarczenia certyfikatów oraz powiadomienie informujące o obsłudze dwóch metod uwierzytelniania (o priorytecie odpowiadającym ich kolejności)*



Pozostałe pakiety widoczne na rysunku 18.23 zawierają zaszyfrowane komunikaty IKE_AUTH. Do ich transmisji wykorzystywany jest port 4500 zamiast 500, a w formowanych pakietach występuje „znacznik non-ESP" składający się z samych zer logicznych (zapisanych na czterech bajtach) [RFC3947], który informuje, że dany ruch jest strumieniem komunikatów IKE, a nie protokołu ESP. Znacznik oraz numery portów występują również w danych z omówionej wcześniej wymiany INFORMATIONAL.

Program Wireshark ma możliwość rozszyfrowywania pakietów IKE, jeśli zostanie uprzednio wyposażony w odpowiednie klucze i wartości SPI. Aby zobaczyć rozszyfrowaną treść pola danych IKE, wystarczy dostarczyć do programu plik dziennika z serwera IKE (wybierając w menu opcje *Edit/Preferences/Protocols/ISAKMP*; twórcy aplikacji są wierni pierwotnej nazwie protokołu, dlatego zamiast nazw IKE i TLS na ekranie widoczne są nazwy ISAKMP i SSL).

Trzeci z pakietów widocznych na rysunku 18.22 jest pierwszym fragmentem datagramu UDP/IP. Datagram ten został odtworzony po odebraniu przez program Wireshark drugiego fragmentu (w pakiecie 4.). Wynik tej operacji został z kolei pokazany na rysunku 18.24.

Na rysunku widoczna jest zawartość dwóch odtworzonych i rozszyfrowanych fragmentów datagramu UDP/IPv4, który stanowi pierwszy pakiet wymiany IKE_AUTH. Wynika z niego, że jednostka kliencka przesyłała następujące pola danych IKE: *IDi*, *CERT*, *CERTREQ*, *AUTH*, *N (MOBIKE_SUPP)*, *CP*, *SA*, *TSi* oraz *TSr*. Pole *IDi* zawiera nazwę inicjatora — *test client*. Pole danych *CERT* przechowuje natomiast certyfikat klienta podpisany przez urząd certyfikacji *Test CA*, który, jak wiadomo, jest akceptowany przez stację serwerową (z uwagi na odpowiednią konfigurację). Pole danych *CERTREQ* przechowuje żądanie uwzględniające urząd Test CA oraz 21 innych urzędów certyfikacji (niewidocznych na rysunku) znanych systemowi Windows 7, wykorzystywanemu w tym przykładzie. Pole danych *AUTH* zawiera blok danych podpisany przez inicjatora z użyciem klucza prywatnego RSA (patrz sekcja 2.15 zalecenia [RFC5996]). Jest on potwierdzeniem autentyczności źródła informacji. Sekcja *N(MOBIKE_SUPPORTED)* informuje o gotowości klienta do działania zgodnie z protokołem MOBIKE. Pole konfiguracji CP (*CFG_REQUEST*) (niepokazane szczegółowo) składa się z następujących atrybutów: INTERNAL_IP4_ADDRESS, INTERNAL_IP4_DNS, INTERNAL_IP4_NBNS oraz parametru PRIVATE_USE (o wartości 23456). Ustawienia te ułatwiają konfigurację łącza VPN i pełnią podobną rolę jak informacje dostarczane lokalnie z serwera DHCP (patrz rozdział 6.). Skrót NBNS odpowiada serwerowi nazw NetBIOS. Sam interfejs programistyczny NetBIOS jest implementowany w wielu mechanizmach obsługi sieci. Jest również standardowym składnikiem systemu Microsoft Windows.

Widoczne na rysunku 18.24 pole danych relacji SA uwzględnia wszystkie informacje niezbędne do utworzenia relacji CHILD_SA. Są w nim wymienione dwie propozycje (niepokazane szczegółowo), z których każda odnosi się do protokołu ESP o 32-bitowych indeksach SPI (w IKE wartości *SPI* były zapisywane na 64 bitach), z algorytmem weryfikacji integralności AUTH_HMAC_SHA1_96 i bez wykorzystania rozszerzonych numerów sekwencyjnych (wynikających z proponowanej transformaty). Pierwsza propozycja odnosi się do szyfrowania z zastosowaniem mechanizmu ENCR_AES_CBC (z 256-bitowymi kluczami). Druga natomiast sugeruje użycie algorytmu ENCR_3DES. Ponieważ nie zarejestrowano pola danych *N(USE_TRANSPORT_MODE)*, można przyjąć, że propozycje odnoszą się do protokołu ESP pracującego w domyślnym trybie tunelowym.

Rysunek 18.24. *Wymiana IKE_AUTH zawiera zaszyfrowane informacje przekazywane w protokole UDP na porcie 4500. Odtworzony na podstawie dwóch fragmentów komunikat IKE (zaszyfrowany i uwierzytelniony) składa się z następujących pól danych: identyfikacja inicjatora (IDi), certyfikat (CERT), żądanie certyfikatu (CERTREQ), uwierzytelnienie (AUTH), powiadomienie (N), konfiguracja (CP), relacja zabezpieczeń (SA), selektor ruchu inicjatora (TSi), selektor ruchu odpowiadającego (TSr)*

Przedstawione na rysunku 18.24 pola danych selektorów ruchu (*TSi* i *TSr*) wskazują zakresy adresów IPv4 i IPv6 akceptowalnych podczas tworzenia relacji SA. W sekcji *TSi* zapisane zostały parametry TS_IPv6_ADDR_RANGE oraz TS_IPv4_ADDR_RANGE, które definiują zakresy adresów i portów. Analogiczne wartości znajdują się w części *TSr* (niepokazanej szczegółowo na rysunku).

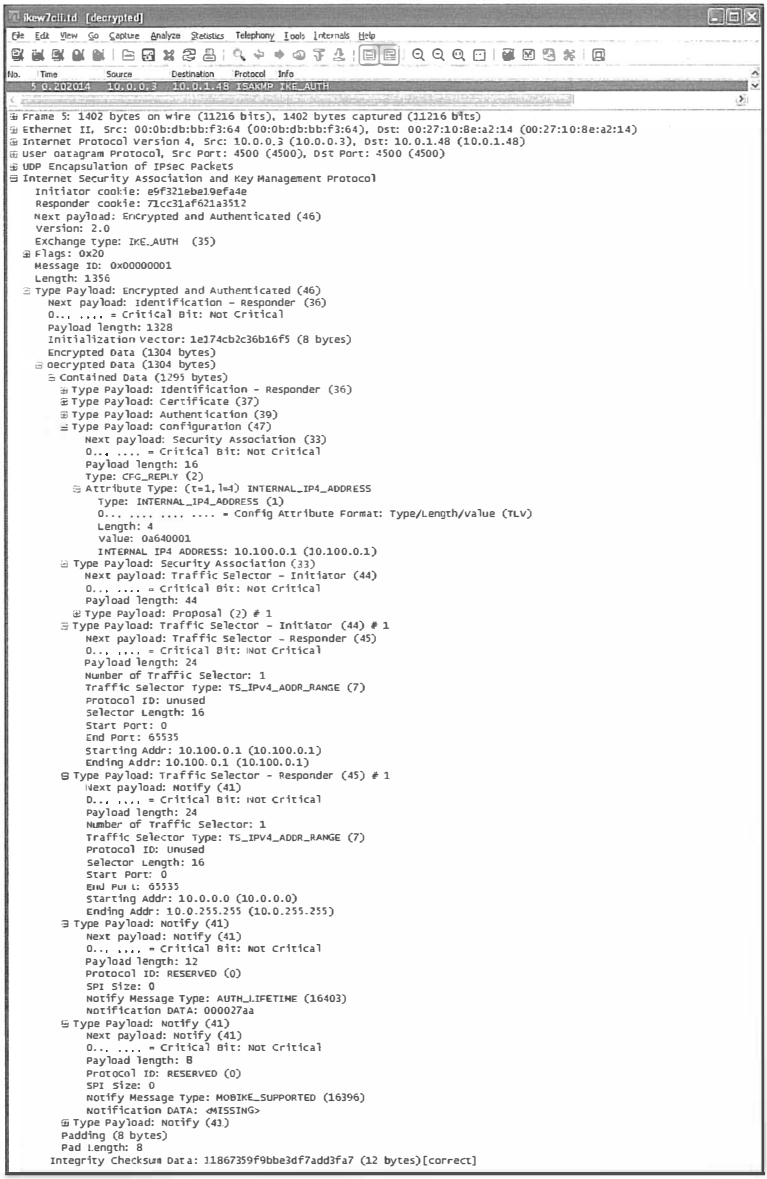
Pierwsza z omówionych wiadomości IKE_AUTH jest dość skomplikowana i wymaga do przekazania informacji więcej niż jednego 1500-bajtowego datagramu UDP/IPv4. Po jej przetworzeniu odpowiadający przygotowuje ostatni komunikat, którego treść została przedstawiona na rysunku 18.25.

Z rysunku wynika, że serwer odsyła odpowiedź z następującymi polami danych: *IDr*, *CERT*, *AUTH*, *CP(CFG_REPLAY)*, *SA*, *TSi* i *TSr*, *N(AUTH_LIFETIME)*, *N(MOBIKE_SUPPORTED)* oraz *N(NO_ADDITIONAL_ADDRESSES)*. Pole *IDr* służy do przekazania nazwy serwera zakodowanej zgodnie z formatem DER. Pole *CERT* dostarcza odpowiedni certyfikat serwera, a pole *AUTH* potwierdza dysponowanie odpowiednim kluczem prywatnym. Sekcja *CP(CFG_REPLAY)* obejmuje atrybut INTERNAL_IP4_ADDRESS przydatny w konfigurowaniu połączenia VPN. Wartość pola SA jest zbliżona do wartości pola klienckiego (prezentowanego na rysunku 18.24) i zawiera jedną propozycję z transformatami ENCR_AES_CBC (o 256-bitowym kluczu), AUTH_HMAC_SHA1_96 oraz brakiem rozszerzonych numerów sekwencyjnych.

Wartości *TSi* i *TSr* zostały „zawężone", aby odpowiadały wartościom ze znacznie mniejszego zakresu niż zawarte w klienckim komunikacie IKE_AUTH. W tym przypadku parametr *TSi* został zredukowany do pojedynczego adresu IPv4 o wartości 10.100.0.1. Natomiast parametr *TSr* odpowiada zakresowi 10.0.0.0/16. Obu selektorom odpowiadają pełne zakresy numerów portów 0 – 65535. Opisany przypadek zawężania jest jednym z najmniej kłopotliwych. W sytuacjach, gdy zawężanie prowadzi do wyznaczenia kilku nieciągłych podzbiorów wartości, konieczne jest wygenerowanie dodatkowych pól *N(ADDITIONAL_TS_POSSIBLE)*. Sama operacja zawężania służy do ustalenia wzajemnie akceptowalnych zakresów wartości w określonej relacji SA.

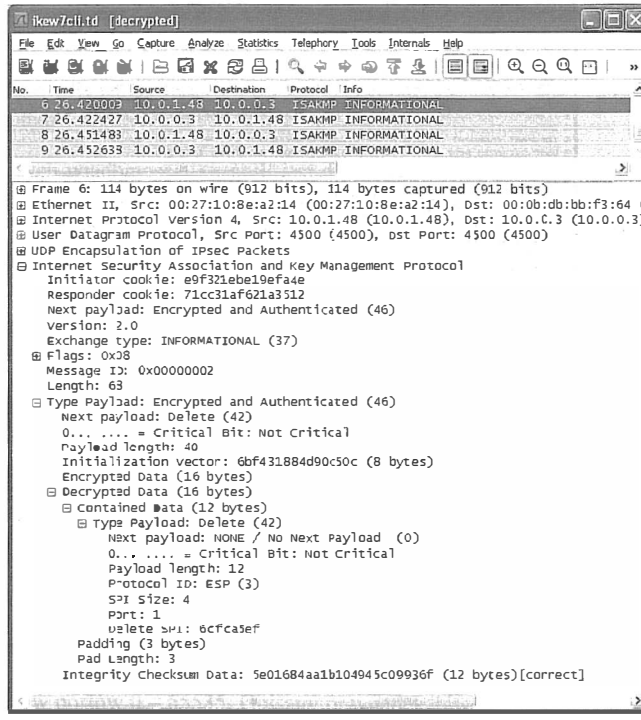
Pole danych *N(AUTH_LIFETIME)* informuje, że uwierzytelnienie pozwala na utrzymanie komunikacji nie dłużej niż przez 2,8 godziny (10 154 s, wyrażone jako 000027aa). Pole danych *N(MOBIKE_SUPPORTED)* sygnalizuje gotowość odpowiadającego do wykorzystania protokołu MOBIKE. Z kolei pole *N(NO_ADDITIONAL_ADDRESSES)* stanowi informację, że w protokole MOBIKE mogą być stosowane jedynie te same adresy, które zostały zdefiniowane w bieżącej wymianie.

Od tego momentu relację CHILD_SA z protokołem ESP pracującym w trybie tunelowym można uznać za ustanowioną, co pozwala na przenoszenie zasadniczego ruchu sieciowego. Sama transmisja pakietów ESP nie została tutaj szczegółowo omówiona (zagadnienie nie należy do szczególnie skomplikowanych). Ciekawe wydaje się natomiast zrywanie relacji *SA*. Za tę czynność odpowiadają dwie wymiany INFORMATIONAL zawierające pola *Delete* — jedno z nich odnosi się do relacji *SA* protokołu ESP, a drugie do relacji *SA* protokołu IKE. Treść żądania odpowiedzialnego za zakończenie relacji *SA* protokołu ESP została pokazana na rysunku 18.26.



Rysunek 18.25. *Zakończenie wymiany IKE_AUTH. Odpowiadający generuje zaszyfrowany i uwierzytelniony komunikat z następującymi polami: identyfikator odpowiadającego (IDr), CERT, AUTH, CP(CFG_REPLAY), SA, zawężone pola TSi i TSr wraz z N(AUTH_LIFETIME), N(MOBIKE_SUPPORTED) oraz N(NO_ADDITIONAL_ ADDRESSES). Po tej wymianie może zostać przesłany pierwszy komunikat w ramach relacji CHILD_SA*

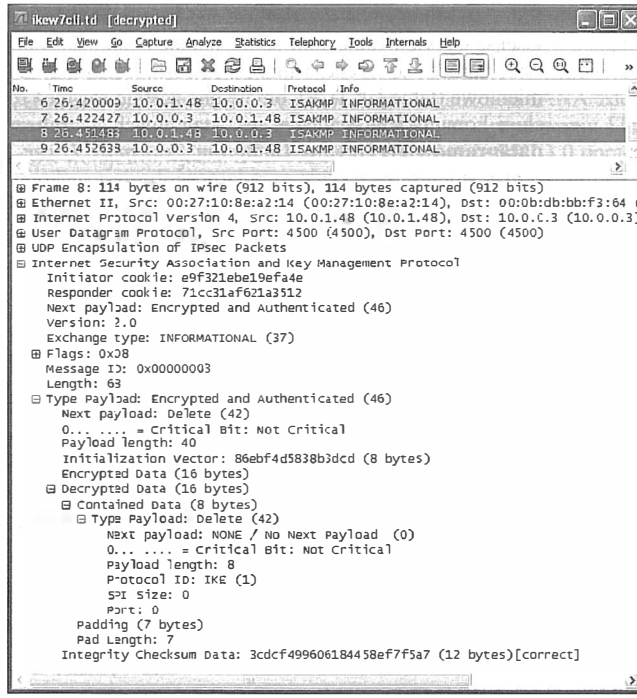
Rysunek 18.26. *Żądanie usunięcia potomnej relacji SA protokołu ESP z indeksem o wartości 6cfca5ef dostarczone w ramach relacji SA protokołu IKE. Pole danych Delete zawiera parametr Port: 1. Jest to błąd programu Wiershark (powinna to być liczba indeksów SPI: 1)*



Na rysunku widoczne są informacje na temat relacji SA usuwanej w odpowiedzi na żądanie zakończenia połączenia wygenerowane przez jednostkę kliencką. Podobnie jak w innych komunikatach IKE, obejmują one zaszyfrowany i uwierzytelniony blok danych. Blok ten składa się z jednego elementu — pola danych usuwania (*Delete*). Pole usuwania może przenosić więcej niż jeden indeks SPI objęty operacją. W tym przypadku jednak przechowuje pojedynczą wartość `0x6cfca5ef`. Pakiet o numerze 7, wygenerowany przez odpowiadającego, ma niemal identyczną treść, ale z innymi ustawieniami pola znaczników (*Flags*) (w inny sposób ustawione są bity inicjatora i odpowiadającego oraz żądania i odpowiedzi), inną wartością wektora IV, inną wartością sumy kontrolnej w polu ochrony integralności oraz inną wartością indeksu *SPI* (`c348faf2`) w polu danych usuwania.

Aby zakończyć relację *IKE_SA*, niezbędna jest wymiana kolejnych komunikatów typu `INFORMATIONAL`. Inicjator rozpoczyna ją od wysłania pakietu pokazanego na rysunku 18.27. Z jego treści wynika, że jest to żądanie zakończenia relacji *SA* protokołu IKE. Procedura kończy się odesłaniem przez odpowiadającego komunikatu IKE zawierającego puste, zaszyfrowane i uwierzytelnione pole danych (w pakiecie 9.), w którym wartość *Następne pole danych* wynosi zero (*NONE*). Oznacza to zakończenie procedury usuwania relacji *SA* protokołu IKE.

Rysunek 18.27. *Żądanie usunięcia relacji SA protokołu IKE. Wartości SPI nie są wymagane, ponieważ cały komunikat jest przenoszony w ramach relacji SA protokołu IKE i nie ma niejednoznaczności w kwestii tego, czego dotyczy*



18.9. Bezpieczeństwo warstwy transportowej (TLS i DTLS)

Dotychczasowe omówienie dotyczyło protokołów zabezpieczeń funkcjonujących na poziomie 2. i 3. warstwy modelu OSI. Jednak najpowszechniej stosowanym mechanizmem zabezpieczającym jest protokół ulokowany tuż nad warstwą transportową, nazywany **zabezpieczaniem warstwy transportowej** (TLS — *Transport Layer Security*). Mechanizm TLS odpowiada za ochronę komunikacji z serwerami WWW oraz wymiany danych w kilku innych popularnych protokołach, włącznie z POP i IMAP (których zabezpieczone wersje są nazywane odpowiednio POP3S oraz IMAPS). Jedną z przyczyn upowszechnienia się rozwiązań TLS jest możliwość implementowania ich w aplikacji lub bezpośrednio pod warstwą aplikacji, ale powyżej protokołów niższych warstw. Tymczasem rozwiązania, takie jak EAP i IPsec, wymagają zazwyczaj wsparcia ze strony systemu operacyjnego oraz implementacji w urządzeniach sieciowych.

Protokół TLS oraz jego poprzednik — **warstwa bezpiecznych gniazd** (SSL — *Secure Sockets Layer*) [RFC6101] — występują w wielu wersjach. W dalszej części książki zostanie omówiona wersja TLS 1.2 [RFC5246], która obowiązywała w trakcie przygotowywania tej publikacji. Mechanizm TLS 1.2 współdziała z większością wcześniejszych wersji protokołów TLS i SSL (np. z TLS 1.0, 1.1 oraz SSL 3.0). Wyjątkiem jest protokół

SSL 2.0. Mimo że on również jest zgodny z pozostałymi rozwiązaniami, jego niedoskonałości są na tyle istotne, że współdziałanie z nim jest zabronione [RFC6176]. W rozdziale tym, oprócz zaprezentowania mechanizmu TLS 1.2 zabezpieczającego protokoły strumieniowe (takie jak TCP), omówione zostało również rozwiązanie przeznaczone do obsługi komunikacji bazującej na datagramach — **zabezpieczenie warstwy transportu datagramów** (DTLS — *Datagram Transport Layer Security*) [RFC4347]. Protokół DTLS stopniowo zyskuje popularność, bo jest wykorzystywany w połączeniach VPN, które nie wymagają stosowania rozwiązań IPsec. Jego specyfikacja bazuje na standardzie TLS 1.1 [RFC4346], choć trwają prace nad jej aktualizacją [IDDTLS].

18.9.1. TLS 1.2

Cele stosowania mechanizmu TLS nie odbiegają od tych, które obowiązują w przypadku użycia protokołów z grupy IPsec. Różnica polega jednak na tym, że TLS działa w wyższej warstwie. Poufność oraz integralność danych jest zapewniana dzięki wykorzystaniu różnorodnych zestawów algorytmów kryptograficznych oraz certyfikatów tworzonych w ramach systemu PKI. Rozwiązania TLS można stosować również do zabezpieczania połączeń między dwiema nieznanymi sobie jednostkami (bez użycia certyfikatów), ale działania tego typu są narażone na ataki typu MITM (nic w tym dziwnego, żadna ze stron nie jest wówczas w pełni zidentyfikowana). Sam protokół TLS jest podzielony na dwie warstwy — **warstwę rekordów** (*record layer*) oraz **wyższą warstwę** (*upper layer*). W warstwie rekordów implementowany jest protokół Record, do którego przenoszenia wykorzystywany jest niezawodny protokół niższej warstwy (np. TCP). Struktura systemu TLS została przedstawiona na rysunku 18.28.

Rysunek 18.28. *Stos protokołów mechanizmu TLS złożony z warstwy rekordów oraz trzech protokołów wyższej warstwy, nazywanych protokołami uzgadniania połączeń. Czwartym protokołem wyższej warstwy jest protokół aplikacji wykorzystujący system TLS. Warstwa rekordów obsługuje fragmentację, kompresję, ochronę integralności oraz szyfrowanie danych. Protokoły uzgadniania połączeń realizują zadania podobne do protokołu IKE w stosie IPsec*

TLS jest protokołem komunikacyjnym typu klient-serwer. Został zaprojektowany z myślą o zapewnieniu bezpieczeństwa w połączeniach między dwiema aplikacjami. Wykorzystywany w tym rozwiązaniu protokół Record odpowiada za fragmentację, kompresję, ochronę integralności oraz szyfrowanie danych wymienianych między klientami i serwerami. Z kolei protokoły uzgadniania połączeń (*handshake protocols*) odpowiadają za wymianę identyfikatorów, uwierzytelnianie stron, generowanie alarmów i dostarczanie do protokołu Record kluczy obowiązujących w danym połączeniu. Grupa protokołów uzgadniania połączeń składa się z czterech elementów: protokołu nawiązywania połączenia (*Handshake*), protokołu alarmowego (*Alert*), protokołu zmiany parametrów szyfrowa-

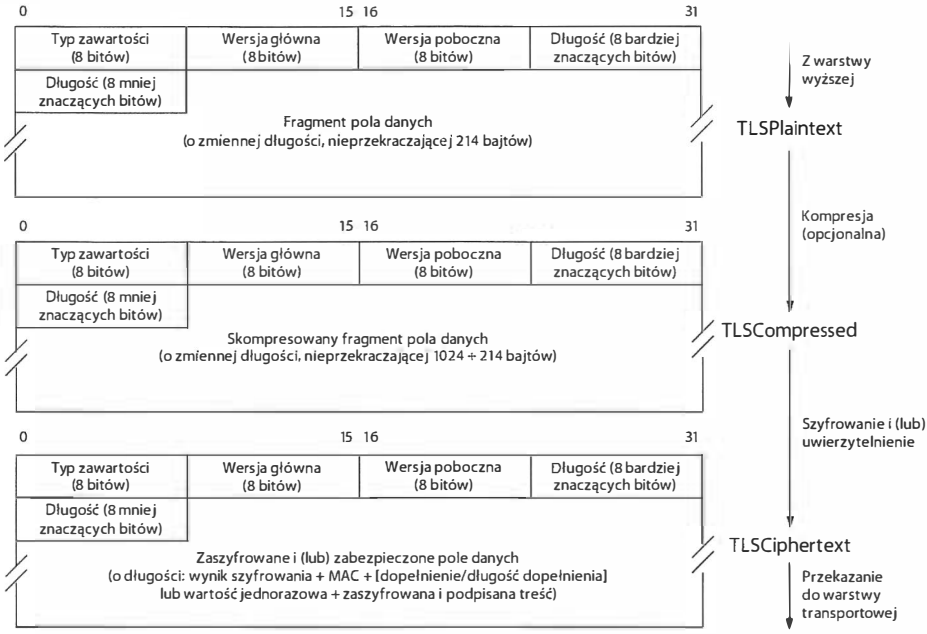
nia (*Change Cipher Spec*) oraz protokołu danych aplikacji. Podobnie jak w standardowym IPsec, rozwiązanie TLS cechuje się dużą elastycznością, co umożliwia współdziałanie z obecnymi i przyszłymi zestawami algorytmów kryptograficznych, które w terminologii TLS są nazywane **zestawami szyfrów** (CS — *Cipher Suit*). Organizacja IANA zdefiniowała wiele zestawów tego typu. Ich pełna lista znajduje się w opracowaniu [TLSPARAMS]. Najnowsze wersje rozwiązań TLS bazują na standardzie SSL 3.0 opracowanym przez firmę Netscape. I choć protokoły TLS i SSL nie mogą bezpośrednio współdziałać, zdefiniowano techniki negocjacyjne, które umożliwiają klientom i serwerom dynamiczne wykrywanie rodzaju używanego protokołu i wybranie odpowiedniego podczas nawiązywania połączenia.

Protokół Change Cipher Spec służy do zmiany bieżących parametrów pracy. Jego działanie poprzedza użycie protokołu Handshake, który ustala „oczekujący" stan połączenia. Potem następuje zmiana bieżącego stanu na stan oczekujący. Wspomniana zmiana jest możliwa tylko w przypadku, w którym przyszły stan został odpowiednio przygotowany. Działanie systemu TLS bazuje na pięciu operacjach kryptograficznych: podpisie cyfrowym, szyfrowaniu strumieniowym, szyfrowaniu blokowym, uwierzytelnianiu z szyfrowaniem (AEAD) oraz szyfrowaniu kluczy publicznych. W celu ochrony integralności danych warstwa rekordów stosu TLS wykorzystuje algorytmy HMAC. Za generowanie kluczy w wersji 1.2 odpowiadają funkcje PRF wspomagane przez algorytm HMAC SHA-256. W połączeniu TLS możliwe jest również użycie opcjonalnego algorytmu kompresji, którego parametry są negocjowane podczas nawiązywania połączenia.

18.9.1.1. Protokół Record

Protokół Record bazuje na rozszerzalnym zbiorze wartości opisujących typy rekordów. Na ich podstawie identyfikuje rodzaje komunikatów podlegających multipleksacji (tj. wyróżnia dane protokołów wyższych warstw). Przez cały czas pracy moduł obsługi protokołu Record utrzymuje informacje o **bieżącym stanie połączenia** oraz **oczekującym stanie połączenia**. Każdy zestaw parametrów opisujących stan połączenia jest dodatkowo podzielony na grupy parametrów odnoszących się do stanu odczytu oraz stanu zapisu. Każda z grup opisuje algorytm kompresji, algorytm szyfrowania oraz algorytm MAC wykorzystywane w komunikacji, a także odpowiadające im klucze i dodatkowe ustawienia. Zmiana klucza wymaga przygotowania stanu oczekującego, do czego używany jest protokół Handshake. Sama synchronizacja stanów jest jednak najczęściej realizowana za pomocą protokołu Cipher Change. Jego zadaniem jest ustanowienie stanu oczekującego stanem bieżącym. Po zainicjowaniu mechanizmu w obu stanach wyznaczany jest algorytm szyfrowania NULL, wyłączona jest kompresja danych oraz przetwarzanie MAC.

Działanie protokołu Record zostało zobrazowane na rysunku 18.29. Sprowadza się ono do dzielenia (fragmentowania) bloków danych pozyskanych z wyższych warstw na rekordy nazywane tekstowymi rekordami TLS (*TLSPlaintext*). Każdy z takich rekordów może mieć rozmiar nie większy niż 2^{14} bajtów (choć zazwyczaj jest mniejszy). Wybór rozmiaru bloku wynika z ustawień mechanizmu TLS (rozmiary komunikatów warstwy wyższej nie są zachowywane). Rekordy tekstowe po utworzeniu podlegają kompresji [RFC3749] zgodnie z algorytmem zdefiniowanym w bieżącym stanie połączenia. W danej chwili



Rysunek 18.29. *Działanie warstwy rekordów TLS rozpoczyna się od przetwarzania rekordu TLSPlaintext, który po zastosowaniu algorytmu kompresji bezstratnej jest przekształcany w rekord TLSCompressed. Ten z kolei podlega szyfrowaniu (oraz uzupełnieniu o skrót MAC), którego wynikiem jest gotowy do transmisji rekord TLSCiphertext. Tradycyjne szyfry strumieniowe i blokowe wymagają dostępności algorytmu MAC. W szyfrowaniu blokowym konieczne jest również dopełnienie danych. W szyfrogramach AEAD uwzględniane są również wartości jednorazowe towarzyszące zaszyfrowanej i zabezpieczonej (przed modyfikacjami) treści. Nie są jednak wykorzystywane oddzielne wartości MAC*

zawsze aktywny jest jeden mechanizm kompresji, nawet jeśli jest to (a zazwyczaj tak jest) mechanizm NULL, który — oczywiście — nie wykonuje żadnej kompresji danych. Algorytm kompresji przekształca rekord TLSPlaintext w rekord skompresowany (*TLSCompressed*). Dozwolone są jedynie bezstratne mechanizmy kompresji, które gwarantują generowanie danych wynikowych nie większych od zbioru danych wejściowych o więcej niż jeden 1 kB. Ochronę pola danych przed ujawnieniem i modyfikacją zapewniają algorytmy szyfrowania i ochrony integralności, które przekształcają rekordy TLSCompressed w rekordy szyfrogramu (*TLSCiphertext*) dostarczane do warstwy transportowej.

Przygotowanie zaszyfrowanego rekordu zgodnie z rysunkiem 18.29 wymaga wyznaczenia numeru sekwencyjnego (bez zapisywania go w wiadomości), wyliczenia wartości MAC (jeśli jest wymagana), a następnie przeprowadzenia szyfrowania symetrycznego. Przed zaszyfrowaniem wiadomość może zostać dopełniona (maksymalnie 255 bajtami) do pełnego bloku, którego rozmiar jest narzucony przez zastosowany algorytm (w przypadku szyfrów blokowych). W rozwiązaniach AEAD (np. CCM lub GCM), zapewniających szyfrowanie i uwierzytelnianie jednocześnie, wyznaczanie wartości MAC nie jest konieczne. Niezbędna jest jednak wartość jednorazowa.

Klucze wykorzystywane w protokole Record są wyznaczane na podstawie **nadrzędnej tajnej wartości** (*master secret*), definiowanej poza samym protokołem Record (najczęściej w ramach protokołu Handshake). Nadrzędna tajna wartość wraz wartościami losowymi, dostarczonymi przez klienta i serwer na początku połączenia, umożliwia wygenerowanie następujących kluczy:

$M_C \mid M_S \mid D_C \mid D_S \mid IV_C \mid IV_S$ = PRF (nadrzędna_tajna_wartość, „rozszerzenie klucza", wartość_losowa_serwera + wartość_losowa_klienta)

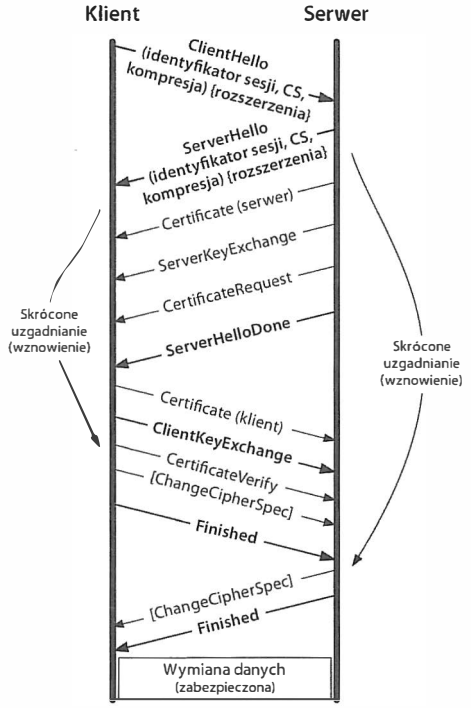
W powyższym równaniu operator | oznacza podział, a operator + odpowiada za konkatenację. Parametr M_C reprezentuje klucz MAC zapisu po stronie klienta. M_S odpowiada kluczowi MAC zapisu po stronie serwera. D_C jest kluczem zapisu danych klienta. D_S to klucz zapisu po stronie serwera. Natomiast parametry IV_C i IV_S reprezentują odpowiednio wektor IV klienta i wektor IV serwera. Zastosowanie operatora | sprawia, że funkcja PRF musi generować określoną liczbę bajtów. Wartości MAC, kluczy szyfrowania oraz kluczy IV (jeśli są wykorzystywane) mają stałą długość zależną od wybranego zestawu szyfrów. Ostatnie dwie z wymienionych wartości są używane tylko w przypadkach, w których potrzebne są wartości jednorazowe, czyli w szyfrach AEAD (patrz sekcja 3.2.1 zalecenia [RFC5116]). Zgodnie z dokumentem [RFC5246] zestawem szyfrującym wymagającym największej ilości danych tego typu jest AES_256_CBC_SHA256. Do jego działania niezbędne są cztery 32-bajtowe klucze o łącznej długości 128 bajtów.

18.9.1.2. Protokoły uzgadniania połączenia

W standardzie TLS zdefiniowano trzy podprotokoły, które realizują takie samo zadanie, jakie wykonuje protokół IKE w rozwiązaniu IPsec. Każdy z nich jest oznaczony numerem weryfikowanym podczas multipleksacji i demultipleksacji na poziomie warstwy rekordów. Do grupy wspomnianych protokołów należą: protokół Handshake (22), protokół Alert (21) oraz protokół Cipher Change (20). Działanie protokołu Cipher Change jest wyjątkowo proste. Polega na przesyłaniu tylko jednego komunikatu składającego się z pojedynczego bajtu o wartości 1. Celem transmisji jest powiadomienie odległej jednostki o zmianie stanu bieżącego na oczekujący. Odebranie powiadomienia powoduje uczynienie oczekującego stanu odczytu stanem bieżącym. Jednocześnie jest sygnałem dla warstwy rekordów o konieczności niezwłocznego przejścia z bieżącego stanu zapisu do oczekującego stanu zapisu. Komunikat ten jest używany zarówno przez klienta, jak i przez serwer.

Protokół Alert odpowiada za dostarczanie informacji statusowych między stronami połączenia TLS. Obejmuje powiadomienia o zakończeniu połączenia (celowym lub wynikającym z błędu) oraz o stanie niezwiązanym z żadnym problemem transmisyjnym. W zaleceniu [RFC5246] zdefiniowane zostały 24 komunikaty tego typu. Więcej niż połowa z nich opisuje błędy (np. niepoprawną wartość MAC, brakującą lub nieznaną wiadomość, niewłaściwe działanie algorytmu).

Protokół Handshake definiuje parametry potrzebne do utrzymania połączenia. Obsługuje komunikację między jednostkami TLS w zakresie: uzgadniania algorytmów, wymiany liczb losowych wykorzystywanych do generowania kluczy szyfrowania symetrycznego, wyznaczania parametrów pracy algorytmów, wymiany certyfikatów oraz wzajemnego uwierzytelnienia, generowania hasła sesji, dostarczenia parametrów zabezpieczeń do warstwy rekordów oraz sprawdzenia, czy wszystkie operacje zostały poprawnie wykonane. Obowiązkowe komunikaty zostały przedstawione na rysunku 18.30.



Rysunek 18.30. *Standardowa wymiana inicjująca połączenie TLS składa się z kilku następujących po sobie komunikatów. Obowiązkowe powiadomienia zostały oznaczone za pomocą pogrubionych strzałek i pogrubionej czcionki. W przypadku wznawiania wcześniejszego połączenia wykonywana jest skrócona procedura uzgadniania połączenia, która nie obejmuje uwierzytelniania punktów końcowych, kosztownego dla jednostek o małej wydajności obliczeniowej*

Uzgadnianie połączenia rozpoczyna się od wymiany komunikatów z grupy Hello. Pakiet ClientHello jest zazwyczaj pierwszą wiadomością przesyłaną z jednostki klienckiej do serwera. Zawiera identyfikator sesji, propozycję zestawu szyfrów (oznaczoną na rysunku jako CS) oraz zestaw akceptowanych algorytmów kompresji (ograniczający się najczęściej do wartości NULL, choć w zaleceniu [RFC3749] zdefiniowano również algorytm DEFLATE). Liczba opcji zestawu szyfrów obsługiwana w protokole TLS przekracza 250 [TLSPARAMS].

Komunikat ClientHello zawiera również numer wersji protokołu TLS oraz liczbę losową oznaczaną jako *ClientHello.random*. Po odebraniu komunikatu ClientHello serwer sprawdza, czy dostarczony stosowny identyfikator sesji jest zapisany w pamięci podręcznej. Jeśli tak, możliwe jest kontynuowanie ustanowionego wcześniej połączenia (nazywane „wznowieniem"), co się wiąże ze **skróconym uzgodnieniem** połączenia. Wykorzystanie skróconego uzgadniania jest kluczowym czynnikiem zwiększonej wydajności algorytmu. Pozwala na wyeliminowanie okresowego weryfikowania tożsamości każdej ze stron. Wymaga jednak synchronizacji zgodnej z zaleceniami specyfikacji algorytmu szyfrującego. Pierwsza fazy wymiany danych kończy się wysłaniem komunikatu ServerHello,

który przenosi liczbę losową serwera (*ServerHello.random*) oraz identyfikator sesji. Jeśli wartość identyfikatora odpowiada wartości klienckiej, jest to sygnał o próbie wznowienia połączenia przez serwer. W przeciwnym przypadku jest ona równa zeru, co oznacza konieczność przeprowadzenia pełnej operacji uzgodnienia połączenia.

Wykonanie pełnego uzgodnienia polega na wymianie kilku wiadomości z grupy Hello, które gwarantują uzgodnienie zestawu szyfrów, algorytmów kompresji oraz wartości losowych po każdej stronie połączenia. Serwer wybiera zestaw szyfrów spośród propozycji dostarczonych przez klienta. Do serwera może również zostać skierowane żądanie przekazania parametrów łańcucha certyfikacji (w komunikacji Certificate), które podlegają weryfikacji po stronie klienta (co jest typowym działaniem w połączeniach HTTPS). Serwer może również przesłać komunikat ServerKeyExchange oznaczający, że certyfikat nie został podpisany, nie ma certyfikatu lub do wygenerowania kluczy sesji powinien zostać użyty tymczasowy klucz.

Komunikat ServerKeyExchange jest wysyłany tylko w przypadkach, w których serwerowy komunikat Certificate nie zawiera wszystkich informacji potrzebnych do wyznaczenia wstępnej tajnej wartości. Wówczas uzgadnianie kluczy DH ma charakter anonimowy lub tymczasowy (mechanizmy szyfrowania pracują w trybie TLS_DHE_anon, TLS_DHE_DSS, TLS_DHE_RSA). Komunikat ServerKeyExchange *nie jest wykorzystywany* w innych zestawach, włącznie z TLS_RSA, TLS_DH_DSS lub TLS_DH_RSA.

Na tym etapie komunikacji serwer może wymagać uwierzytelnienia klienta. Jeśli tak jest, generuje komunikat CertificateRequest. Po tej operacji kończy drugą część wymiany informacji, wysyłając obowiązkową wiadomość ServerHelloDone. Klient, odbierając ten (prawdopodobnie potokowy) komunikat serwera, może być zobowiązany do poświadczenia swojej tożsamości (tzn. do wykazania, że posiada klucz odpowiadający certyfikatowi). W takim przypadku przesyła certyfikat za pomocą komunikatu Certificate i przy zastosowaniu takiego samego formatowania, z jakiego korzystał serwer. Następnie dostarcza obowiązkową wiadomość ClientKeyExchange, której treść zależy od rodzaju wykorzystywanego zestawu kryptograficznego. Zazwyczaj jednak obejmuje ona klucz zaszyfrowany w algorytmie RSA lub parametry algorytmu Diffiego-Hellmana, które umożliwiają wyznaczenie wartości zaburzającej potrzebnej podczas generowania nowych kluczy (nazywanej **wstępną tajną wartością** [*premaster secret*]). W końcu generuje komunikat CertificateVerify, który potwierdza posiadanie klucza prywatnego odpowiadającego dostarczonemu wcześniej certyfikatowi (jeśli serwer zażąda uwierzytelnienia klienta). Wiadomość ta zawiera sygnaturę wszystkich komunikatów z procesu uzgadniania połączenia, które klient otrzymał i wysłał do danej chwili.

Ostatni etap wymiany obejmuje komunikat ChangeCipherSpec należący do niezależnego protokołu TLS (z technicznego punktu widzenia nie jest to wiadomość protokołu Handshake). Niemniej jednak obowiązkowy komunikat Finished protokołu Handshake może zostać wysłany jedynie po uprzedniej wymianie komunikatów ChangeCipherSpec. Wiadomość Finished jest pierwszą, która jest chroniona zgodnie z ustawieniami uzgodnionymi wcześniej i zawiera dane do zweryfikowania. Treść weryfikowanego pola jest generowana w następujący sposób:

```
weryfikowane_dane = PRF(nadrzędna_tajna_wartość, tekst_zakończenia,
Skrót(komunikaty_Handshake))
```

Wartość zmiennej `tekst_zakończenia` to `client finished` po stronie klienta oraz `server finished` po stronie serwera. Dobór funkcji skrótu zależy od funkcji PRF wybieranej podczas początkowej wymiany Hello. Standard TLS 1.2 umożliwia definiowanie bloku danych do weryfikacji o dowolnym rozmiarze, ale wszystkie wcześniejsze wersje oraz obowiązujące obecnie zestawy szyfrów generują 12-bajtowy ciąg testowy. Wartość nadrzędna_tajna_wartość (składająca się z 48 bajtów) jest wyznaczana w następujący sposób:

```
nadrzędna_tajna_wartość = PRF(wstępna_tajna_wartość, „master secret",
ClientHello.random + ServerHello.random)
```

Znak (+) reprezentuje w tym przypadku operację konkatenacji. Komunikat Finished jest bardzo istotny, ponieważ pozwala ustalić z bardzo dużym prawdopodobieństwem, czy protokół Handshake poprawnie wykonał swoje zadanie oraz czy można rozpocząć wymianę danych.

18.9.1.3. Rozszerzenia TLS

Z porównania funkcji protokołów IKE i TLS wynika, że mechanizm IKE ma zdolność przenoszenia informacji wykraczających poza zakres niezbędny do ustanowienia podstawowej relacji SA. Odpowiadają za to pola danych powiadomienia i konfiguracji. Aby uzyskać podobną rozszerzalność mechanizmu TLS, można włączyć do wiadomości TLS 1.2 różnorodne **rozszerzenia**. Podstawowa specyfikacja protokołu TLS 1.2 [RFC5246] obejmuje rozszerzenie „algorytmów podpisu", które klient wykorzystuje do powiadomienia serwera o obsługiwanych mechanizmach skrótu i podpisu cyfrowego (zgodnie ze standardem dozwolone są następujące algorytmy skrótu: MD5, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 oraz następujące algorytmy podpisu cyfrowego: RSA, DSA, ECDSA). Lista technik definiowana jest w formie par ułożonych w kolejności od preferowanej do najmniej pożądanej (ponieważ część systemów obsługuje tylko wybrane kombinacje). Aktualne zestawienie rozszerzeń jest podane w dokumencie [TLSEXT].

W poprzednich wersjach protokołu TLS zdefiniowanych zostało sześć rozszerzeń. Listę obowiązującą w standardzie TLS 1.2 uzupełniono (w zaleceniu [RFC6066]) o następujące pozycje: `server_name` (nazwa domenowa serwera), `max_fragment_length` (maksymalny rozmiar wiadomości wyrażony jako 2^n bajtów, przy n wynoszącym od 9 do 12), `client_certificate_url` (powiadomienie o obsłudze komunikatu CertificateURL, który pozwala na przekazanie adresu URL certyfikatu zamiast jego treści), `trusted_ca_keys` (skórty kluczy publicznych i (lub) certyfikatów zaufanych urzędów CA), `truncated_hmac` (wykorzystanie jedynie pierwszych 80 bitów do wyznaczania wartości HMAC) oraz `status_request` (żądanie użycia przez serwer protokołu OCSP i dostarczenia odpowiedzi w formacie DER w ramach komunikatu CertificateStatus, który jest wysyłany podczas weryfikowania certyfikatu na etapie uzgadniania połączenia). Każde z wymienionych rozszerzeń może zostać dołączone do (rozszerzonego) komunikatu ClientHello, a w pewnych okolicznościach może również wystąpić w komunikacie ServerHello (w celu potwierdzenia uzgodnienia). Poza rozszerzeniami i dwoma opisanymi wcześniej komunikatami protokołu Handshake specyfikacja [RFC6066] definiuje również cztery komunikaty alarmowe: `certificate_unobtainable` (niemożliwe uzyskanie certyfikatu), `unrecognized_name` (nierozpoznana nazwa), `bad_certificate_status_response` (błędna odpowiedź opisująca status certyfikatu) oraz `bad_certificate_hash_value` (błędna wartość skrótu certyfikatu). Powiadomienia te nie są wysyłane, jeśli klient nie zasygnalizuje wcześniej obsługi rozszerzonych komunikatów ClientHello.

Specyfikacje protokołu uzupełnia jeszcze kilka dodatkowych rozszerzeń, z których część jest zarezerwowana. Przykładowo w dokumencie [RFC4681] opisano rozszerzenie `user_mapping` przeznaczone do określania kontekstu użycia identyfikatora użytkownika (np. w domenie Windows). Inne rozszerzenie zmienia rozszerzenie `cert_type` w taki sposób, aby obejmowało ono poza certyfikatami X.509 również certyfikaty OpenPGP [RFC6091]. Poza tym w dokumencie [RFC4492] opisano zestawy szyfrów bazujące na krzywych eliptycznych. Opracowanie [RFC5054] zawiera opis użycia w środowisku TLS **protokołu bezpiecznych haseł zdalnych** (SRP — *Secure Remote Password*). W zaleceniu [RFC5764] przedstawiono wykorzystanie rozszerzenia `use_srtp`, które zostało zaprojektowane jako odmiana **bezpiecznego protokołu transmisji w czasie rzeczywistym** (SRTP — *Secure Real-time Transport Protocol*) bazująca na mechanizmie DTLS (patrz punkt 18.9.2). Rozszerzenie `SessionTicket` [RFC5077] definiuje bilety sesji, które eliminują konieczność przechowywania przez serwer informacji o stanie połączenia, niezbędnych podczas wznawiania sesji. Jego działanie polega na zapisywaniu wspomnianych informacji w zaszyfrowanej formie po stronie klienta. Bardzo ważne jest również rozszerzenie `renegotiation_info`, które usuwa lukę w zabezpieczeniach związaną z operacją renegocjacji. Problem ten został opisany w kolejnym podpunkcie.

18.9.1.4. Renegocjacja

Protokół TLS umożliwia ponowne negocjowanie parametrów kryptograficznych z zachowaniem dotychczasowego połączenia. Procedurę renegocjacji może zainicjować serwer lub klient. Jeśli stroną inicjującą jest serwer, wysyła komunikat HelloRequest, a klient odpowiada za pomocą nowego komunikatu ClientHello. Analogiczny komunikat ClientHello może zostać spontanicznie wysłany przez klienta, bez konieczność wcześniejszego powiadomienia serwera o takim zamiarze.

Obsługa renegocjacji jest opcjonalna, ale „mocno zalecana". Okazuje się niezbędna w sytuacji przepełnienia liczników numerów sekwencyjnych. Odmowa przeprowadzenia renegocjacji jest sygnalizowana przez strony komunikacji za pomocą komunikatu alarmowego `no_renegotiation` (typu 100). Choć odebranie tego rodzaju alarmu nie musi oznaczać zerwania połączenia, może do tego doprowadzić, jeśli tak stanowi lokalna polityka bezpieczeństwa.

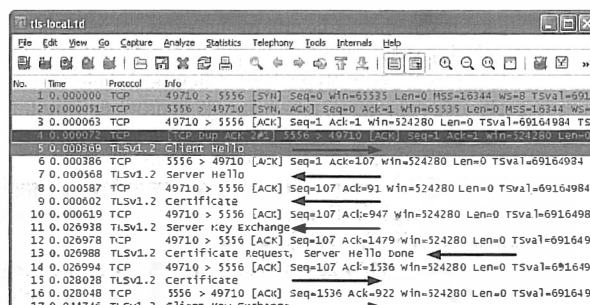
W roku 2009 zademonstrowano udany atak na protokół TLS właśnie z wykorzystaniem funkcji renegocjacji. Szczegółowy opis problemu został zamieszczony w punkcie 18.12. Luka umożliwiła atakującym ustanowienie sesji TLS z serwerem, która później w wyniku ataku MITM została złączona z prawidłową sesją kliencką. Po stronie serwera zauważono jedynie wykonanie zgodnej ze standardem operacji renegocjacji. Rozwiązanie problemu zostało opisane w dokumencie [RFC5746]. Polega ono na ściślejszym powiązaniu procedury renegocjacji z istniejącą sesją za pomocą rozszerzenia TLS o nazwie `renegotiation_info` (typ `0xff01`). Podczas ustanawiania nowego połączenia wartość renegotiation_info jest niezdefiniowana. Jeśli renegocjację rozpoczyna klient, przesyła za pomocą rozszerzenia „dane_weryfikacyjne_klienta". Jeśli renegocjacja jest inicjowana przez serwer, zmienna przechowuje wynik konkatenacji „danych_weryfikacyjnych_klienta" i „danych_weryfikacyjnych_serwera". Pole dane_weryfikacyjne_klienta jest definiowane w taki sam sposób jak podczas wysyłania komunikatu Finished kończącego proces uzgadniania połączenia. W protokole TLS jest to 12-bajtowa wartość (w protokole SSLv3

składa się ona z 36 bajtów). Analogicznie dane_weryfikacyjne_serwera są wyznaczane tak samo jak w czasie przygotowywania komunikatu Finished sygnalizującego zakończenie procedury uzgadniania połączenia po stronie serwera.

Niektóre z działających serwerów TLS (i SSL) przerywają połączenie po odebraniu nieznanego im rozszerzenia. Aby nie dopuścić do wystąpienia takiej sytuacji po przesłaniu (relatywnie nowego) rozszerzenia `renegotiation_info`, opracowano również rozwiązanie alternatywne. W czasie ustanawiania połączenia możliwe jest zdefiniowanie zestawu `TLS_EMPTY_RENEGOTIATION_INFO_SCSV`, które zastępuje rozszerzenie `renegotiation_info`. Operacja ma na celu użycie **wartości sygnalizacyjnego zestawu szyfrów** (SCSV — *Signaling Cipher Suit Value*), która nie opisuje żadnego rzeczywistego zestawu szyfrów, ale definiuje określony zbiór funkcji (podobna sztuczka jest wykorzystywana w protokole DNSSEC w rekordach NSEC3; więcej informacji na ten temat znajduje się w podpunkcie 18.10.1.3).

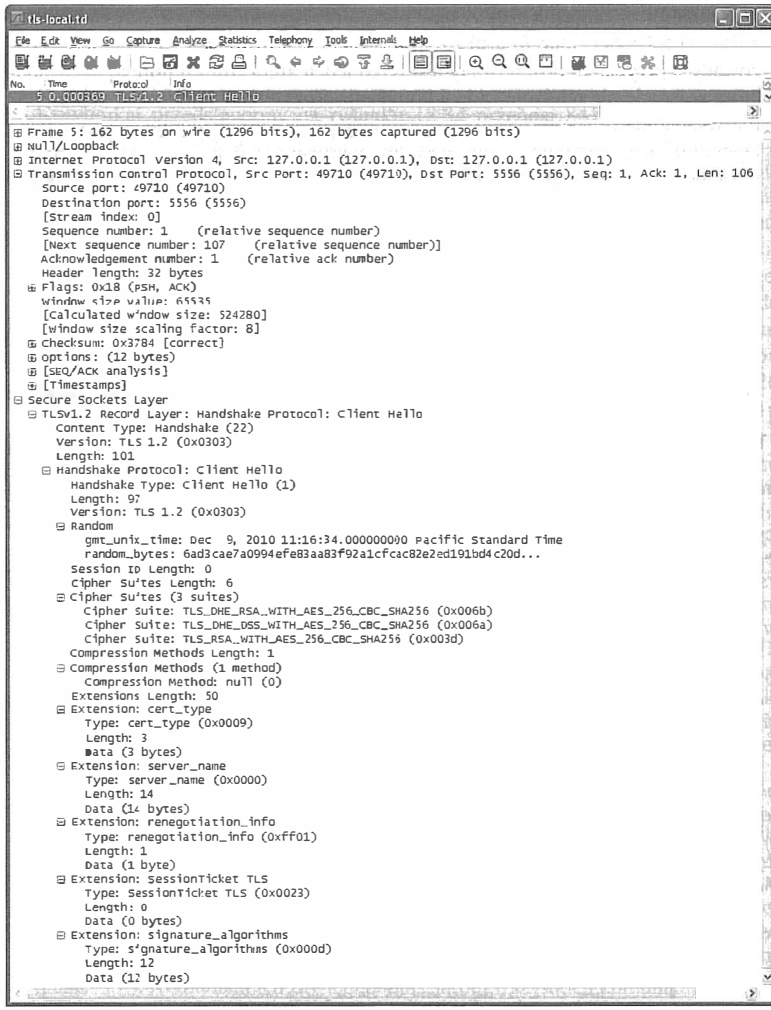
18.9.1.5. Przykład

Na rysunku 18.31 została przedstawiona przykładowa wymiana komunikatów w trakcie ustanawiania połączenia TLS 1.2 z wykorzystaniem protokołów TCP/IP na interfejsie pętli zwrotnej. Klient i serwer dysponują certyfikatami RSA i dostarczają je do siebie nawzajem. Etap uzgadniania połączenia TCP i aktualizacji okna, a także źródłowy i docelowy adres (127.0.0.1) nie zostały pokazane. Naniesione strzałki służą do łatwiejszego wyszukania odpowiednich informacji. Strzałka skierowana w prawo symbolizuje segment TCP, który zawiera przynajmniej jeden komunikat TLS wysłany przez klienta do serwera. Strzałka skierowana w lewo wskazuje komunikat przesyłany z serwera do klienta. Uzyskanie prezentowanego zestawienia było możliwe po wybraniu opcji *SSL* z menu *Analyze/Decode As...*



Rysunek 18.31. *Etap nawiązywania połączenia TLS 1.2 zarejestrowany przy użyciu programu Wireshark. Serwer pracuje na porcie 5556. Komunikaty klienckie dostarczane do serwera są oznaczone strzałką skierowaną w prawą stronę. Komunikaty przesyłane z serwera do klienta są oznaczone za pomocą strzałki skierowanej w lewą stronę. Segmenty TLS są przeplatane potwierdzeniami protokołu TCP. Po przekazaniu komunikatu ChangeCipherSpec (segment 21.) wszystkie kolejne wiadomości są zaszyfrowane i uwierzytelnione. W segmencie 13. widoczny jest komunikat ServerHelloDone*

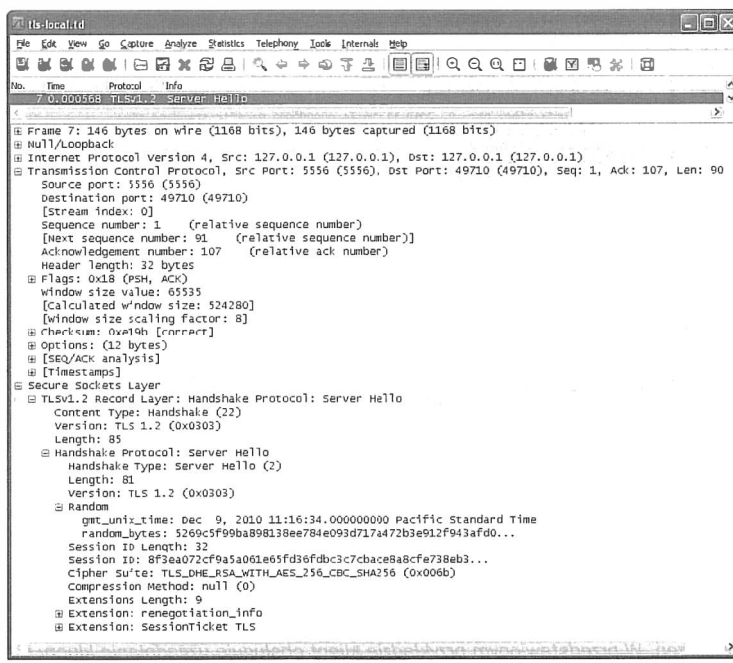
Zgodnie z rysunkiem 18.31 po zakończeniu etapu inicjowania połączenia TCP rozpoczyna się wymiana danych w protokole TLS (pierwszym komunikatem jest ClientHello). Segmenty TLS są przeplatane potwierdzeniami TCP. Po przekazaniu komunikatu ChangeCipherSpec rozpoczyna się szyfrowanie informacji. Aby zrozumieć zasadę działania mechanizmu, trzeba szczegółowo przeanalizować kilka pierwszych segmentów TLS. Na rysunku 18.32 przedstawiono zatem treść komunikatu ClientHello.



Rysunek 18.32. *Komunikat ClientHello protokołu TLS 1.2 zawierający informację na temat wersji mechanizmu, listę obsługiwanych zestawów szyfrów oraz algorytmów kompresji, dane losowe oraz kilka rozszerzeń. W przedstawionym przykładzie klient obsługuje uzgadnianie kluczy Diffiego-Hellmana, a także wymianę kluczy z użyciem algorytmu RSA. Do szyfrowania wykorzystuje algorytm AES-256 w trybie CBC, natomiast do ochrony integralności mechanizm SHA-256*

Przedstawiony na rysunku 18.32 komunikat ClientHello jest segmentem protokołu Record, który przenosi wiadomość ClientHello protokołu uzgadniania połączenia. Zawiera 32-bitowy znacznik czasu w formacie uniksowym (odpowiadający liczbie sekund, które upłynęły od północy 1 stycznia 1970 roku) oraz 28-bajtową wartość losową (*ClientHello.random*) służącą do generowania kluczy. Ponieważ operacja jest wykonywana na zupełnie nowym połączeniu, identyfikator sesji ma wartość 0. Lista zestawów szyfrów obsługiwanych przez stację kliencką składa się z trzech pozycji (zajmujących 6 bajtów) wymienionych w kolejności ważności. Każdy zestaw został przedstawiony za pomocą 16-bitowej wartości z listy TLS Cipher Suite Registry zdefiniowanej w dokumencie [TLSPARAMS]. W analizowanym przypadku obsługiwany jest tylko jeden algorytm kompresji — metoda NULL, która w istocie nie realizuje żadnej kompresji. Rozszerzenia protokołu zajmują 50 bajtów. Pierwsze z nich — `cert_type` — sygnalizuje, że klient obsługuje certyfikaty X.509 oraz OpenPGP. Rozszerzenie `server_name` ma wartość `127.0.0.1`, która odpowiada adresowi wpisanemu w aplikacji klienckiej. Pole `renegotiation_info` jest puste, podobnie jak rozszerzenie `SessionTicket TLS`, ponieważ analizowany jest pierwszy segment z procesu uzgadniania połączenia. Rozszerzenie `signature_algorithms` dostarcza następującą listę algorytmów podpisu obsługiwanych po stronie klienckiej: `sha1-rsa`, `sha1-dsa`, `sha256-rsa`, `sha384-rsa` oraz `sha512-rsa`.

W konfiguracji serwera zdefiniowany został tylko jeden zestaw szyfrów — `TLS_DHE_RSA_WITH_AES_256_CBC_SHA256` (`0x006b`). Serwer informuje o tym fakcie, odpowiadając na komunikat ClientHello komunikatem ServerHello (pokazanym na rysunku 18.33).



Rysunek 18.33. *Komunikat ServerHello protokołu TLS 1.2 zawierający informację na temat wersji mechanizmu, listę obsługiwanych zestawów szyfrów oraz algorytmów kompresji i kilka rozszerzeń. Wynika z niego, że serwer obsługuje uzgadnianie kluczy Diffiego-Hellmana. Do szyfrowania wykorzystuje algorytm AES-256, a do ochrony integralności — mechanizm SHA-256*

Na rysunku została zaprezentowana odpowiedź serwera (komunikat ServerHello) na żądanie klienckie (ClientHello). Serwer przekazuje za jej pomocą informację o jego czasie lokalnym oraz 28-bajtową wartość losową. Dołącza również 32-bajtowy identyfikator sesji. Z treści komunikatu wynika, że obsługuje tylko jeden zestaw szyfrów (uzgadnianie kluczy DH z wykorzystaniem certyfikatów RSA, szyfrowanie za pomocą algorytmu AES-256 w trybie CBC oraz mechanizm SHA-256 do ochrony integralności danych). Podobnie jak klient, nie pozwala na stosowanie kompresji. Przesyła również puste rozszerzenia `renegotiation_info` oraz `SessionTicket TLS`. Po wysłaniu pierwszego segmentu generowany jest komunikat Certifcate, którego treść została przedstawiona na rysunku 18.34.

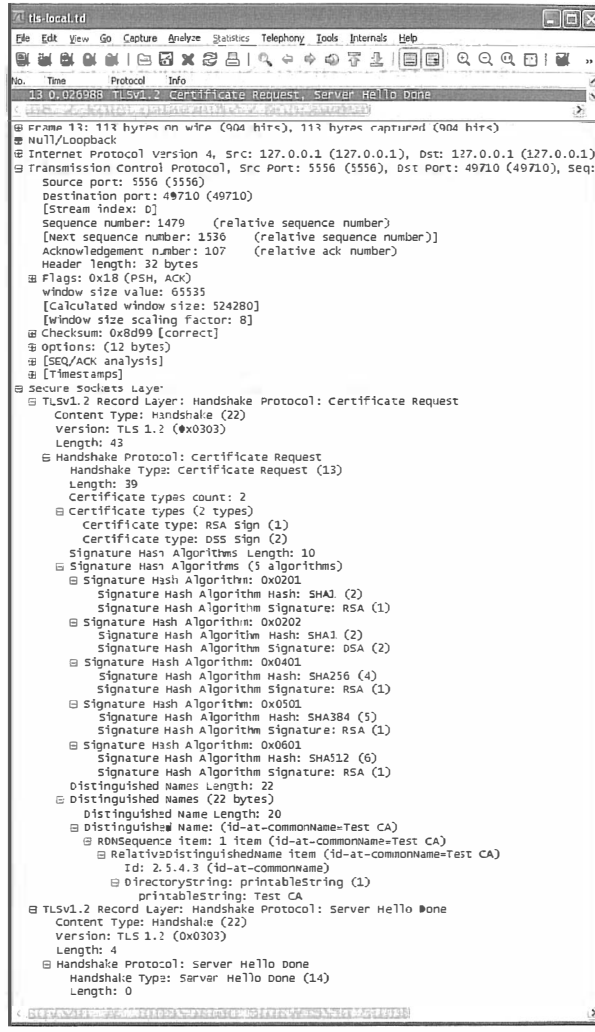
Komunikat przedstawiony na rysunku 18.34 przenosi 841-bajtowy certyfikat serwera zapisany w formacie X.509. W omawianym przypadku certyfikat został podpisany przez testowy urząd certyfikacji o nazwie *Test CA* (co można stwierdzić, analizując pole *Issuer*). Pole o nazwie *SubjectPublickeyInfo* zawiera 270-bajtowy klucz publiczny (wygenerowany za pomocą algorytmu RSA), za którego pomocą klient potwierdzi tożsamość serwera. Do odpowiedzi dołączonych zostało sześć rozszerzeń: `basicConstraints` (krytyczne), `subjectAltName` (nazwa domenowa serwera wykorzystującego certyfikat), `extKeyUsage` (rozszerzone zastosowanie klucza; dany klucz służy również do identyfikacji serwera), `keyUsage` (krytyczne; informuje, że dołączony klucz służy do szyfrowania klucza lub do generowania podpisów cyfrowych), `subjectKeyIdentifier` (20-bajtowy numer identyfikujący podpisany klucz publiczny) oraz `authorityKeyIdentifier` (20-bajtowy numer identyfikujący klucz wykorzystywany przez urząd certyfikacji do wystawienia danego certyfikatu).

Komunikat ClientKeyExchange nie został pokazany w szczegółach, ponieważ zazwyczaj przenosi dane binarne, potrzebne do przeprowadzenia wymiany DH. Kolejny segment wart zainteresowania to segment 13. (pojedynczy segment protokołu TCP) przenoszący zarówno komunikat CertificateRequest, jak i komunikat ServerHelloDone. Jego treść jest widoczna na rysunku 18.35.

Na rysunku 18.35 został pokazany segment TCP obejmujący dwa komunikaty TLS 1.2 — CertificateRequest oraz ServerHelloDone. Żądanie CertificateRequest jest dla klienta poleceniem dostarczenia certyfikatu i potwierdzenia swojej tożsamości w kolejnym żądaniu CertificateVerify. Certyfikat powinien zostać podpisany za pomocą algorytmu RSA lub DSS przez urząd certyfikacji Test CA. Dozwolone algorytmy podpisu to: `sha1-rsa`, `sha1-dsa`, `sha256-rsa`, `sha384-rsa` oraz `sha512-rsa`.

Informacje o łańcuchu certyfikacji klienta oraz stosowny klucz publiczny są dostarczane do serwera w pakiecie 15. (Certificate; niepokazanym szczegółowo na rysunku). W analizowanym przypadku pole nazwy stacji ma wartość *test client*, a pole *Issuer* wartość *Test CA*. Oznacza to, że certyfikaty klienta i serwera zostały podpisane przez ten sam urząd certyfikacji, a łańcuch certyfikacji składa się z jednego certyfikatu. Klient potwierdza posiadanie odpowiedniego klucza prywatnego, generując komunikat CertificateVerify (pakiet 19.). Komunikat ten zawiera podpis skrótu wszystkich komunikatów wymienianych w czasie uzgadniania sesji, do którego utworzenia został użyty prywatny klucz klienta. Operacja ta potwierdza nie tylko autentyczność klienta, ale również jego udział w dotychczasowej wymianie danych protokołu TLS (bez utraty lub zmiany kolejności segmentów). Po dostarczeniu komunikatu CertificateVerify wysyłany jest segment ChangeCipher sygnalizujący rozpoczęcie (szyfrowanej) komunikacji.

Rysunek 18.34. *Po wysłaniu segmentu ServerHello serwer generuje komunikat Certificate, który przenosi certyfikat serwera. Klient wykorzystuje dostarczony w ten sposób certyfikat do uwierzytelnienia serwera. Komunikat o analogicznym formacie służy do uwierzytelnienia klienta po stronie serwera*



Rysunek 18.35. *Komunikaty CertificateRequest i ServerHelloDone są przenoszone w jednym segmencie TCP. Klient wykorzystuje dostarczony certyfikat do uwierzytelnienia serwera. Wiadomość o analogicznym formacie służy również do uwierzytelnienia klienta po stronie serwera*

18.9.2. Protokół TLS do obsługi datagramów (DTLS)

Działanie protokołu TLS bazuje na założeniu, że do przekazywania komunikatów wykorzystywany jest strumieniowy protokół transportowy. Założenie to nie obowiązuje w przypadku użycia protokołu DTLS, który jest pewną odmianą protokołu TLS przystosowaną do współpracy z datagramami. Zapewnia uzyskanie takich samych rezultatów przy użyciu niemal identycznych formatów komunikatów. Pierwotnie znajdował on zastosowanie w systemach SIP, które działają na bazie protokołu UDP, ale nie używają stosu IPsec [RFC5406]. Później został zaadaptowany w rozwiązaniach DCCP [RFC5238]

i SCTP [RFC6083]. W czasie pisania książki obowiązywała wersja DTLS 1.0 [RFC4347] bazująca na mechanizmie TLS 1.1. Jednocześnie trwały prace nad wersją bazującą na protokole TLS 1.2 [IDDTLS]. W protokole DTLS wykorzystywane są takie same warstwy protokołów, jakie zostały przedstawione na rysunku 18.28, oraz większość komunikatów obowiązujących w rozwiązaniu TLS 1.2.

Największą trudnością w przygotowywaniu usług o działaniu zbliżonym do mechanizmu TLS, ale działających bez wsparcia niezawodnych protokołów transportowych, jest wyeliminowanie ryzyka utraty, zmiany kolejności oraz duplikowania datagramów. Wymienione problemy mogą mieć istotny wpływ na sposób działania funkcji szyfrowania oraz protokołu Handshake. W obu jest istotna kolejność komunikatów. Aby wyeliminować wspomniane niedogodności, w protokole DTLS wprowadzono jawne numerowanie wszystkich przenoszonych rekordów (w protokole TLS numerowanie ma charakter niejawny) oraz retransmisję uzależnioną od czasu. System wykorzystuje numery sekwencyjne niezależne od tych, które są stosowane w protokole Handshake (dodawane w warstwie rekordów).

18.9.2.1. Warstwa rekordów protokołu DTLS

Kolejność rekordów wysyłanych w protokole TLS jest niezwykle istotna, ponieważ wartość MAC wyliczana dla każdego z rekordów jest zależna od wartości obejmującej rekordy wcześniejsze. Precyzyjniej rzecz ujmując, obliczenie wartości MAC zależy od niejawnego 64-bitowego numeru sekwencyjnego zapisanego w każdym rekordzie, który staje się niepoprawny w przypadku zmiany kolejności lub utraty segmentów. Rozwiązaniem zastosowanym w protokole DTLS jest jawne zapisanie numeru sekwencyjnego podczas przetwarzania komunikatów w warstwie rekordów. Liczniki numerów sekwencyjnych są zerowane po każdorazowym odebraniu komunikatu ChangeCipherSpec. Z kolei do ich generowania wykorzystywany jest dodatkowy 16-bitowy **numer epoki** (*epoch number*), dołączany do nagłówka rekordu. Numer epoki jest zwiększany o jeden po każdorazowej zmianie stanu protokołu. To wyklucza sytuację, w której przetwarzane są komunikaty o tej samej wartości numeru sekwencyjnego, ale wygenerowane w ramach różnych procedur uzgodnienia połączenia (które mogą być realizowane jednocześnie).

Algorytm obliczania wartości MAC jest nieco zmodyfikowany w porównaniu z protokołem TLS, ponieważ obejmuje 64-bitowy ciąg powstały z połączenia dwóch nowych pól (numeru epoki oraz numeru sekwencyjnego). Dzięki temu każdy rekord może być przetwarzany niezależnie od innych. Trzeba pamiętać, że w mechanizmie TLS błędna wartość MAC powoduje przerwanie połączenia. W przypadku protokołu TLS zrywanie połączenia jest bezcelowe. Odbiorca może po prostu odrzucić pakiet z błędną wartością MAC lub wysłać komunikat alarmowy (który w takim przypadku musi zakończyć komunikację).

Zduplikowane datagramy są odrzucane lub opcjonalnie traktowane jako próba ataku z ponowieniem pakietu. Ewentualne wykrywanie ataków tego typu polega na utrzymywaniu okna bieżących numerów sekwencyjnych po stronie odbiorcy. Rozmiar okna musi umożliwiać buforowanie co najmniej 32 komunikatów, choć sugerowane jest zwiększenie jego pojemności do 64 komunikatów. Przetwarzanie datagramów jest w tym przypadku zbliżone do rozwiązania stosowanego w standardzie IPsec w odniesieniu do protokołów AH i ESP. Rekordy o numerach sekwencyjnych niższych niż dolna granica okna są odrzucane jako przeterminowane lub zduplikowane. Te, które mają numery z bieżącego prze-

działu, podlegają weryfikacji w celu ustalenia, czy nie zostały zduplikowane. Komunikat o numerze z bieżącego przedziału opatrzony poprawną wartością MAC jest zachowywany, nawet jeśli nie jest kolejnym odebranym segmentem. Rekordy z błędnymi wartościami MAC są odrzucane. Z kolei datagramy z poprawnymi wartościami MAC o numerach wyższych niż górna krawędź okna powodują przesunięcie okna. Górna krawędź okna wyznacza więc jednocześnie rekord o najwyższym numerze sekwencyjnym spośród wszystkich poprawnie odebranych rekordów.

Pojedynczy datagram może zawierać kilka rekordów DTLS, ale pojedynczy rekord nie może być transmitowany w kilku datagramach. Warstwa rekordów umożliwia aplikacji korzystanie z procesu PMTUD zbliżonego do stosowanego w mechanizmie TCP (patrz rozdział 15.), co pozwala na unikanie wysyłania datagramów, które prawdopodobnie zostałyby podzielone na fragmenty. Próba wysłania datagramu o rozmiarze przekraczającym wartość PMTU lub maksymalny rozmiar datagramu (wartość PMTU pomniejszona o narzut DTLS) jest sygnalizowana aplikacji jako błąd. Wyjątkiem od tej reguły jest sposób obsługi protokołu Handshake, którego komunikaty mają relatywnie duży rozmiar.

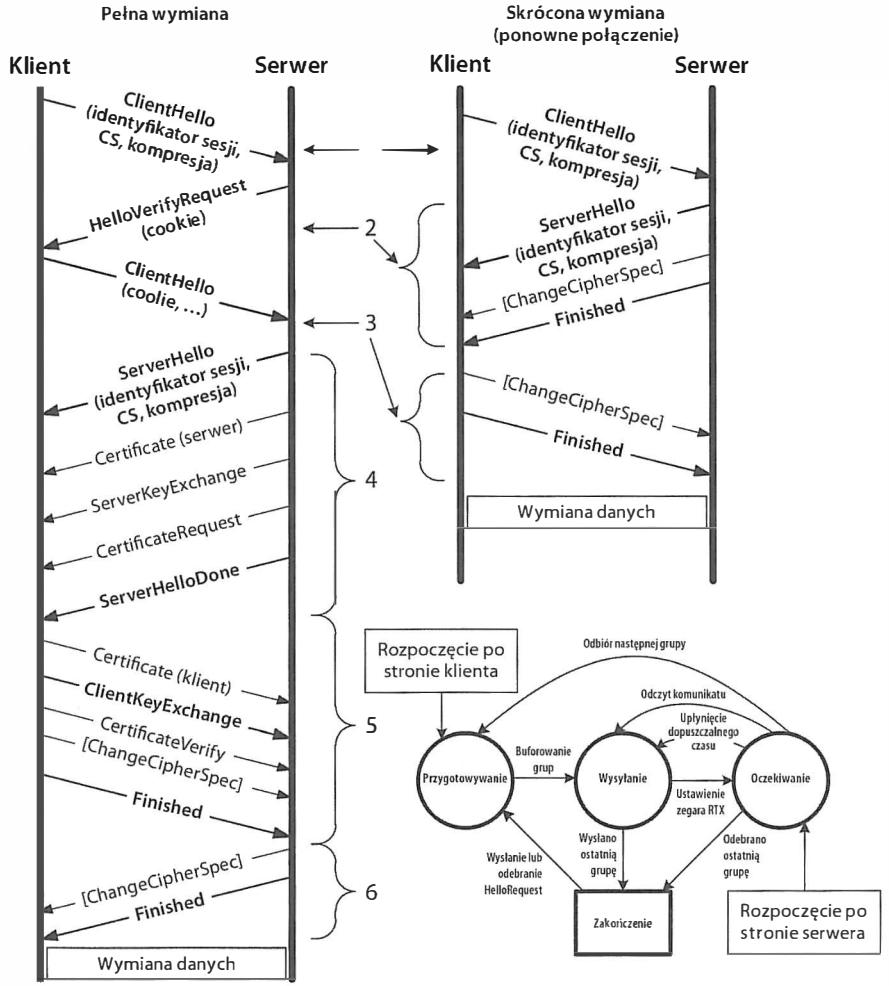
18.9.2.2. Protokół Handshake w protokole DTLS

Komunikaty protokołu Handshake mogą mieć rozmiar $2^{24} - 1$ bajtów, choć w praktyce nie przekraczają kilku kilobajtów. Jednak i tak przekraczają maksymalną pojemność datagramu UDP, wyznaczoną na 1,5 kB. Aby wymiana informacji Handshake została przeprowadzona poprawnie, komunikat musi zostać podzielony na kilka rekordów DTLS za pomocą mechanizmu fragmentacji. Każdy z fragmentów jest wówczas przenoszony w osobnym rekordzie, zapisanym w datagramie niższej warstwy. Implementacja fragmentacji wymusiła na projektantach wprowadzenie do komunikatu Handshake trzech pól: 16-bitowego numeru sekwencyjnego (*Sequence Number*), 24-bitowego pola przesunięcia fragmentu (*Fragment Offset*) oraz 24-bitowego pola długości fragmentu (*Fragment Length*).

Sama operacja fragmentacji polega na podzieleniu treści komunikatu na kilka następujących po sobie bloków danych. Każdy blok musi mieć rozmiar mniejszy niż maksymalna wielkość fragmentu, ponieważ jest zapisywany jako osobny komunikat. Każdy fragment otrzymuje taki sam numer sekwencyjny jak wiadomość oryginalna. Pola przesunięcia fragmentu i długości fragmentu są wyrażone w bajtach i umożliwiają nadawcy zadbanie o to, by poszczególne bloki danych nie nakładały się na siebie. Nie zwalnia to jednak odbiorcy z obowiązku obsłużenia również sytuacji, w której takie nałożenie występuje, bo istnieje możliwość zmiany rozmiaru rekordu w czasie trwania komunikacji i wykonania retransmisji.

Rozwiązanie problemu utraty komunikatów w protokole DTLS sprowadza się do wykorzystania zegara retransmisji uruchamianego w odniesieniu do *grup* komunikatów będących w danym czasie w trakcie transmisji.

Na rysunku 18.36 zostały przedstawione początkowe wymiany danych — pełna (po lewej stronie) i skrócona (po prawej stronie) — wraz diagramem stanów protokołu Handshake w mechanizmie DTLS.



Rysunek 18.36. *Protokół DTLS musi obsługiwać przypadki utraty datagramów. Pełna początkowa wymiana danych (po lewej) polega na wysłaniu sześciu grup, z których każda może być retransmitowana. Skrócona procedura uzgadniania (po prawej) wymaga przesłania tylko trzech datagramów, co różni ją nieznacznie od mechanizmu TLS. Algorytm DTLS jest algorytmem stanów skończonych o trzech stanach (pokazanych z prawej dolnej części rysunku)*

Numery grup komunikatów zostały zamieszczone pomiędzy kolumnami wymiany pełnej i skróconej. Pełna wymiana jest bardzo zbliżona do wymiany TLS przedstawionej na rysunku 18.30. Różnica polega jedynie na emisji dodatkowego komunikatu HelloVerifyRequest oraz drugiego komunikatu ClientHello (zawierającego za drugim razem dane cookie). Skrócona wersja wymiany przebiega zupełnie inaczej. W protokole DTLS pierwszy komunikat Finished jest przesyłany przez serwer, podczas gdy w rozwiązaniu TLS klient przesyła komunikat Finished jako pierwszy.

Prawa dolna część rysunku 18.36 przedstawia diagram stanów implementowany w modułach DTLS do obsługi protokołu Handshake. Wyróżniono na nim trzy podstawowe stany: przygotowywanie, wysyłanie oraz oczekiwanie. Działania jednostki klienckiej rozpoczynają się od stanu przygotowywania, w którym tworzony jest komunikat ClientHello. Praca serwera rozpoczyna się od stanu oczekiwania, bez zbuforowanych komunikatów oraz z nieaktywnym zegarem. Podczas wysyłania datagramu zegar retransmisji jest ustawiany, a proces obsługi protokołu przechodzi do stanu oczekiwania. Pozostaje w nim do czasu zakończenia transmisji. Jeśli wcześniej upłynie czas odliczany za pomocą zegara retransmisji (RTX), protokół przechodzi do stanu wysyłania, w którym realizowana jest retransmisja. Ten sam skutek daje odebranie retransmitowanej grupy. System lokalny wykonuje retransmisję własnych datagramów, zakładając, że odebranie retransmitowanych komunikatów jednostki zdalnej jest sygnałem o całkowitej lub częściowej utracie danych z poprzedniej emisji. Jeśli wysyłanie zakończy się powodzeniem, proces kończy działanie lub przechodzi do stanu przygotowania w celu utworzenia następnej grupy komunikatów.

Działanie mechanizmu jest zależne od zegara retransmisji, którego zalecaną domyślną wartością jest 1 s. Jeśli w tym czasie nadawca nie zarejestruje odpowiedzi ze strony drugiego urządzenia, emisja grupy komunikatów jest ponawiana (z takim samym numerem sekwencyjnym protokołu Handshake, ale z kolejnymi numerami sekwencyjnymi warstwy rekordów). Kolejne retransmisje bez odpowiedzi powodują podwojenie wartości RTX, aż do maksymalnego poziomu 60 s. Wyzerowanie tej wartości jest możliwe po poprawnym odebraniu odpowiedzi lub po dłuższej przerwie (dziesięciokrotnie dłuższej od bieżącej wartości zegara).

18.9.2.3. Zabezpieczenie przed atakami DoS w protokole DTLS

Gdy zamiast protokołu strumieniowego stosuje się datagramy, trzeba się liczyć z pewnymi dodatkowymi zagrożeniami. Szczególnie groźne wydają się dwie formy ataków DoS. W obu wykorzystuje się relatywnie łatwą do przeprowadzenia operację podmiany źródłowego adresu IP w komunikacie ClientHello. Wysłanie dużej liczby pakietów tego typu może doprowadzić do ataku DoS na serwer DTLS, co prowadzi do wyczerpania zasobów jednostki podczas przygotowywania odpowiedzi. W pewnej odmianie opisanego ataku używanych jest więcej komputerów atakujących, które ustawiają ten sam sfałszowany adres IP (ofiary). Serwer wysyła wówczas wiele odpowiedzi do jednostki o podanym adresie IP, wykonując atak DoS.

Rozwiązaniem problemu ataków DoS jest wprowadzenie procedury bezstanowej weryfikacji danych cookie w ramach wymiany Hello. Odebranie przez serwer komunikatu ClientHello powoduje wygenerowanie nowego komunikatu HelloVerifyRequest zawierającego 32-bitową wartość cookie (odpowiadającą pewnej tajnej wartości, adresowi IP stacji klienckiej lub parametrom połączenia). Kolejny komunikat ClientHello musi zawierać określoną wartość cookie. W przeciwnym przypadku serwer zignoruje ten komunikat. Niestety, nie rozwiązuje to problemu skoordynowanego ataku z wielu jednostek o poprawnych adresach IP, które mogą zakończyć wymianę cookie zgodnie z założeniami protokołu.

18.10. Bezpieczeństwo protokołu DNS (DNSSEC)

Po zapoznaniu się z mechanizmami zabezpieczeń implementowanymi na poziomie warstwy łącza danych, sieciowej oraz transportowej przeanalizujmy rozwiązania funkcjonujące w warstwie aplikacji. Jako pierwsze omówione zostaną zabezpieczenia systemu nazw domenowych (DNS — *Domain Name System*), mimo że w czasie pisania książki nie były one jeszcze powszechnie wdrażane. Ochrona systemu DNS obejmuje zarówno zabezpieczenie danych zapisanych w serwerze DNS (rekordy zasobów [RR — *Resource Record*]), jak i transakcji, które synchronizują (aktualizują) przechowywane dane. Z uwagi na wielkie znaczenie usługi DNS w funkcjonowaniu Internetu wdrożenie mechanizmów ochronnych stało się jednym z najważniejszych zadań osób zajmujących się bezpieczeństwem sieci. Efektem ich prac jest rozwiązanie określane jako **rozszerzenia zabezpieczające system nazw domenowych** (DNSSEC — *Domain Name System Security Extensions*), które zostało opisane w kilku dokumentach RFC [RFC4033] [RFC4034] oraz [RFC4035]. Są one niekiedy nazywane specyfikacją **DNSSECbis**, ponieważ zastępują wcześniejsze opracowanie DNSSEC. Przed szczegółową analizą rozwiązań DNSSEC warto poświęcić chwilę na zapoznanie się z podstawami działania systemu DNS (omówionymi w rozdziale 11.).

Wspomniane rozszerzenia zapewniają autentyczność źródła danych oraz integralność danych wymienianych w ramach systemu DNS. Umożliwiają również (w ograniczonym zakresie) dystrybuowanie kluczy zabezpieczających transmisję. Celem ich stosowania jest zagwarantowanie na podstawie bezpiecznych algorytmów kryptograficznych, że określony blok informacji DNS został utworzony przez wskazaną jednostkę oraz że informacje te nie zostały zmienione podczas transmisji. W standardzie DNSSEC występuje również termin **uwierzytelnionego nieistnienia** (*authenticated nonexistence*). Odpowiedzi DNS informujące o nieistnieniu określonej nazwy domenowej są zabezpieczane w taki sam sposób jak odpowiedzi uwzględniające nazwy domenowe. Rozwiązanie DNSSEC nie zapewnia poufności danych, ochrony przed atakami DoS oraz kontroli dostępu do usługi. Zabezpieczenie transakcji (wykorzystywane w rozwiązaniach DNSSEC) jest definiowane oddzielnie, dlatego zostanie krótko opisane po przedstawieniu zasadniczych funkcji zabezpieczeń DNSSEC.

W działaniu mechanizmu DNSSEC wykorzystywane są moduły odwzorowania nazw (*resolver*) o różnych poziomach integracji z systemem zabezpieczeń. Moduł nazywany **resolverem walidującym** (*validating resolver*) sprawdza podpisy kryptograficzne otrzymywanych danych, aby ustalić, czy przetwarzane informacje DNS są prawdziwe. Moduły resolverów implementowane w jednostkach końcowych oraz komponenty odpowiedzialne za rekurencyjne działanie serwerów nazw mogą zawierać mechanizmy zabezpieczające, ale nie wykonują walidacji kryptograficznej. Są natomiast zobowiązane do ustanowienia bezpiecznych relacji z resolverami walidującymi. Dalsze omówienie zostało poświęcone przede wszystkim resolverom walidującym, ponieważ są najbardziej wyrafinowanymi i interesującymi komponentami systemu. W ramach realizowanych zadań ustalają, czy przetwarzane informacje DNS są **bezpieczne** (poprawne i zabezpieczone stosownymi podpisami), **niebezpieczne** (z poprawnego podpisu wynika, że w danych występują wartości, których nie powinno być), **sfałszowane** (dane wydają się poprawne, ale nie można ich uwierzytelnić) oraz **nieokreślone** (nie można określić prawdziwości informacji, najczęściej z powodu braku podpisów). W przypadku braku informacji dodatkowych zazwyczaj dane są uznawane za nieokreślone.

Bezpieczne działanie mechanizmu DNSSEC zależy od tego, czy administrator domeny podpisze plik strefy, czy istnieje podstawa zaufania oraz czy zarówno oprogramowanie serwerowe, jak i klienckie korzystają z protokołu. Zagwarantowanie bezpieczeństwa danych DNS wymaga od resolverów walidujących sprawdzenia sygnatur, które z kolei muszą być powiązane z co najmniej jedną kotwicą zaufania (podobnie jak certyfikaty głównych urzędów CA w środowisku PKI). Trzeba jednak pamiętać, że system DNSSEC nie jest środowiskiem PKI — operacje podpisywania i unieważniania kluczy są realizowane jedynie w ograniczonym zakresie i nie dysponują żadnymi rozwiązaniami analogicznymi do listy unieważnionych certyfikatów [RFC5011].

Wykonując zapytanie DNS w protokole DNSSEC, resolver wykorzystuje rozszerzenie EDNS0 i ustawia bit DO (DNSSEC OK) w metarekordzie zasobu OPT dołączonym do żądania. Bit ten wskazuje, że klient jest zainteresowany pozyskaniem informacji DNSSEC oraz obsługuje rozszerzenie EDNS0. Bit DO jest pierwszym (najbardziej znaczącym) bitem w drugim 16-bitowym polu „rozszerzonego kodu RCODE i znaczników" w metarekordzie zasobu EDNS0 (patrz sekcja 3. dokumentu [RFC3225] oraz sekcja 4. dokumentu [RFC2671]). Serwery, które odbierają żądania z nieustawionym bitem DO (lub nieobecnym), nie mają prawa zwracania większości rekordów zasobów opisanych w punkcie 18.10.1, chyba że dostarczenie określonego rekordu zostało wprost określone w żądaniu. Takie rozwiązanie pozwala na zwiększenie wydajności pracy serwerów, ponieważ zapobiega wysyłaniu danych RR związanych z zabezpieczeniami, które i tak nie są przetwarzane w resolverach pozbawionych mechanizmów zabezpieczających. Poza tym serwery DNS standardowo wysyłają relatywnie niewielkie pakiety UDP, a w razie konieczności przekazania dużej porcji danych wykorzystują protokół TCP. Protokół TCP z kolei wnosi istotne opóźnienie wynikające z obowiązku przeprowadzenia procedury trzyetapowego uzgodnienia połączenia.

Podczas przetwarzania żądania dostarczonego z resolvera zgodnego ze standardem DNSSEC serwer analizuje bit CD (sprawdzanie wyłączone [*checking disabled*]) (więcej informacji na ten temat znajduje się w rozdziale 11.). Wartość 1 oznacza, że klient żąda dostarczenia danych bez ich potwierdzania. Standardowy proces przygotowywania odpowiedzi uwzględnia operację kryptograficznego potwierdzenia wysyłanych informacji. Poprawne wykonanie zadania jest sygnalizowane ustawieniem bitu AD (dane autentyczne — *authentic data*) w polu odpowiedzi [RFC4035]. Resolvery, które uwzględniają zabezpieczenia, ale nie weryfikują danych we własnym zakresie, polegają na oznaczonych w ten sposób informacjach, jeśli mają gwarancję bezpiecznej komunikacji z samym serwerem. Najbezpieczniejszym rozwiązaniem jest bez wątpienia uruchomienie walidującego resolvera, który sam wykona weryfikację kryptograficzną. Wówczas żądania mogą być generowane z ustawionym bitem CD. Mechanizm gwarantuje bezpieczeństwo komunikacji między jednostkami końcowymi systemu DNS (bez konieczności ufania resolverom pośredniczącym). Dodatkowo zmniejsza obciążenie obliczeniowe serwerów pośrednich, bo zwalnia je z obowiązku wykonywania walidacji kryptograficznej.

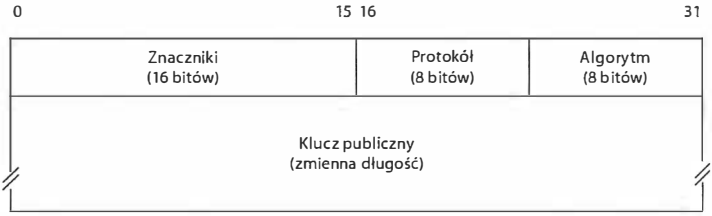
18.10.1. Rekordy zasobów DNSSEC

Zgodnie ze specyfikacją [RFC4034] protokół DNSSEC wykorzystuje cztery nowe rekordy zasobów (RR) oraz dwa bity nagłówka komunikatu (CD i AD). Wymaga również obsługi rozszerzenia EDNS0 i wspomnianego wcześniej bitu DO. Dwa z czterech nowych rekordów RR służą do przenoszenia podpisów odpowiednich obszarów przestrzeni nazw

DNS. Dwa pozostałe odpowiadają za dystrybuowanie i potwierdzanie kluczy. Zmiana opisana w dokumencie [RFC5155] spowodowała wprowadzenie dwóch kolejnych rekordów, które powinny zastąpić jeden z pierwotnie zdefiniowanej czwórki.

18.10.1.1. Rekordy zasobów odpowiadające za bezpieczeństwo DNS (DNSKEY)

Omówienie zagadnienia rozpoczniemy od analizy sposobu przechowywania i dystrybuowania kluczy w systemie DNSSEC. Klucze publiczne są przechowywane w rekordzie DNSKEY. Można ich używać jedynie w ramach komunikacji DNSSEC. Pozostałe rekordy (np. CERT RR [RFC4398]) przydają się do składowania kluczy i certyfikatów przeznaczonych również do innych celów. Format pola *RDATA* rekordu DNSKEY został pokazany na rysunku 18.37.



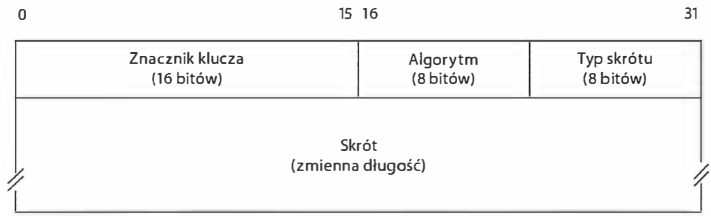
Rysunek 18.37. *Pole RDATA rekordu DNSKEY przechowuje klucz publiczny przeznaczony jedynie dla mechanizmu DNSSEC. Pole Znaczniki obejmuje znaczniki Klucza strefy (bit 7.), Bezpiecznego punktu wejścia (bit 15.) oraz Unieważnienia (bit 8.). Bit klucza strefy jest w zasadzie ustawiony podczas przetwarzania wszystkich kluczy DNSSEC. Jeśli dodatkowo ustawiony zostanie bit SEP, dany klucz jest nazywany kluczem podpisującym klucz i jest wykorzystywany do weryfikowania delegacji do stref podrzędnych. Jeśli bit nie jest ustawiony, dany klucz służy jedynie do podpisania strefy, ma krótszy okres ważności i odnosi się tylko do zawartości strefy, a nie do delegacji. Podany klucz jest przeznaczony do użycia z algorytmem zdefiniowanym w polu Algorytm*

W przedstawionym na rysunku 18.37 polu *Znaczniki* (*flags*) wykorzystywane są jedynie trzy bity. Bit 7. jest bitem *Klucza strefy* (*zone key*). Jeśli jest ustawiony, nazwa właściciela rekordu DNSKEY musi odpowiadać nazwie strefy, a dołączony do rekordu klucz jest nazywany *Kluczem podpisującym strefę* (ZSK — *Zone Signing Key*) lub *Kluczem podpisującym klucz* (KSK — *Key Signing Key*). Wartość 0 oznacza, że rekord przechowuje klucz DNS innego rodzaju, którego nie można użyć do weryfikacji podpisów strefy. Bit 15. jest nazywany bitem *Bezpiecznego punktu wejścia* (SEP — *Secure Entry Point*) i stanowi rodzaj podpowiedzi wykorzystywanej przez oprogramowanie przeznaczone do debugowania lub podpisywania danych w celu ustalenia, jakie jest przeznaczenie danego klucza. Procedura weryfikacji podpisu nie uwzględnia interpretacji bitu SEP. Jednak klucze z ustawionym bitem SEP są zazwyczaj kluczami KSK przeznaczonymi do zabezpieczenia hierarchii DNS przez potwierdzanie kluczy stref podrzędnych (za pośrednictwem rekordów DNS; patrz podpunkt 18.10.1.2). Bit 8. jest bitem *Unieważnienia* (*revoke*) [RFC5011]. Jego ustawienie oznacza, że danego klucza nie wolno użyć do walidacji. W bieżącej wersji mechanizmu DNSSEC parametr *Protokół* (*protocol*) ma stałą wartość 3. W polu *Algorytm* (*algorithm*) podawany jest rodzaj algorytmu podpisu [DNSSECALG]. Zgodnie z dokumentem [RFC4034] dla rekordu DNSKEY zdefiniowane zostały jedynie algorytmy DSA i RSA z funkcją SHA-1 (o wartościach odpowiednio 3 i 5),

choć w innych specyfikacjach opisano także inne algorytmy (np. ECC-GOST o wartości 12 [RFC5933] lub SHA-256 o wartości 8 [RFC5702]). Wartość pola jest wykorzystywana również w kilku innych rekordach zasobów DNSSEC. Pole *Klucz publiczny* (*public key*) przechowuje klucz publiczny o formacie zależnym od zawartości pola *Algorytm*.

18.10.1.2. Rekordy podpisującego delegację (DS)

Rekord **podpisującego delegację** (DS — *Delegation Signer*) jest używany do wskazania rekordu DNSKEY (najczęściej z poziomu strefy nadrzędnej do strefy podrzędnej). Rekordy DS są potrzebne w procesie uwierzytelnienia do weryfikacji klucza publicznego (więcej informacji na ten temat znajduje się w punkcie 18.10.2). Format rekordu DS został pokazany na rysunku 18.38.



Rysunek 18.38. *Pole RDATA rekordu DS zawierającego odwołanie do rekordu DNSKEY (w polu Znacznik klucza). Obejmuje dodatkowo skrót rekordu DNSKEY oraz nazwy właściciela, a także informacje na temat typu skrótu oraz algorytmu*

Widoczne na rysunku 18.38 pole *Klucz* (*key tag*) stanowi odsyłacz do rekordu DNSKEY. Nie ma jednak unikatowego charakteru. Wiele rekordów DNSKEY może być wskazywanych za pomocą tej samej wartości pola. Należy więc je postrzegać jedynie jako podpowiedź przy wyszukiwaniu (co nie zwalnia z obowiązku przeprowadzenia walidacji). Wartość pola jest wyliczana jako 16-bitowa suma wartości składających się na wskazywane pole *RDATA* rekordu DNSKEY (widoczne na rysunku 18.37; wynik jest liczbą bez znaku; przeniesienia są ignorowane). Pole *Algorytm* przechowuje takie same wartości, jakie są stosowane w polu algorytmu w rekordzie DNSKEY. Rodzaj użytej funkcji skrótu jest odzwierciedlany w polu *Typ skrótu* (*digest type*). Zgodnie ze specyfikacją [RFC4034] dopuszczalne jest stosowanie tylko wartości 1 odpowiadającej algorytmowi SHA-1. W dokumencie [RFC4509] zaproponowano jednak również funkcję SHA-256 (o wartości 2). Aktualny wykaz obsługiwanych mechanizmów znajduje się rejestrze DS RR Type Digest [DSRRTYPES]. Pole *Skrót* (*digest*) zawiera skrót wskazywanej wartości DNSKEY liczony w następujący sposób:

skrót = algorytm_skrótu(nazwa właściciela DNSKEY | *RDATA* rekordu DNSKEY)

Operator | odpowiada za konkatenację, natomiast dane *RDATA* rekordu DNSKEY są wyznaczane w następujący sposób:

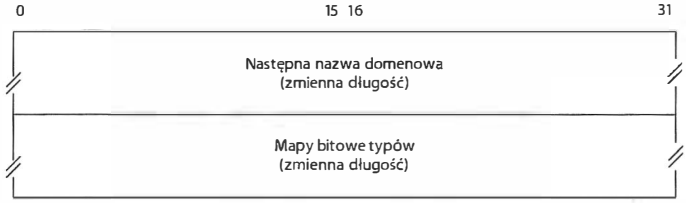
RDATA rekordu DNSKEY = *Znaczniki* | *Protokół* | *Algorytm* | *Klucz publiczny*

Funkcja SHA-1 generuje wynik o długości 20 bajtów, natomiast SHA-256 zwraca 32 bajty. Rekord DS stanowi odnośnik do kolejnego rekordu w łańcuchu uwierzytelniania, przekraczający granicę strefy. Wskazywany rekord DNSKEY musi więc opisywać klucz strefy (tzn. 7. bit pola *Znaczniki* w rekordzie DNSKEY musi mieć wartość 1).

W czasie pisania książki trwały prace nad pewnymi wariantami rekordu DS (określanymi jako DS2) [IDDS2]. Zgodnie z założeniami mają one wprowadzić do rekordu DS nazwę kanoniczną podpisującego, dzięki czemu kilka stref o identycznej zawartości może zostać rozróżnionych i potwierdzonych przez kilku (różnych) podpisujących. Ponadto wprowadzono rekord DLV [RFC4431] reprezentujący delegacje w przypadkach, w których strefa nadrzędna nie jest podpisana lub nie opublikowano rekordu DS. Format rekordu DLV jest identyczny z formatem rekordu DS. Różnica tkwi jedynie w sposobie interpretowania zawartości.

18.10.1.3. Rekordy NextSECure (NSEC oraz NSEC3)

Znając rekordy zasobów odpowiedzialne za przechowywanie kluczy oraz bezpieczne ich przekazywanie, możemy zająć się analizą rekordów przeznaczonych do weryfikowania struktury strefy oraz zawartych w niej wpisów. Pierwszym z nich jest rekord NextSECure (NSEC) wykorzystywany do definiowania następnej (*next*) nazwy domenowej właściciela grupy rekordów (RRset) na liście nazw (patrz podpunkt 18.10.2.1) lub określenia punktu delegacji (typu *NS*) w grupie RRset (grupę RRset należy w tym przypadku rozumieć jako zbiór rekordów należących do tego samego właściciela, do tej samej klasy, o takiej samej wartości TTL, ale różnych danych). Rekord ten służy również do przechowywania listy typów rekordów powiązanych z nazwą właściciela rekordu NSEC. Pozwala to na uwierzytelnienie i sprawdzenie integralności struktury strefy. Format rekordu NSEC został przedstawiony na rysunku 18.39.



Rysunek 18.39. *Pole RDATA rekordu NSEC zawierające następną nazwę właściciela grupy RRset w strefie (określaną w porządku kanonicznym). Obejmuje również informację o typach rekordów powiązanych z nazwą właściciela rekordu NSEC*

Rekord NSEC jest wykorzystywany do formowania łańcucha nazw odpowiadającego grupie RRset w strefie. Oznacza to, że grupy niewymienione w łańcuchu można uznać za nieistniejące. Dzięki temu możliwe jest generowanie wiarygodnych powiadomień o nieistnieniu, wspomnianych w początkowej części podrozdziału. Pole *Następna nazwa domenowa* (*next domain name*) przechowuje następny wpis z kanonicznie posortowanego łańcucha nazw domenowych strefy, w którym nie są stosowane techniki kompresji nazw domenowych opisane w rozdziale 11. Wartość pola w ostatnim rekordzie NSEC w łańcuchu odpowiada początkowi strefy (nazwie właściciela rekordu SOA strefy).

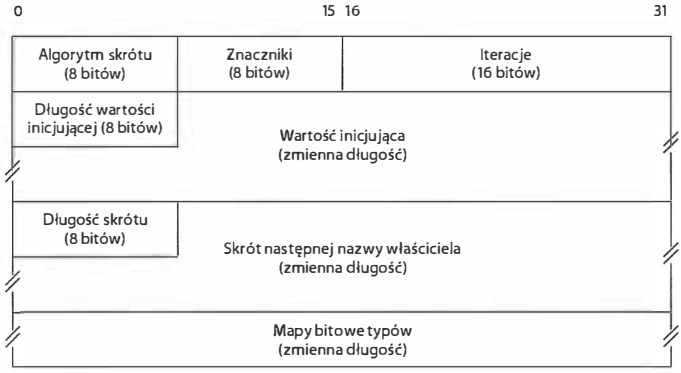
Pole *Mapy bitowe typów* (*type bit map*) zawiera mapę bitową typów rekordów powiązanych z nazwą domenową właściciela rekordu NSEC. Możliwe jest określenie 64 tysięcy typów, z których ok. 100 zostało zdefiniowanych w dokumencie [DNSPARAMS]. Jednak tylko niewielka ich część jest wykorzystywana na szerszą skalę. Przykładowo główna strefa Internetu (symbolizowana jako „.”), zgodna z mechanizmem DNSSEC od 15 lipca 2010 roku, zawiera pole *Następna nazwa domenowa* o wartości ac (jest to domena krajowa) oraz mapę bitową wskazującą na występowanie rekordów o następujących typach: NS, SOA, RRSIG, NSEC i DNSKEY.

Aby zakodować informację o dostępności rekordów określonego typu, cała przestrzeń typów RR jest dzielona na 256 bloków, ponumerowanych od 0 do 255. Każdy blok może odzwierciedlać występowanie maksymalnie 256 typów rekordów, do których kodowania używa się odpowiedniej maski bitowej. Bit znajdujący się na pozycji *P* w bloku *N* opisuje typ o numerze wyznaczonym przez wzór (*N**256 + *P*). Przykładowo bit zapisany w bloku 1. na pozycji 2. odpowiada typowi rekordu o identyfikatorze 258 (obecnie niezdefiniowanym). Kodowanie wartości pola sprowadza się do zastosowania poniższej zależności:

$$\text{Mapy bitowe typów} = (\text{numer bloku} \mid \text{długość mapy bitowej} \mid \text{mapa bitowa})^*$$

Znak | symbolizuje operację konkatenacji, a symbol * reprezentuje domknięcie Kleene'ego. Liczba reprezentująca jeden blok zawiera się w przedziale od 0 do 255, a długość mapy bitowej odzwierciedla długość pojedynczej mapy wyrażonej w bajtach (maksymalną wartością jest więc 32). Numer bloku i długość mapy bitowej są wartościami jednobajtowymi, a sama mapa bitowa może zawierać maksymalnie 32 bajty (256 bitów, po jednym na każdy dopuszczalny typ RR w bloku). Bloki, w który nie występuje żaden typ RR, nie są uwzględniane. Kodowanie jest więc efektywne nawet w przypadku nielicznych typów (rozmieszczonych w różnych blokach). Gdyby np. dostępne były jedynie rekordy o typach 1 (A) oraz 15 (MX), wynik kodowania byłby następujący: 0x00024001 = (0x00 | 0x02 | 0x4001).

Podstawowa struktura rekordów NSEC (opisana w dokumencie [RFC4034]) umożliwia dowolnej osobie wyliczenie rekordów w strefie przez „przejście" wzdłuż łańcucha NSEC, nazwane **enumeracją strefy** (*zone enumeration*). Działanie to może doprowadzić do „wycieku" informacji na temat konfiguracji danej sieci. Z tego powodu rekord NSEC został zastąpiony dwoma innymi rekordami opisanymi w zaleceniu [RFC5155]. Pierwszy z nich — NSEC3 — wykorzystuje skróty kryptograficzne nazw domenowych właściciela zamiast samych niezakodowanych nazw. Format rekordu został przedstawiony na rysunku 18.40.



Rysunek 18.40. *Pole RDATA rekordu NSEC3 zawiera skrót następnej nazwy właściciela grupy RRset w strefie (określaną w porządku kanonicznym). Pole Iteracje informuje o liczbie wywołań funkcji skrótu. Wartość inicjująca jest dodawana do nazwy przed wykonaniem funkcji skrótu. Zwiększa ona odporność systemu na ataki słownikowe. Pole Mapy bitowe typów mają taką samą strukturę jak w rekordach NSEC. Analogiczny format mają rekordy NSEC3PARAM. Różnią się jedynie tym, że zawierają parametry skrótu (a nie wartości Skrót następnego właściciela i Mapy bitowe typów)*

Występujące w rekordzie NSEC3 pole *Algorytm skrótu* (*hash algorithm*) wyznacza funkcję skrótu, która została wykonana w odniesieniu do następnej nazwy właściciela, generując wartość pola *Skrót następnej nazwy właściciela* (*next hash owner*). Obecnie do tego celu wykorzystuje się jedynie algorytm SHA-1 (o identyfikatorze 1) [NSEC3PARAMS]. Najmniej znaczący bit pola *Znaczniki* opisuje opcję *opt-out*, która (w przypadku włączenia) informuje odbiorcę, że rekord NSEC3 może obejmować niepodpisane delegacje. Jest to przydatne w sytuacjach, w których delegacja (grupa RRset typu NS) odnosi się do niepodpisanej strefy podrzędnej. Pole *Iteracje* informuje, ile razy wykonano funkcję skrótu. Większa liczba iteracji ułatwia obronę przed próbami ustalenia nazw właścicieli odpowiadających wartościom zapisanym w rekordach NSEC3 (eliminacja ataków słownikowych). Pole *Długość wartości inicjującej* (*salt length*) odzwierciedla długość pola *Wartość inicjująca* (*salt*). Samo pole *Wartość inicjująca* przechowuje wartość dołączaną do nazwy właściciela przed obliczeniem wartości skrótu. Celem operacji jest utrudnienie ataków słownikowych.

Drugi rekord zasobów opisany w dokumencie [RFC5155] to NSEC3PARAM (rekord ten nie został osobno opisany). Jego budowa odpowiada formatowi rekordu NSEC3 z pewnym wyjątkiem — nie ma w nim pól *Długość skrótu* (*hash length*), *Następny skrót nazwy właściciela* (*next hashed owner*) oraz *Mapy bitowe typów*. Rekord ten jest wykorzystywany przez autorytatywny serwer nazw podczas wybierania rekordów NSEC3 w negatywnej odpowiedzi. Dostarcza parametry niezbędne do wyliczenia skrótu nazwy właściciela.

Aby otrzymać wartość pola *Skrót następnej nazwy właściciela*, wykonywana jest następująca operacja:

IH(0) = H(nazwa właściciela | *Wartość inicjująca*)

IH(k) = H(IH(k–1) | *Wartość inicjująca*), jeśli k > 0

Następny skrót nazwy właściciela = H(IH(*Iteracje*) | *Wartość inicjująca*)

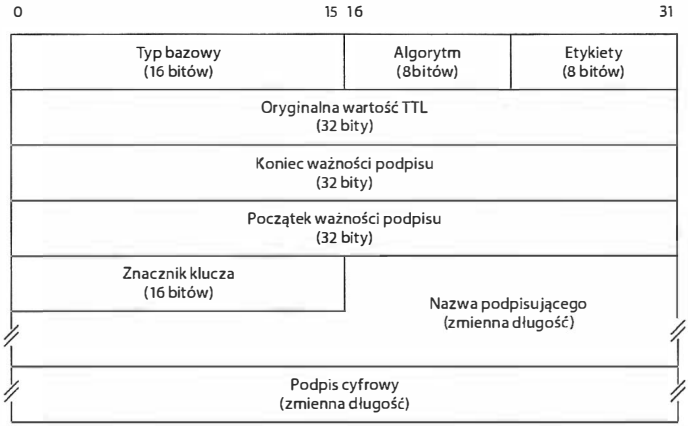
w której H reprezentuje funkcję opisaną w polu *Algorytm skrótu*, a nazwa właściciela jest podawana w formie kanonicznej. Liczba iteracji oraz wartość inicjująca pochodzą z odpowiednich pól rekordu NSEC3.

W celu uniknięcia niejednoznaczności między rekordami NSEC i NSEC3 w dokumencie [RFC5155] zapisano obowiązek użycia specjalnych identyfikatorów algorytmów zabezpieczających o wartościach 6 i 7 jako aliasów dla identyfikatorów 3 (DSA) i 5 (SHA-1). Wspomniane aliasy są stosowane jedynie w rekordach NSEC3. Resolvery nieobsługujące rekordów NSEC3 po odebraniu informacji z podanymi wyróżnikami traktują pozyskane rekordy jako niepewne. Takie rozwiązanie gwarantuje pewną formę zgodności z wcześniejszymi rozwiązaniami (operacja kończy się niepowodzeniem, ale nie powoduje błędnego zinterpretowania danych).

18.10.1.4. Sygnatura rekordu zasobów (RRSIG)

Po zabezpieczeniu struktury DNS należy zastanowić się nad ochroną zawartości plików stref. Celem administratorów serwerów jest zagwarantowanie autentyczności i integralności przesyłanych rekordów zasobów. Mechanizm DNSSEC podpisuje i weryfikuje sygnatury grupy RRset, używając do tego celu **sygnatur rekordów zasobów** (RRSIG — *Resource Record Signature*). Każdy autorytatywny rekord strefy musi zostać podpisany

(wyjątkiem są rekordy sklejające oraz delegacje NS). Rekord RRSIG przechowuje podpis cyfrowy określonej grupy RRset wraz z informacją umożliwiającą określenie klucza publicznego niezbędnego do sprawdzenia sygnatury. Struktura rekordu RRSIG została pokazana na rysunku 18.41.



Rysunek 18.41. *Pole RDATA rekordu RRSIG zawiera podpis cyfrowy określonej grupy RRset. Obejmuje również wartość TTL zbioru rekordów (o takiej samej wartości, jaka występuje w strefie autorytatywnej), a także informacje o zastosowanym algorytmie i okresie ważności sygnatury. Pole Znacznik klucza odnosi się do rekordu DNSKEY przechowującego klucz publiczny potrzebny do sprawdzenia podpisu. Pole Etykiety określa, ile etykiet składa się na oryginalną nazwę rekordu*

Pole *Typ bazowy* (*type covered*) reprezentuje typ grupy RRset, do którego odnosi się sygnatura. Wartość pochodzi ze standardowego rejestru typów zdefiniowanego w [DNSPARAMS]. Wartość *Algorytm* odzwierciedla rodzaj algorytmu podpisu cyfrowego. Zgodnie z opracowaniem [RFC4034] w rekordach RRSIG podczas tworzenia rekordów RRSIG używane mogą być jedynie mechanizmy DSA i RSA wraz z algorytmem SHA-1 (o wartościach 3 i 5). Jednak w dokumencie [RFC5702] uwzględniono również algorytm SHA-2, a w zaleceniu [RFC5933] wymieniono także mechanizm GOST (z Rosji). Pole *Etykiety* przechowuje liczbę etykiet oryginalnej nazwy właściciela rekordu RRSIG. Parametr *Oryginalna wartość TTL* jest kopią wartości TTL wyznaczonej dla grupy RRset w strefie autorytatywnej (serwery buforujące mogą zmniejszać wartość TTL). Pola *Początek ważności podpisu* (*signature inception*) i *Koniec ważności podpisu* (*signature expiration*) wyznaczają okres ważności, którego granice są wyrażane liczbą sekund, jakie upłynęły od godziny 00:00:00 UTC 1 stycznia 1970 roku. Wartość *Znacznik klucza* ułatwia identyfikację rekordu DNSKEY, potrzebnego do pozyskania klucza publicznego. Klucz ten służy do weryfikacji sygnatury zapisanej w polu *Podpis cyfrowy*. Format podpisu jest taki sam jak w rekordzie DS.

18.10.2. Działanie mechanizmu DNSSEC

Znając wszystkie rekordy zasobów zdefiniowane w standardzie DNSSEC, możemy zająć się analizą zasad funkcjonowania mechanizmu. Najpierw ustalmy znaczenie terminu **porządek kanoniczny** (*canonical ordering*), który został użyty wielokrotnie w opisie rekordów NSEC i NSEC3. Celem definiowania porządku kanonicznego jest określenie ta-

kiej techniki listowania zawartości pliku strefy, aby uzyskiwane wyniki były powtarzalne niezależnie od środowiska, a co się z tym wiąże, aby można było używać podpisu cyfrowego (różna kolejność wpisów powodowałaby generowanie różnych wartości skrótu nawet w przypadku zastosowania tej samej funkcji). Dalej w tym punkcie omówiony został przebieg operacji podpisywania strefy oraz weryfikacji podpisu.

18.10.2.1. Porządek kanoniczny i formy reprezentacji rekordów

Podczas analizy działania mechanizmu DNSSEC można napotkać trzy obszary zastosowań porządku kanonicznego: sortowanie nazw kanonicznych w ramach strefy, wyznaczanie kanonicznej postaci pojedynczego rekordu RR, sortowanie kanoniczne w ramach grupy `RRset` [RFC4034]. Z informacji zamieszczonych w rozdziale 11. wynika, że każdy rekord ma przypisaną nazwę właściciela (nazwę domenową właściciela) składającą się z szeregu etykiet. Traktując każdą z etykiet jak wyrównany do lewej strony ciąg bajtów i przekształcając wielkie litery z zestawu ASCII na małe, otrzymujemy listę nazw. Listę tę sortujemy na podstawie najbardziej znaczącej etykiety (występującej na samym końcu nazwy), następnie na podstawie przedostatniej etykiety (mniej znaczącej) itd. Wpisy pozbawione etykiety na danym poziomie powinny być zapisywane przed wpisami o zerowej wartości bajtu. Otrzymujemy w ten sposób kanoniczny porządek nazw. Oto przykład: `com`, `firma.com`, `*.firma.com`, `PL.firma.COM`, `usa.firma.com`. Dopuszczalne jest stosowanie symboli wieloznacznych.

Kanoniczna forma zapisu odnosi się również do sposobu reprezentacji każdego z rekordów `RR`. Rekordy zapisane w formie kanonicznej muszą spełniać następujące reguły.

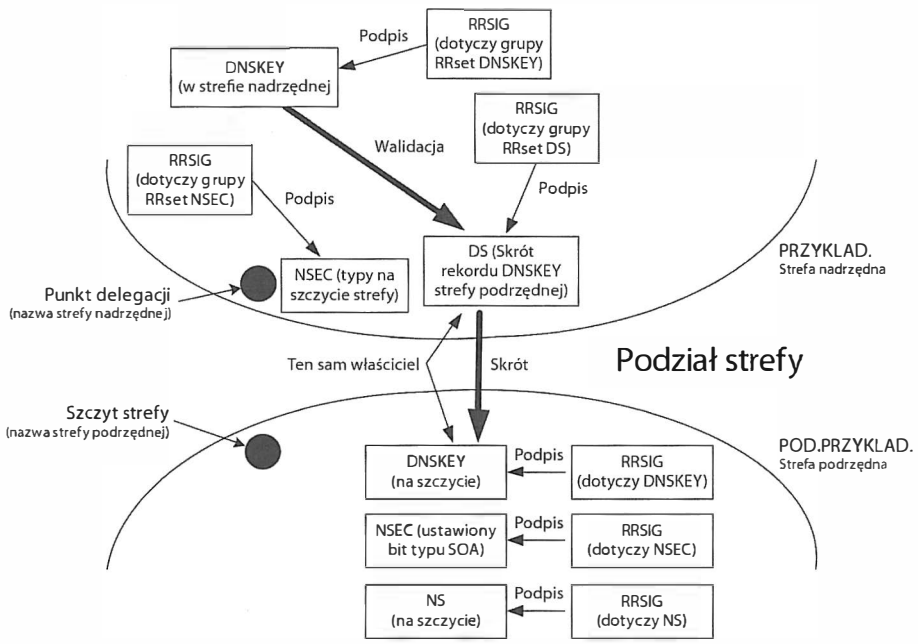
- Wszystkie nazwy domenowe muszą być zapisane w formie FQDN i z rozwiniętymi elementami (bez skompresowanych etykiet).
- Wszystkie wielkie litery występujące w nazwie właściciela muszą zostać zastąpione odpowiadającymi im małymi literami.
- Wszystkie wielkie litery muszą zostać zastąpione małymi, jeśli nazwa domeny występuje w sekcji RDATA rekordów o typach: 2 – 9, 12, 14, 15, 17, 18, 21, 24, 26, 33, 35, 36, 39 oraz 38.
- Symbole wieloznaczne nie są zastępowane.
- Wartość TTL musi odpowiadać oryginalnej wartości TTL zapisanej w strefie autorytatywnej lub wartości pola *Oryginalna wartość TTL* w rekordzie `RRSIG`.

W dokumentach z grupy DNSSECbis wprowadzono wiele wyjaśnień i zmian, dlatego poszukując szczegółowych informacji na ten temat, warto zapoznać się z najnowszą wersją opracowania [IDDCIN].

Kanoniczny porządek rekordów zasobów (`RR`) w grupie `RRset` odpowiada regułom obowiązującym podczas wyznaczania kolejności nazw właściciela, ale odnosi się do zawartości pola *RDATA* (traktowanego jak wyrównany do lewej strony ciąg bajtów).

18.10.2.2. Podpisane strefy i podział strefy

Działanie mechanizmu DNSSEC bazuje na podpisywaniu stref. W strefach tych przechowywane są rekordy RRSIG, DNSKEY oraz NSEC (bądź NSEC3), a także niekiedy rekordy DS (jeśli dana strefa obejmuje również podpisany punkt delegacji). Podpisywanie wymaga zastosowania algorytmów kryptograficznych wykorzystujących klucze publiczne, które są przechowywane w systemach DNS i przez nie dystrybuowane. Na rysunku 18.42 przedstawiono przykładowy punkt delegacji opisujący odwołanie ze strefy nadrzędnej do strefy podrzędnej.



Rysunek 18.42. *Podział strefy w przypadku uwierzytelnionej delegacji oznacza, że w strefie nadrzędnej znajduje się rekord DS przechowujący skrót rekordu DNSKEY ze strefy podrzędnej. Wszystkie grupy RRset są podpisywane z użyciem odpowiednich rekordów RRSIG poza rekordem delegacji NS (oraz rekordami sklejającymi) w strefie nadrzędnej. Rekordy NSEC mogą zostać wykorzystane do weryfikacji typów obecnych w strefie. Mogą więc wskazywać typ SOA na szczycie strefy podrzędnej*

Z rysunku wynika, że strefa nadrzędna zawiera własny rekord DNSKEY, który dostarcza klucz publiczny odpowiadający kluczowi prywatnemu wykorzystanemu wraz z rekordem RRSIG do podpisania wszystkich autorytatywnych grup RRset (możliwe jest zdefiniowanie wielu rekordów DNSKEY). Występujący w strefie nadrzędnej rekord DS przechowuje skrót jednego z rekordów DNSKEY strefy podrzędnej. Dzięki temu tworzony jest łańcuch zaufania między strefą nadrzędną i podrzędną. Jeśli resolver walidujący uzna za zaufany rekord DS strefy nadrzędnej, może zweryfikować rekord DNSKEY strefy podrzędnej, a w konsekwencji również rekordy RRSIG oraz podpisane grupy RRset w strefie podrzędnej. Taka sytuacja zachodzi jednak tylko wówczas, gdy walidator może ustanowić relację zaufania z rekordem DNSKEY strefy nadrzędnej.

18.10.2.3. Przykład działania resolvera

Znając łańcuch powiązań między podpisanymi strefami i dysponując resolverem walidującym, możemy przeanalizować proces weryfikacji odpowiedzi generowanych przez serwery DNS. W najlepszym przypadku informacje o danej strefie można uzyskać, podążając wzdłuż łańcucha zaufania od strefy głównej. Organizacja ICANN utrzymuje wykaz stref zgodnych z mechanizmem DNSSEC, zapisując rekordy DS w strefach głównych oraz podpisane rekordy DNSKEY [TLD-REPORT].

Za cel działań przyjmijmy odwzorowanie i weryfikację rekordu typu A odpowiadającego nazwie domenowej www.icann.org. Zadanie musimy rozpocząć od wyznaczenia kotwicy zaufania (tj. rekordu DNSKEY) w strefie głównej. Następnie pobierzemy rekordy DS odnoszące się do domeny org., które są zapisane w jednym z głównych serwerów nazw. Odczytamy również rekordy RRSIG i NSEC (NSEC3). Ten sam proces powtórzymy w odniesieniu do domen org. i icann.org. oraz odpowiadających im serwerów DNS. Zacznijmy więc od odwołań do strefy głównej:

```
Linux% dig @a.root-servers.net. . dnskey +noquestion +nocomments +nostats +multiline
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.7.2-P3 <<>> @a.root-servers.net. . dnskey +noquestion +nocomments
+nostats +multiline
; (1 server found)
;; global options: +cmd
.   86400  IN     DNSKEY       257 3 8 ( AwEAAagAIKl ... ) ; key id = 19036
.   86400  IN     DNSKEY       256 3 8 ( AwEAAb5gVAz ... ) ; key id = 21639
.   86400  IN     DNSKEY       256 3 8 ( AwEAAcAPhPM ... ) ; key id = 40288
```

Powyższy listing wyznacza kotwicę zaufania dla strefy głównej, która jest jednocześnie punktem startowym łańcucha zaufania we wszystkich operacjach DNSSEC w Internecie. Pierwszy z kluczy jest kluczem typu KSK, o czym świadczy wartość 257 (bit SEP ma wartość 1). Ten właśnie klucz powinien zostać wykorzystany do utworzenia łańcucha zaufania. Pozostałe klucze są wartościami ZSK. Kolejnym krokiem jest sprawdzenie, czy wszystkie widoczne na liście rekordy rzeczywiście powinny być na niej uwzględnione oraz czy mają odpowiednie podpisy. Do weryfikacji potrzebne będą rekordy RRSIG:

```
Linux% dig @a.root-servers.net. . rrsig +noquestion +nocomments +nostats +noauthority \
 +noadditional
;; Truncated, retrying in TCP mode.

; <<>> DiG 9.7.2-P3 <<>> @a.root-servers.net. . rrsig +noquestion+nocomments +nostats
+noauthority +noadditional
; (1 server found)
;; global options: +cmd
.  86400  IN   RRSIG NSEC 8 0 86400 20101228000000 20101220230000
                          40288 . RyoG81dxxX...
.  86400  IN   RRSIG DNSKEY 8 0 86400 20110105235959 20101221000000
                            19036 . f8bzNvPmHR...
...
```

Rekord RRSIG odnoszący się do rekordu DNSKEY przechowuje identyfikator klucza o wartości 19306, czyli takiej, jaka jest widoczna w wierszu klucza KSK (w rekordzie DNSKEY) strefy głównej. Strefa główna zawiera również inne rekordy RRSIG (odnoszące się do rekordów SOA i NS), ale w realizowanym zadaniu ważniejsze są wpisy dotyczące rekordów

DNSKEY oraz NSEC. Aby zyskać całkowitą pewność, że rekord DNSKEY powinien być dostępny w strefie, trzeba zweryfikować listę typów rekordów zapisaną w rekordzie NSEC strefy głównej:

```
Linux% dig @a.root-servers.net. . nsec +noquestion +nocomments +nostats +noauthority \
 +noadditional
; <<>> DiG 9.7.2-P3 <<>> @a.root-servers.net. . nsec +noquestion +nocomments +nostats
+noauthority +noadditional
; (1 server found)
;; global options: +cmd
. 86400 IN NSEC ac. NS SOA RRSIG NSEC DNSKEY
```

Z powyższego listingu wynika, że strefa główna powinna obejmować rekordy o typach NS, SOA, RSSIG, NSEC oraz DNSKEY. Oznacza to, że dotychczasowa walidacja przebiegła pomyślnie (jak nietrudno zauważyć domena ac. jest pierwszą domeną najwyższego poziomu na liście ułożonej w porządku kanonicznym). Następna czynność polega na sprawdzeniu podpisów delegacji ze strefy głównej do strefy org. Oto stosowne polecenie:

```
Linux% dig @a.root-servers.net. org. rrsig +noquestion +nocomments +nostats \
 +noadditional +dnssec
; <<>> DiG 9.7.2-P3 <<>> @a.root-servers.net. org. rrsig +noquestion +nocomments
+nostats +noadditional +dnssec
; (1 server found)
;; global options: +cmd
org.    172800   IN        NS        d0.org.afilias-nst.org.
org.    172800   IN        NS        b2.org.afilias-nst.org.
org.    172800   IN        NS        a0.org.afilias-nst.info.
org.    172800   IN        NS        b0.org.afilias-nst.org.
org.    172800   IN        NS        a2.org.afilias-nst.info.
org.    172800   IN        NS        c0.org.afilias-nst.info.
org.    86400    IN        DS        21366 7 2 96EEB2FFD9 ...
org.    86400    IN        DS        21366 7 1 E6C1716CF8 ...
org.    86400    IN RRSIG  DS 8 1 86400 20101228000000 20101220230000
                           40288 . jpcJOGclvvlnx9Kvz5 ...
```

Dostępność grupy RRset DS oraz związanego z nią rekordu RRSIG sugeruje, że rzeczywiście mamy do czynienia z delegacją zabezpieczoną za pomocą mechanizmu DNSSEC. Wpis RRSIG obejmuje identyfikator klucza o wartości 40288. Jest to odniesienie do trzeciego rekordu DNSKEY z prezentowanego wcześniej listingu strefy głównej (do klucza ZSK). Rekordy NS dostarczają informacji o kolejnych serwerach, z których można skorzystać w następnych etapach procedury testowej. Powtórzmy zatem polecenia wykonane w odniesieniu do strefy głównej, ale tym razem w odniesieniu do domeny org. Stosowne zapytania zostaną skierowane do jednego z serwerów wymienionych w rekordach NS strefy głównej, ale odnoszących się do domeny org.:

```
Linux% dig @d0.org.afilias-nst.org. org. dnskey +dnssec +nostats +noquestion
+multiline
; <<>> DiG 9.7.2-P3 <<>> @d0.org.afilias-nst.org. org. dnskey +dnssec
+nostats +noquestion +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8061
;; flags: qr aa rd; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; ANSWER SECTION:
org. 900 IN  DNSKEY     256 3 7 ( AwEAAZTErUF ... ) ; key id = 1743
org. 900 IN  DNSKEY     256 3 7 ( AwEAAazTpnm ... ) ; key id = 43172
org. 900 IN  DNSKEY     257 3 7 ( AwEAAYpYfj3 ... ) ; key id = 21366
org. 900 IN  DNSKEY     257 3 7 ( AwEAAZTjbIO ... ) ; key id = 9795
org. 900 IN  RRSIG DNSKEY 7 1 900 20101231154644
                          20101217144644 21366 org.
                          aIZgEsoJO+Q8ZXM ...
org. 900 IN  RRSIG DNSKEY 7 1 900 20101231154644
                          20101217144644 43172 org. MWWosWBdEmM8CiM ...
```

Z listingu wynika, że zdefiniowane zostały cztery rekordy DNSKEY, z których dwa odpowiadają kluczom KSK (wartość 257), a dwa reprezentują klucze ZSK (wartość 256). Trzeci z wpisów (21366) odnosi się do rekordu DS zawartego w prezentowanej wcześniej strefie głównej. Klucz ten został wykorzystany w rekordzie RRSIG, podobnie jak klucz ZSK o identyfikatorze 43172. Aby ustalić, czy ich występowanie na liście jest dozwolone, wystarczy sprawdzić rekordy NSEC lub NSEC3 w strefie org.:

```
Linux% dig @d0.org.afilias-nst.org. org. nsec +dnssec +nostats +noquestion
; <<>> DiG 9.7.2-P3 <<>> @d0.org.afilias-nst.org. nsec org. +dnssec
+nostats +noquestion
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61632
;; flags: qr aa rd; QUERY: 1, ANSWER: 0, AUTHORITY: 4, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; AUTHORITY SECTION:
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org. 86400 IN NSEC3 1 1 1
                                     D399EAAB
                                     H9RSFB7FPF2L8HG35CMPC765TDK23RP6
                                     NS SDA RRSIG DNSKEY NSEC3PARAM
h9p7u7tr2u91d0v0ljs9l1gidnp90u3h.org. 86400 IN RRSIG NSEC3 7 2
                                     86400 20110105003654
                                     20101221233654
                                     43172 org. eBtna4fok ...
```

W zestawieniu wynikowym znajduje się rekord NSEC3 z nazwą właściciela równą skrótowi domeny org. Występują również rekordy DNSKEY i RRSIG, a także rekordy NS i NS i NSEC3PARAM. Wykorzystując ostatni z wymienionych typów rekordów, możemy pozyskać więcej informacji na temat rekordu NSEC3:

```
Linux% .dig @a0.org.afilias-nst.info. org. nsec3param +dnssec +nostats +noadditional \
 +noauthority +noquestion
; <<>> DiG 9.7.2-P3 <<>> @a0.org.afilias-nst.info. org. nsec3param +dnssec +nostats
+noadditional +noauthority +noquestion
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 38602
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 7, ADDITIONAL: 13
;; WARNING: recursion requested but not available
```

```
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags: do; udp: 4096
;; ANSWER SECTION:
org.            900     IN      NSEC3PARAM 1 0 1 D399EAAB
org.            900     IN      RRSIG NSEC3PARAM 7 1 900 20101231154644
                                20101217144644 43172 org. fS2kFw53elY ...
```

Jednakowa wartość podpisu (D399EAAB) świadczy o tym, że rekordy NSEC3PARAM i NSEC3 odpowiadają sobie nawzajem. Z zestawienia wynika również, że sygnatura zapisana w rekordzie RRSIG została wyznaczona z użyciem klucza prywatnego skojarzonego z rekordem DNSKEY o identyfikatorze 43172. Zgodność sygnatur potwierdza poprawność łańcucha zaufania. Aby dokończyć zadanie, trzeba pobrać informacje na temat domeny icann.org.:

```
Linux% dig @a0.org.afilias-nst.info. icann.org. any +dnssec +nostats +noadditional
; <<>> DiG 9.7.2-P3 <<>> @a0.org.afilias-nst.info. icann.org. any +dnssec +nostats
+noadditional
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 61234
;; flags: qr rd; QUERY: 1, ANSWER: 0, AUTHORITY: 8, ADDITIONAL: 3
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;icann.org.  IN ANY
;; AUTHORITY SECTION:
icann.org.          86400 IN   NS    a.iana-servers.net.
icann.org.          86400 IN   NS    b.iana-servers.org.
icann.org.          86400 IN   NS    c.iana-servers.net.
icann.org.          86400 IN   NS    d.iana-servers.net.
icann.org.          86400 IN   NS    ns.icann.org.
icann.org.          86400 IN   DS    41643 7 1 93358DB ...
icann.org.          86400 IN   DS    41643 7 2 B8AB67D ...
icann.org.          86400 IN   RRSIG DS 7 2 86400 20101231154644
                               20101217144644 43172 org. cZ1Z30w// ...
```

Listing obejmuje rekord DS definiujący podpisaną delegację do strefy icann.org. ze strefy org. Rekord RRSIG powiązany z grupą DS został podpisany za pomocą klucza ZSK o identyfikatorze 43172. Wykorzystując jeden z adresów zapisanych w rekordach NS, możemy wygenerować żądania do ostatecznego serwera:

```
Linux% dig @a.iana-servers.net. icann.org. dnskey +dnssec +nostats +noquestion \
 +multiline
; <<>> DiG 9.7.2-P3 <<>> @a.iana-servers.net. icann.org. dnskey +dnssec +nostats
+noquestion +multiline
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22065
;; flags: qr aa rd; QUERY: 1, ANSWER: 5, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; ANSWER SECTION:
```

```
icann.org. 3600 IN  DNSKEY 256 3 7 ( AwEAAbDmrVc ... ) ; key id = 41295
icann.org. 3600 IN  DNSKEY 256 3 7 ( AwEAAbgrYZd ... ) ; key id = 55469
icann.org. 3600 IN  DNSKEY 257 3 7 ( AwEAAZuSdr4 ... ) ; key id = 7455
icann.org. 3600 IN  DNSKEY 257 3 7 ( AwEAAcyguBH ... ) ; key id = 41643
icann.org. 3600 IN  RRSIG DNSKEY 7 2 3600 20101229153632
                    20101222042536 41643 icann.org.
                    UxR/5vyOIS ...
```

Z listingu wynika, że istnieją cztery rekordy DNSKEY — dwa typu KSK i dwa ZSK. Czwarty z wpisów (41643) jest powiązany z rekordem DS zapisanym w strefie org. Jest on także wykorzystywany w rekordzie RRSIG. Aby ustalić ostateczną odpowiedź na początkowe zapytanie, wystarczy zażądać dostarczenia rekordu A:

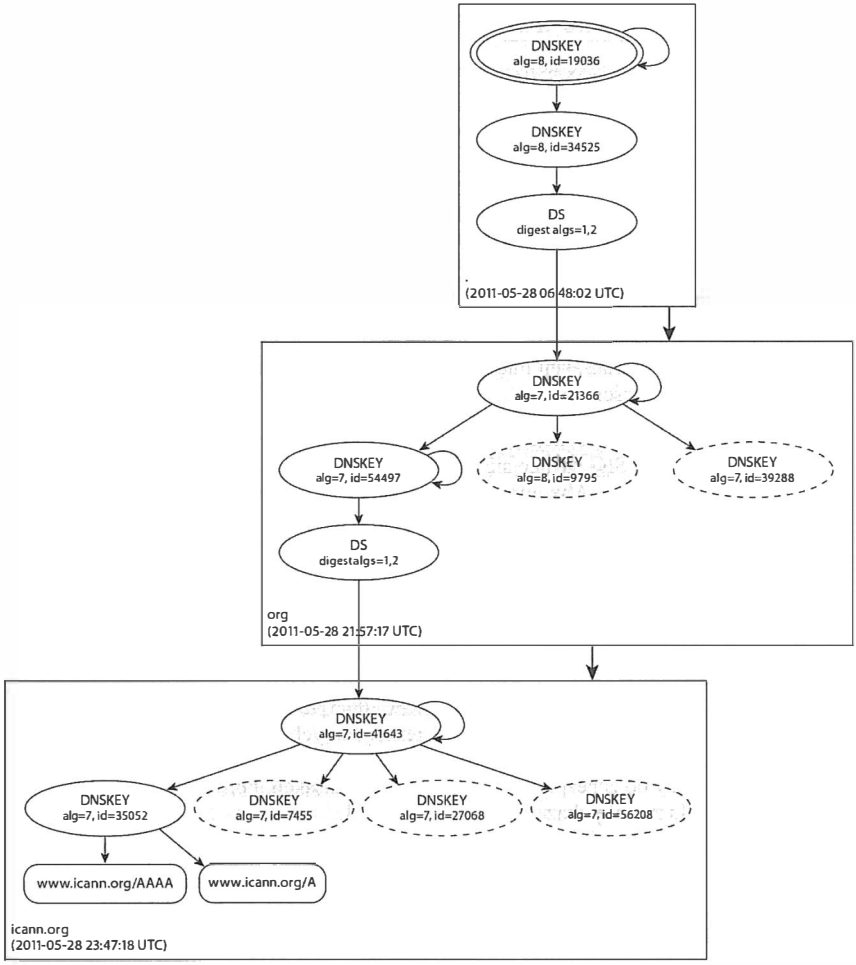
```
Linux% dig @a.iana-servers.net. www.icann.org. a +dnssec +nostats +noquestion \
 +noauthority +noadditional
; <<>> DiG 9.7.2-P3 <<>> @a.iana-servers.net. www.icann.org. a +dnssec +nostats
+noquestion +noauthority +noadditional
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56258
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 6, ADDITIONAL: 3
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; ANSWER SECTION:
www.icann.org.          600   IN   A     192.0.32.7
www.icann.org.          600   IN   RRSIG A 7 3 600 20101229143630
                                   20101222042536 55469 icann.org.
                                   YRhlL/RA ...
```

W końcu uzyskaliśmy poszukiwany rekord *A* (odpowiadający adresowi *www.icann.org*). Przechowuje on adres IP 192.0.32.7, który został podpisany za pomocą rekordu RRSIG z użyciem klucza o identyfikatorze 55469. Jest to klucz z czwartego rekordu DNSKEY strefy icann.org. Na tym etapie prac wszystkie uzyskane informacje wydają się poprawne. Jednak nie sprawdziliśmy jeszcze, czy prezentowane sygnatury są poprawne. Walidacja podpisów należy do zadań następującego polecenia:

```
Linux% dig @a.root-servers.net. www.icann.org. a +sigchase +topdown \
 +trusted-key=trusted-keys
```

Powyższa instrukcja zadziała poprawnie tylko w przypadku, w którym program dig został skompilowany z opcją -DDIG_SIGCHASE=1 i plik trusted-keys zawiera grupę rekordów DNSKEY strefy głównej. Wśród wielu wierszy listingu wynikowego można znaleźć jedną, która informuje o powodzeniu operacji. Znacznie łatwiejsze okazuje się wykorzystanie do tego celu aplikacji internetowej dostępnej pod adresem *http://dnsviz.net*. Wynik wykonania zapytania został przedstawiony na rysunku 18.43.

Na rysunku przedstawiamy udaną weryfikację rekordów typu A i AAAA odnoszących się do domeny o nazwie www.icann.org. Każdy z prostokątów reprezentuje strefę o podanej nazwie (oraz dacie analizy). Elipsy znajdujące się wewnątrz prostokątów symbolizują węzły w łańcuchu zaufania — rekordy DNSKEY lub DS. Przerywana linia obrysu elipsy stanowi informację o tym, że dany klucz nie został użyty do wygenerowania sygnatur



Rysunek 18.43. *Wizualizacja łańcucha zaufania wyznaczanego przez mechanizm DNSSEC. Prostokąty oznaczają strefy. Elipsy reprezentują węzły łańcucha, a zacienione elipsy odpowiadają węzłom z ustawionym bitem SEP. Poprawne rekordy RRSIG oraz skróty DS są symbolizowane za pomocą strzałek. Elipsa o podwójnym obrysie jest kotwicą zaufania*

istotnych dla wykonania zadania. Strzałki między elipsami wskazują rekordy `RRSIG` i `DS`. Ponadto na podstawie rysunku można stwierdzić, że wykorzystywane są dwa rodzaje algorytmów. Przykładowo w strefie głównej jest używany mechanizm RSA/SHA-1 [RFC5702], o czym świadczy zapis `alg = 8`. Dzięki czemu możliwe jest zastosowanie rekordów NSEC3 [RFC5155]. Natomiast zapis `digest algs = 1,2` informuje, że do utworzenia rekordu `DS` wykorzystano rozwiązania SHA-1 [RFC4034] oraz SHA-256 [RFC4509].

18.10.3. Uwierzytelnianie transakcji (TSIG, TKEY oraz SIG(0))

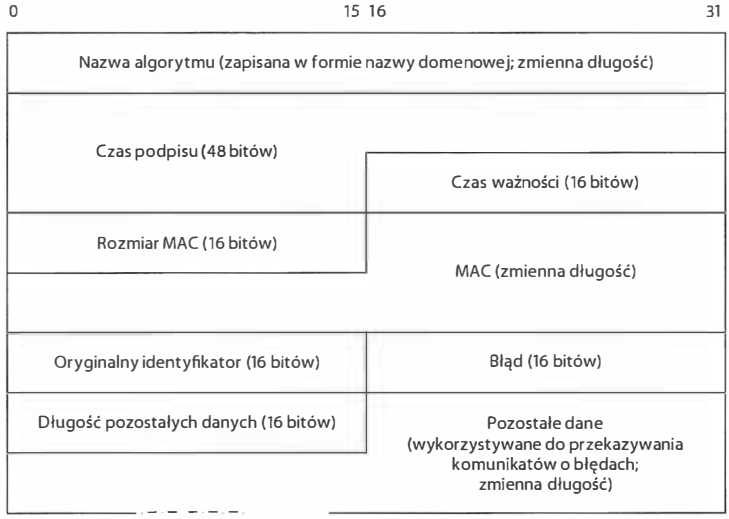
Niektóre transakcje DNS (np. transfer strefy lub dynamiczne aktualizacje) mogą doprowadzić do uszkodzenia struktury strefy lub jej zawartości (jeśli określona operacja zostanie wykonana nieprawidłowo). Niezbędne okazuje się więc wdrożenie pewnych form uwierzytelnienia. Jest ono stosowane również w standardowych odwzorowaniach DNS — gdy resolver spodziewa się potwierdzonych odpowiedzi DNS, ale nie obsługuje standardu DNSSEC. Uwierzytelnianie transakcji stanowi formę zabezpieczenia wymiany danych między resolverem a serwerem (lub między serwerami). Trzeba jednak pamiętać, że nie służy ono bezpośrednio do ochrony treści plików stref. Za to odpowiada mechanizm DNSSEC. Rozwiązania DNSSEC i uwierzytelnianie transakcji uzupełniają się wzajemnie i mogą być wdrażane łącznie. Pierwsze z nich gwarantuje autentyczność źródła danych oraz integralność danych strefy. Natomiast drugie zapewnia integralność i uwierzytelnienie określonej transakcji między klientem i serwerem bez weryfikowania poprawności wymienianych treści.

Uwierzytelnianie transakcji DNS bazuje na dwóch mechanizmach: TSIG oraz SIG(0). W rozwiązaniu TSIG stosuje się klucze współdzielone, natomiast w SIG(0) pary kluczy publiczny-prywatny. Aby uprościć wdrożenie systemu bazującego na jednym z wymienionych standardów, parametry potrzebne do formowania kluczy (np. wartość DH) można przechowywać w rekordach TKEY.

18.10.3.1. TSIG

Protokół **uwierzytelniania transakcji DNS na podstawie tajnego klucza** (*Secret Key Transaction Authentication for DNS*) nazywany również protokołem **sygnatur transakcji** (TSIG — *Transction Signatures*) [RFC 2845] uzupełnia komunikację DNS o funkcję uwierzytelniania z użyciem sygnatur generowanych na podstawie wspólnego klucza. Działanie mechanizmu TSIG bazuje na pseudorekordzie TSIG, który jest wyznaczany na żądanie i służy do zabezpieczenia pojedynczej transakcji. Format pola *RDATA* wspomnianego rekordu został pokazany na rysunku 18.44.

Na rysunku przedstawiono format pseudorekordu TSIG. Rekordy tego typu są przesyłane w sekcji dodatkowych danych żądania lub odpowiedzi DNS. Pierwotny algorytm MAC (opisany w dokumencie [RFC2845) bazował na rozwiązaniu HMAC-MD5. Z czasem wprowadzono jednak nowsze rozwiązania GSS-API (Kerberos) [RFC3645] oraz SHA-1 i SHA-256 [RFC4635]. Wykaz aktualnie obowiązujących mechanizmów znajduje się w dokumencie [TSIGALG]. Początkowo nazwy algorytmów kodowano w sposób odpowiadający nazwom domenowym (np. HMAC-MD5.SIG-ALG.REG.INT). Obecnie jednak stosuje się bardziej opisowe ciągi tekstowe (np. hmac-sha1, hmac-sha256). Zajmujące 48 bitów pole *Czas podpisu* (*signed time*) odpowiada uniksowemu formatowi czasu (liczbie sekund, które upłynęły od 1 stycznia 1970 roku) i określa moment podpisania treści wiadomości. Wartość pola również jest objęta podpisem i umożliwia wykrywanie ataków z ponawianiem pakietów oraz przeciwdziałanie im. Konsekwencją użycia bezwzględnej wartości czasu jest konieczność przeprowadzania komunikacji TSIG w okresie, którego liczba sekund została zdefiniowana w polu *Czas ważności* (*fudge*). Pole *Rozmiar MAC* (*MAC size*) przechowuje liczbę bajtów składających się na pole *MAC*. Jego wartość jest zależna od rodzaju użytego algorytmu podpisu. Pole *Długość pozostałych danych* (*other size*) wyznacza rozmiar pola *Pozostałe dane* (*other data*) w bajtach. Samo pole *Pozostałe dane* jest wykorzystywane jedynie do przenoszenia informacji o błędach.



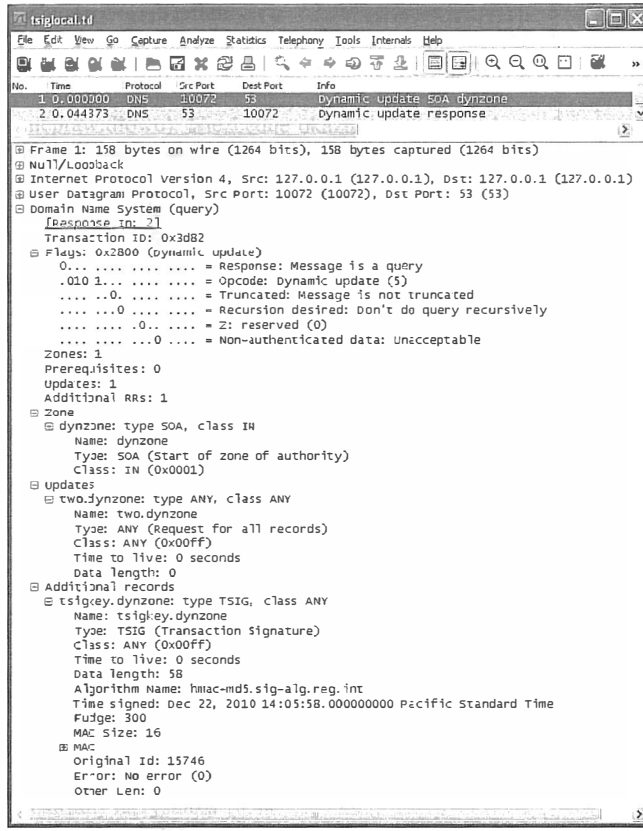
Rysunek 18.44. *Pole RDATA pseudorekordu TSIG zawiera identyfikator algorytmu podpisu cyfrowego, czas wygenerowania podpisu, czas ważności oraz wartość MAC. Początkowo do generowania sygnatur wykorzystywano jedynie algorytm MD5. Obecnie standard obejmuje również rozwiązania SHA-1 i SHA-2. Synchronizacja jednostek TSIG musi zostać przeprowadzona w czasie wyznaczanym przez pole Czas ważności (wyrażonym w sekundach). Rekord TSIG jest przenoszony w dodatkowym polu danych komunikatu DNS*

Aby przeanalizować działanie protokołu TSIG w praktyce, przygotowaliśmy przykładową strefę o nazwie `dynzone.` i przeprowadziliśmy dynamiczną aktualizację jej zawartości z użyciem podpisu cyfrowego. Do aktualizacji został wykorzystany program `nsupdate` wchodzący w skład pakietu BIND9.

```
Linux% nsupdate
> zone dynzone.
> server 127.0.0.1
> key tsigkey.dynzone. 1234567890abcdef
> update delete two.dynzone.
> send
```

Powyższe instrukcje odpowiadają za utworzenie komunikatu aktualizacji strefy DNS, który jest następnie podpisywany zgodnie ze standardem TSIG. Przesłanie komunikatu do serwera następuje po wpisaniu polecenia `send`. Treść żądania została pokazana na rysunku 18.45.

Dynamiczna aktualizacja DNS została podpisana za pomocą algorytmu HMAC-MD5 z użyciem klucza o nazwie `tsigkey.dynzone.` Żądanie ma na celu zaktualizowanie strefy `dynzone.` przez usunięcie wpisu `two.dynzone.` Z rysunku wynika, że do podpisania wiadomości wykorzystano algorytm oznaczony jako HMAC-MD5.SIG-ALG.REG.INT, który jest jedynym mechanizmem podpisu obsługiwanym przez program narzędziowy. Warto zwrócić uwagę na to, że pole *Oryginalny identyfikator* (*Original ID*; o wartości dziesiętnej 15746) odpowiada wartości *Identyfikatora transakcji* (*Transaction ID*; o wartości 0x3d82) przedstawionej na rysunku 18.46.

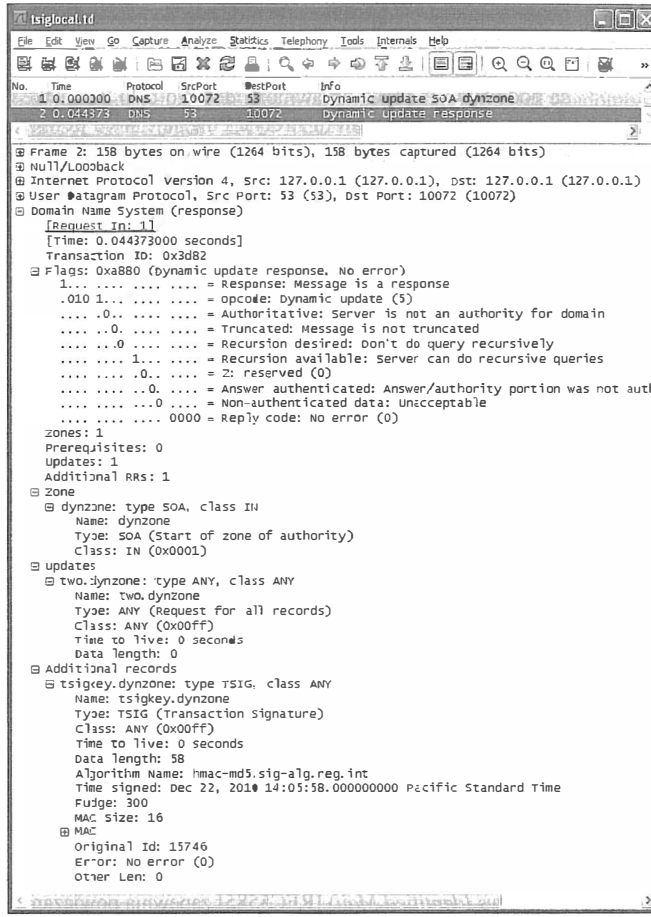


Rysunek 18.45. *Dynamiczna aktualizacja DNS podpisana zgodnie ze standardem TSIG. Żądanie ma na celu usunięcie rekordu two.dynzone. Zostało podpisane za pomocą klucza o nazwie tsigkey.dynzone. Jako algorytm podpisu wykorzystano funkcję HMAC-MD5, która generuje sygnaturę o 128-bitowej (16-bajtowej) długości*

Na rysunku 18.46 widoczna jest odpowiedź serwera DNS na żądanie dynamicznej aktualizacji. Również ona została podpisana zgodnie z założeniami mechanizmu TSIG. Pole *Znaczniki* (*Flags*) informuje, że operacji nie towarzyszyły żadne błędy. Również w tym przypadku pseudorekord TSIG został zapisany w dodatkowym polu informacyjnym.

18.10.3.2. SIG(0)

W początkowych wersjach mechanizmu DNSSEC stosowano rekordy sygnatur (SIG) o podobnym znaczeniu, jakie mają wykorzystywane obecnie rekordy RRSIG. Jeden z wariantów rekordów SIG, o nazwie SIG(0) [RFC2931], nie miał charakteru statycznego wpisu, ale był generowany w sposób automatyczny w czasie transakcji. Liczba 0 w nazwie SIG(0) odwołuje się do rozmiaru pola danych rekordu objętego podpisem. W zasadzie rekordy SIG(0) można stosować zamiast rekordów TSIG i uzyskiwać ten sam wynik. Jednak ich implementacja nieznacznie się różni. W rozwiązaniu SIG(0) zaufanie wynika z użycia kluczy publicznych zamiast współdzielonych. Ponieważ popularność tego rozwiązania zmniejsza się z upływem czasu na korzyść TSIG, nie będzie ono szczegółowo prezentowane.



Rysunek 18.46. *Odpowiedź na dynamiczną aktualizację DNS podpisana zgodnie ze standardem TSIG. Rekord two.dynzone. został poprawnie usunięty*

18.10.3.3. TKEY

W celu uproszczenia mechanizmów zabezpieczania transakcji DNS (takich jak TSIG lub SIG(0)) opracowano specjalny metarekord TKEY [RFC2930]. Jest on tworzony w sposób dynamiczny i służy do przenoszenia kluczy lub danych potrzebnych do wygenerowania kluczy (np. parametrów DH). Treść rekordu jest przesyłana w dodatkowym polu informacyjnym żądania lub odpowiedzi DNS.

18.10.4. DNSSEC z protokołem DNS64

W rozdziale 11. został opisany protokół DNS64, który przekształca żądania DNS IPv6 w żądania właściwe dla protokołu IPv4 i tworzy rekordy AAAA na podstawie rekordów A wyszukanych w serwerach DNS IPv4. Rozwiązane okazuje się bardzo użyteczne, gdy

trzeba zapewnić jednostkom IPv6 dostęp do serwerów i usług działających w protokole IPv4. Podstawą działania mechanizmu DNS64 jest dynamiczne tworzenie rekordów AAAA. Z kolei funkcjonowanie protokołu DNSSEC bazuje na założeniu, że rekordy zasobów są podpisywane przez uprawnioną do tego celu jednostkę (zazwyczaj przez właściciela domeny lub administratora strefy). Powstaje więc pewien problem. W jaki sposób serwer DNS64 ma przygotować rekord zasobów, skoro nie dysponuje kluczami do wygenerowania sygnatur odpowiadających standardowi DNSSEC? Odpowiedź jest prosta — nie generować podpisów (patrz sekcje 5.5 i 6.2 w dokumencie [RFC6147]).

Połączenie protokołów DNS64 i DNSSEC oznacza konieczność przeniesienia walidacji do komputera (po zaimplementowaniu mechanizmu DNS64) lub do urządzenia DNS64, które jest połączone za pomocą bezpiecznego kanału z resolverem i pracuje jako rekurencyjny serwer nazw. Serwer DNS64 realizujący zadania walidacji jest określany jako serwer vDNS64. Jednostka vDNS64 interpretuje bity CD i DO nadchodzących żądań. Jeśli żaden z nich nie jest ustawiony, syntezuje rekord i wykonuje walidację, ale nie ustawia bitu AD w (potwierdzonej) odpowiedzi. Jeśli ustawiony jest tylko bit CD, serwer syntezuje i waliduje odpowiedź, a następnie zwraca zweryfikowaną odpowiedź z ustawionym bitem AD (klient uznaje wówczas zwracane rekordy zasobów za autentyczne). Należy pamiętać, że serwer DNS64 najpierw żąda dostarczenia rekordów AAAA z serwera IPv4. Syntezowanie rekordów A następuje dopiero wtedy, gdy serwer uzyska wiarygodną informację o braku rekordów AAAA należących do określonego właściciela. Jeśli oba bity (DO i CD) są ustawione, jednostka DNS64 może wykonać walidację, ale nie syntezę. Najczęściej w takich przypadkach zakłada się, że walidacja zostanie przeprowadzona po stronie klienckiej, ale rodzi się pewien problem. Jeśli klient wymaga stosowania mechanizmów zabezpieczeń, ale „nie ma świadomości" wykonania translacji, dostarczone do niego rekordy zasobów i tak będą bezużyteczne w sieci IPv6.

18.11. Identyfikowanie poczty za pomocą kluczy domenowych (DKIM)

Mechanizm **identyfikowania poczty za pomocą kluczy domenowych** (DKIM — *DomainKeys Identified Mail*) [RFC5585] zapewnia powiązanie między określoną jednostką a jej nazwą domenową, które można wykorzystać do ustalenia źródła wiadomości (co jest szczególnie istotne w systemach poczty elektronicznej). Umożliwia uwierzytelnianie osób podpisujących wiadomości, które nie zawsze są nadawcami, co z kolei ułatwia eliminowanie spamu na etapie dystrybucji poczty (tj. pomiędzy jednostkami agentów poczty). Wykonanie zadania polega na dodaniu do listu o standardowym formacie wiadomości internetowej [RFC5322] pola **sygnatury DKIM**. Pole to przechowuje podpis cyfrowy nagłówka i treści wiadomości. Standard DKIM zastępuje wcześniejsze rozwiązanie wykorzystujące pole **sygnatury klucza domeny** (*domain-key signature*).

18.11.1. Sygnatury DKIM

Utworzenie cyfrowego podpisu wiadomości wymaga wygenerowania **identyfikatora domeny podpisującej** (SDID — *Signing Domain Identifier*). Służą do tego algorytmy RSA/SHA-1 lub RSA/SHA-256 oraz odpowiedni klucz prywatny. Identyfikatory SDID

są nazwami domenowymi rejestrowanymi w serwerach DNS, które umożliwiają pobieranie kluczy publicznych przechowywanych w rekordach TXT. Sygnatura DKIM ma formę nagłówka wiadomości w formacie Base64 (podobnie jak certyfikat PEM) i stanowi podpis cyfrowy dla wymienionych wprost pól wiadomości oraz treści tej wiadomości. Po odebraniu wiadomości e-mail agent SMTP odczytuje identyfikator SDID i na jego podstawie wysyła żądanie do serwera DNS, poszukując klucza publicznego potrzebnego do zweryfikowania sygnatury. Rozwiązanie to eliminuje potrzebę wdrażania systemu PKI. Weryfikowana nazwa domenowa jest skonstruowana w taki sposób, aby oprócz samej domeny uwzględniała również **selektor** (selektor klucza publicznego). Przykładowo klucz publiczny opisany za pomocą selektora `klucz35` w domenie `przyklad.com` musiałby zostać zapisany w rekordzie TXT o nazwie właściciela `klucz35._domainkey.przyklad.com`.

Pole sygnatury DKIM [RFC6376] jest dodawane do nagłówka wiadomości i może obejmować kilka pól składowych (pełna ich lista została wymieniona w dokumencie [DKPARAMS]). Działanie mechanizmu DKIM nie odbiega znacząco od systemu SPF w środowisku DNS (więcej informacji na jego temat znajduje się w rozdziale 11.). Jednak zastosowanie kryptograficznych podpisów cyfrowych gwarantuje wyższy poziom zabezpieczenia. Jednak nic nie stoi na przeszkodzie, aby rozwiązania DKIM i SPF były implementowane razem.

Domeny zgodne z mechanizmem DKIM mogą być wykorzystywane w definiowaniu **zasad podpisywania domen** (ADSP — *Author Domain Signing Practices*) [RFC5617]. Rozszerzenie ADSP zakłada utworzenie czytelnego dla komputerów **oświadczenia o zasadach podpisywania** (*signing practices statement*) danych w domenie, które jest zapisywane w systemie DNS za pomocą rekordów TXT o nazwie `_adsp._domainkey.domena`. W czasie pisania książki struktura rekordów była bardzo prosta i obejmowała jedynie informację o tym, w jaki sposób wykorzystywane są sygnatury DKIM. Dopuszczalne wartości to `unknown` (nieznany), `all` (wszystkie) oraz `discardable` (wymuszenie odrzucenia). Każda z wymienionych wartości stanowi podpowiedź dla agenta SMTP, co powinien zrobić z odebraną wiadomością. Ustawienie `unknown` nie wprowadza żadnych zmian w sposobie postępowania. Wartość `all` informuje, że autor podpisuje wszystkie wiadomości, ale wiadomości niepodpisane również należy traktować jak użyteczne. Z kolei wartość `discardable` jest poleceniem odrzucania wszystkich niepodpisanych wiadomości. Jest to zarazem najbardziej rygorystyczne ustawienie.

18.11.2. Przykład

Aby przekonać się o tym, w jaki sposób sygnatury DKIM są zapisywane w wiadomości e-mail, wystarczy wyodrębnić pole *DKIM-Signature* z treści wiadomości generowanych przez „większych" dostawców usług pocztowych (np. Google Gmail):

```
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/relaxed;
        d=gmail.com; s=gamma;
        h=domainkey-signature:mime-version:received:
          sender:received:date
          :x-google-sender-auth:message-id:subject:from:to:content-type;
        bh=PU2XIErWsXvhvt1W96ntPWZ2VImjVZ3vBY2T/A+wA3A=;
        b=WneQe6kpeu/BfMfa2RSlAllTvYKfIKmoQRXNc
          IQJDIVoE38+fGDajOuhNm8vXp/8kJ
          I8HqtkV4/P6/QVPMN+/5bS5dsnlhzOS/YoP
          bZx0Lt2bD67G4HPsvm6eLsaIC9rQECUSL
          MdaTBK3BgFhYo3nenq3+8GxTe9I+zBcqWAVPU=
```

Z powyższego listingu wynika, że sygnatura została wygenerowana zgodnie z wersją 1. protokołu — za utworzenie skrótu odpowiada algorytm SHA-256, a do jego podpisania użyto mechanizmu RSA. **Algorytmy normalizacji** (*canonicalization algorithms*) nagłówka i treści dopuszczają pewną swobodę w definiowaniu treści, o czym świadczy wartość `relaxed` w polu `c`. Algorytmy normalizacji umożliwiają przekształcenie wiadomości do ujednoliconego formatu. Obowiązujące obecnie ustawienia to: `simple` (wartość domyślna), oznaczająca, że tekst musi zostać zachowany w niezmienionej postaci, oraz `relaxed` umożliwiające przekształcanie tekstu wejściowego (np. zmianę znaków odstępu lub zwijanie wierszy nagłówka). Selektor (`s`) ma wartość `gamma`, a domena (`d`) odpowiada nazwie `gmail.com`. Drugi z elementów posłuży do pobrania odpowiedniego klucza publicznego. Pola uwzględnione w podpisie cyfrowym (`h`) to `domainkey-signature` (poprzednik standardu DKIM), wersja typu MIME, data nadania, `x-google-sender-auth`, `message-id`, `subject`, `from` oraz `content-type`. Pole składowe `bh` przechowuje wartość skrótu treści wiadomości zapisanej w kodowaniu Base64. Natomiast wartość pola `b` jest podpisem RSA skrótu nagłówków wymienionych w podpolu `h`.

Aby pobrać klucz publiczny umożliwiający weryfikację podpisu, wystarczy wykonać następujące zapytanie:

```
Linux% dig gamma._domainkey.gmail.com. txt +nostats +noquestion
; <<>> DiG 9.7.2-P3 <<>> gamma._domainkey.gmail.com. txt
+nostats +noquestion
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17372
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; ANSWER SECTION:
gamma._domainkey.gmail.com. 296       IN   TXT  "k=rsa\; t=y\; p=MIGfMA0GCS
qGSIb3DQEBAQUAA4GNADCBiQKBgQDIhyR3oItOy22ZOaBrIVe9m/iME3RqOJeasANSpg2YTHTYV
+Xtp4xwf5gTjCmHQEMOs0qYu0FYiNQPQogJ2t0Mfx9zNu06rfRBDjiIU9tpx2T+NGlWZ8qhbiLo
5By8apJavLyqTLavyPSrvsx083YzC63T4Age2CDqZYA+OwSMWQIDAQAB"
```

Na podstawie uzyskanego wyniku można stwierdzić, że poszukiwany klucz jest kluczem publicznym algorytmu RSA. Wpis `t=y` świadczy o tym, że administratorzy domeny testują rozwiązanie DKIM. Wyniki walidacji nie powinny więc bezpośrednio wpływać na dalsze przetwarzanie wiadomości. Aby uzyskać informacje o zabezpieczeniu ADSP, należy wykonać poniższe polecenie.

```
Linux% host -t txt _adsp._domainkey.paypal.com.
_adsp._domainkey.paypal.com descriptive text "dkim=discardable"
```

Wynik wykonania instrukcji dostarcza informacji o tym, że firma Paypal wykorzystuje najsilniejszą technikę podpisywania DKIM, zgodnie z którą wiadomości niespełniające założeń walidacji DKIM powinny zostać odrzucone. Definiowanie reguł ADSP jest wciąż bardzo rzadko stosowane, ponieważ różne systemy poczty elektronicznej w odmienny sposób przepisują wiadomości.

18.12. Ataki na protokoły zabezpieczeń

Ataki na protokoły zabezpieczeń różnią się nieznacznie od ataków na inne protokoły (opisywanych w pozostałych rozdziałach). Zagrożenia omawiane wcześniej wynikają z prób zakłócenia pracy protokołów, które podczas projektowania nie zostały wyposażone w stosowne mechanizmy zabezpieczające. Osoby atakujące starają się więc wykorzystać pewne niedoskonałości w budowie lub implementacji tych mechanizmów. Ataki na protokoły zabezpieczeń wymagają dodatkowo znajomości kryptografii, a tym samym matematycznych podstaw rozwiązań kryptograficznych. Działania tego typu mogą się zakończyć sukcesem, jeśli do zabezpieczenia komunikacji użyto kiepskich algorytmów, prostych lub zbyt krótkich kluczy lub niewłaściwie dobranych komponentów, które, współpracując, sprawiają, że system jest mniej odporny niż byłby w innym przypadku (jeden z klasycznych i niezwykle fascynujących przypadków tego typu opisano w analizie kryptograficznej systemu VENONA [VENONA]).

Rozpocznijmy przegląd ataków na protokoły zabezpieczeń od najniższej warstwy. Wiele z działań przestępczych jest wymierzonych w systemy 802.11 oraz EAP. Zabezpieczenia wdrażane w początkowym okresie stosowania rozwiązań z grupy 802.11 (np. WEP lub WPA–TKIP) okazały się łatwe do obejścia ze względu na słabości kryptograficzne [TWP07], [OM09]. Wprowadzony później mechanizm WPA2-AES jest znacznie odporniejszy na włamania, choć niewłaściwy dobór współdzielonych kluczy (PSK — *Pre-Shared Key*) może istotnie zwiększyć podatność systemu na ataki słownikowe.

Standard EAP nie definiuje, co prawda, własnych metod uwierzytelniania, ale może stać się podatny na ataki z uwagi na luki w metodach uwierzytelniania, od których jest zależny. Systemy EAP wykorzystujące klucze generowane na podstawie haseł użytkowników (np. EAP-GSS, EAP-LEAP, czy EAP-SIM) są często narażone na ataki słownikowe. Z kolei mechanizm 802.1X/EAP jest podatny na ataki MITM w przypadku wykorzystania tunelowanych protokołów uwierzytelniania (zagadnienie to zostało opisane w dokumencie [ANN02]). Problem wiąże się z operacją wyznaczania klucza sesji w chwili, w której tylko jedna strona połączenia została uwierzytelniona. Jeśli np. serwer uwierzytelnia się w jednostce klienckiej, a realizowana wymiana komunikatów jest podstawą do sformowania tunelu (zabezpieczonego kluczem sesji), w którym inny protokół wykonuje uwierzytelnienie w przeciwnym kierunku, możliwe staje się przeprowadzenie ataku MITM z podstawieniem jednostki atakującej jako klienta uprawnionego do ustanowienia połączenia.

Również mechanizm IPsec stał się celem wielu udokumentowanych ataków sieciowych. Pewna ich grupa bazuje na wykorzystywaniu słabości połączeń szyfrowanych, ale nieobjętych ochroną integralności [PY06]. Opcja konfiguracyjna umożliwiająca taki tryb pracy została zdefiniowana w dokumentacji IPsec, choć odradzane jest jej użycie. Ogólnie rzecz ujmując, problemem jest możliwość zmodyfikowania szyfrogramu w taki sposób (za pomocą techniki **zamiany bitów** [*bit flipping attack*]), żeby datagramy uzyskane po rozszyfrowaniu były przekłamane w przewidywalny sposób. Przykładowo datagram ESP transmitowany w trybie tunelowym po odpowiedniej zamianie bitów może zostać rozszyfrowany jako pakiet ze sztucznie zwiększoną wartością pola **długości nagłówka** (IHL). To z kolei powoduje przetworzenie pola danych jako opcji IP, a w konsekwencji wygenerowanie komunikatu ICMP, który może się okazać użyteczny dla atakującego.

Na poziomie warstwy transportowej rozwiązaniem szczególnie podatnym na ataki okazał się protokół SSL 2.0. Jeden z takich ataków — **odwrotne uzgadnianie szyfrów** (*cipher suite rollback*) — bazuje na technice MITM, która umożliwia przekonanie każdej ze stron komunikacji, że jednostka zdalna akceptuje jedynie słabe algorytmy szyfrowania. W rezultacie stacje końcowe wybierają niedostatecznie bezpieczny zestaw algorytmów kryptograficznych, co może zostać wykorzystane przez osobę atakującą. Nieco bardziej wyrafinowany atak na protokoły SSL/TLS wykorzystuje określoną kolejność działań strony odbiorczej: deszyfrowanie, usunięcie dopełnienia i sprawdzenie wartości MAC. W przypadku nieodpowiedniej długości dopełnienia lub wartości MAC stacja odbiorcza generuje komunikat o błędzie mechanizmu SSL. Analizując czas generowania wspomnianych komunikatów, można generować dopełnienia [CHVV03], które pozwolą na uzyskanie tekstu jawnego z połączenia w OpenSSH. Mechanizm ten pozwala na ustalenie, czy tekst jawny wykorzystany do utworzenia szyfrogramu ma odpowiednią ilość dopełnienia. Zgodnie z zamieszczonymi wcześniej informacjami najnowsze ataki (na protokół TLS 1.2) są formą ataku MITM, w której do relacji TLS wstrzykiwany jest prefiks o dowolnej długości, co powoduje renegocjację połączenia (ale z podtrzymaniem) po odebraniu komunikatu od uprawnionego użytkownika [RD09]. Rozwiązanie to wiąże się z powiązaniem wcześniejszych parametrów kanału z parametrami następnego kanału komunikacyjnego za pomocą rozszerzenia TLS. Problem bezpieczeństwa i wiązania kanałów został szczegółowo opisany w dokumencie [RFC5056].

Zabezpieczenia protokołu DNS były opracowywane przez bardzo długi czas. Potrzeba ich wdrożenia stała się oczywista po przeprowadzeniu przez Kamińskiego ataku z zatruciem pamięci podręcznej serwera, opisanego w rozdziale 11. Jednym z pierwszych problemów był również atak z **enumeracją strefy**, który stał się możliwy po wprowadzeniu rekordów `NSEC` i wyeliminowany za pomocą rekordów NSEC3 [BM09]. Pod koniec 2009 roku w przemówieniu otwierającym warsztaty poświęcone systemowi DNSSEC Dan Bernstein wymienił kilka słabości rozwiązania [B09]: mechanizm ten może posłużyć jako element wzmacniający ataki DoS, wyciek danych strefy jest możliwy nawet po zastosowaniu rekordów `NSEC3`, implementacja systemu zawiera luki, nie można unieważniać sygnatur, stosowane mechanizmy kryptograficzne mogą być przedmiotem kryptoanalizy, niektóre rekordy `NS` i `A` są podatne na ataki. Na krótko przed zakończeniem prac nad książką strefa główna została podpisana za pomocą mechanizmów DNSSEC. Rozwiązanie to zostało również wdrożone przez kilka organizacji. Można więc liczyć na to, że w kolejnych latach będą opracowywane poprawki i udoskonalenia mechanizmu.

18.13. Podsumowanie

Zagadnienie bezpieczeństwa komunikacji jest szerokie i niezwykle interesujące, a przedstawione tutaj rozwiązania stanowią ledwie wzmiankę na ten temat. Myśląc o bezpiecznej wymianie danych, spodziewamy się zazwyczaj zachowania poufności, autentyczności, integralności i niezaprzeczalności informacji. Najważniejszym narzędziem w dążeniu do zapewnienia wymienionych właściwości jest kryptografia, a dokładniej jej algorytmy i klucze. Dwie najistotniejsze formy działalności w tym obszarze to stosowanie symetrycznych algorytmów kryptograficznych, które nie obniżają wydajności systemu, ale wymagają przechowywania tajnych kluczy, oraz wdrażanie kryptografii symetrycznej, w której każda strona komunikacji dysponuje parą kluczy, w tym jednym znanym publicznie. Mechanizmy kryptograficzne bazujące na kluczu publicznym zapewniają zarówno

uwierzytelnienie, jak i poufność informacji, i mogą współdziałać z rozwiązaniami symetrycznymi, które gwarantują większą wydajność przetwarzania. Do innych algorytmów matematycznych ściśle związanych z kryptografią zaliczamy mechanizm Diffiego-Hellmana służący do uzgadniania kluczy szyfrowania symetrycznego, funkcje pseudolosowe odpowiedzialne za wyznaczanie przypadkowych wartości w procesie generowania kluczy oraz funkcje MAC do sprawdzania integralności danych. Protokoły uwzględniające losowe wartości jednorazowe są odporniejsze na ataki polegające na ponawianiu komunikatów, ponieważ wymagają zapisania w każdym żądaniu i każdej odpowiedzi wspólnej nowo wyznaczonej wartości. Walka z atakami słownikowymi polega natomiast na wprowadzaniu wartości zaburzającej, która zmienia dane wejściowe lub sposób działania algorytmu.

Polegając na rozwiązaniach wykorzystujących klucze publiczne, chcemy mieć pewność, że klucze te zostały podpisane i uwierzytelnione przez jednostkę lub grupę jednostek, którym ufamy. Najczęściej do tego celu służy infrastruktura klucza publicznego (PKI) z jej urzędami certyfikacji, choć to samo zadanie można zrealizować, bazując na sieci zaufania. Klucze publiczne oraz certyfikaty (a także inne dane PKI) są najczęściej zapisywane w formacie ITU-T X.509. Certyfikaty są podpisywane zgodnie z hierarchiczną zależnością, która na najwyższym poziomie wyznacza kotwicę zaufania. Uznanie któregokolwiek z certyfikatów za zaufany wymaga sprawdzenia, czy łańcuch zaufania nie został przerwany oraz że żaden z certyfikatów stanowiących elementy łańcucha nie został unieważniony. Weryfikacja statusu certyfikatu sprowadza się do odczytania zawartości listy unieważnień (CRL) lub wysłania zapytania w protokole OCSP. Proces walidacji można również zlecić innej jednostce, wykorzystując powołany do tego celu protokół SCVP.

Certyfikaty i klucze są przechowywane w plikach o różnych formatach. Formaty DER i CER są stosowane w kodowaniu binarnym zgodnym ze standardem ASN.1. Pliki PEM zawierają treść pliku DER, ale zakodowaną w formacie ASCII, dzięki czemu można je łatwo analizować i edytować. Format PKCS#12 (następca formatu PFX opracowanego przez firmę Microsoft) pozwala na przechowywanie zarówno certyfikatów, jak i kluczy prywatnych. Zabezpieczenie treści klucza prywatnego wymaga zaszyfrowania treści pliku. Jednak narzędzia, takie jak openssl, umożliwiają przekształcenie jednego formatu w inny.

W każdej warstwie stosu protokołów funkcjonują pewne protokoły zabezpieczeń. Część z nich jest nawet implementowana pomiędzy warstwami. Od warstwy łącza danych w górę dostępne są technologie bazujące na własnych protokołach szyfrowania i uwierzytelniania, które nie wchodzą w skład klasycznego stosu TCP/IP. Przykładem jest protokół EAP, który odpowiada za uwierzytelnienie stron komunikacji i wykorzystuje do tego certyfikaty komputerów, certyfikaty użytkowników, karty inteligentne, hasła itp. Mechanizmy EAP są używane przede wszystkim w sieciach korporacyjnych, w których działają serwery uwierzytelnienia i autoryzacji (serwery AAA). Protokół EAP znajduje zastosowanie również podczas uwierzytelniania stron komunikacji IPsec.

Platforma IPsec jest kolekcją protokołów zapewniających bezpieczeństwo na poziomie 3. warstwy modelu OSI. W jej skład wchodzą protokoły IKE, AH i ESP. Zadanie mechanizmu IKE polega na ustanawianiu i obsłudze relacji zabezpieczeń, czyli powiązania między dwiema stronami komunikacji. Sama komunikacja może wymagać uwierzytelniania (w protokole AH) lub szyfrowania (w protokole ESP) w trybie transportowym albo tunelowym. Praca w trybie transportowym oznacza, że uwierzytelnienie lub szyfrowanie

powodują zmianę treści nagłówka IP. Z kolei w trybie tunelowym datagram IP jest umieszczany w nowym datagramie IP. Protokół ESP jest częściej stosowanym rozwiązaniem. Wszystkie protokoły IPsec umożliwiają wybieranie algorytmów i parametrów (zestawów kryptograficznych) w takich operacjach jak szyfrowanie, kontrola integralności, uzgadnianie kluczy DH oraz uwierzytelnianie.

W kolejnej warstwie stosu protokołów, w warstwie transportowej, funkcjonują mechanizmy (najnowszy to TLS 1.2) odpowiedzialne za zabezpieczenie procesu przekazywania informacji między aplikacjami. Mechanizm TLS 1.2 sam jest implementowany w formie stosu, w którym działa protokół warstwy rekordów oraz trzy protokoły uzgadniania połączenia: Cipher Change, Alert i Handshake. Dane aplikacji są przenoszone przez protokół Record, który zapewnia szyfrowanie i ochronę integralności danych z wykorzystaniem parametrów wynegocjowanych w ramach protokołu Handshake. Protokół Cipher Change służy jednie do zmiany stanu mechanizmu z bieżącego na nowy. Połączenie rozwiązań TCP/IP i TLS jest podstawą zabezpieczenia (szyfrowania) połączeń z serwerami WWW (HTTPS). Jednak pewna odmiana mechanizmu TLS o nazwie DTLS znajduje zastosowanie w ochronie datagramów UDP i DCCP.

Do lepszego zabezpieczenia systemu nazw domenowych, a tym samym komunikacji WWW, służy protokół DNSSEC. W dniu 15 lipca 2010 roku główna strefa DNS została podpisana zgodnie z założeniami DNSSEC, co pozwala sądzić, że z czasem rozwiązanie to będzie wdrażane na szeroką skalę. Standard DNSSEC wprowadza do użycia kilka nowych rekordów zasobów: `DNSKEY`, `DS`, `NSEC/NSEC3/NSEC3PARAM` oraz `RRSIG`. Dwa pierwsze przechowują i wyznaczają miejsce składowania kluczy publicznych używanych w procesie podpisywania struktury i zawartości stref. Rekordy `NSEC` oraz `NSEC3/NSEC3PARAM` ułatwiają budowanie wykazów nazw w porządku kanonicznym oraz zestawiają typy rekordów odpowiadających nazwie domenowej. Dzięki nim można w niezawodny sposób ustalić nieistnienie nazwy domenowej oraz występowanie rekordów określonego typu w danej strefie. Rekordy `RRSIG` przechowują sygnatury innych rekordów. Podpisanie strefy wymaga wygenerowania rekordów `RRSIG` w odniesieniu do wszystkich autorytatywnych rekordów zdefiniowanych w ramach tej strefy. O bezpieczeństwo zapytań DNS dbają resolvery walidujące oraz serwery nazw, które opierają swoje działanie na dostępności kotwic zaufania. Ich zadanie polega na sprawdzeniu, czy podpisy cyfrowe rekordów odpowiadają kluczom publicznym dostarczanym przez system DNS. Powoduje to generowanie błędów w przypadku stwierdzenia niezgodności, a co się z tym wiąże, wyeliminowanie przypadków przejmowania nazw domenowych, w których jednostki atakujące podają się za stacje uprawnione do świadczenia usług DNS. W niektórych środowiskach wymagane jest dodatkowo zabezpieczenie transakcji DNS. Pomocne okazują się wówczas protokoły TSIG i SIG(0), które gwarantują uwierzytelnienie kanału komunikacyjnego. Wymienione rozwiązania służą przede wszystkim do zabezpieczania dynamicznych aktualizacji stref DNS oraz transferów stref.

Ataki na protokoły zabezpieczeń nie sprowadzają się jedynie do wykorzystywania luk implementacyjnych i niedoskonałości projektowych. Obejmują również próby znalezienia matematycznych rozwiązań pozwalających na pozyskanie utajnionych informacji (np. bitów klucza). Z biegiem czasu stało się jasne, że kluczem do bezpiecznego przesyłania danych jest elastyczność mechanizmów kryptograficznych. Większość z omawianych protokołów uwzględnia zestawy algorytmów kryptograficznych, które mogą ewoluować wraz ze zwiększaniem mocy obliczeniowych systemów informatycznych i zapewniać

wyższy poziom bezpieczeństwa. Wiele protokołów uznanych za bezpieczne (nawet te, które zostały drobiazgowo przebadane przez ekspertów) zostało złamanych po serii testów służących wyszukaniu możliwych do wykorzystania luk. Dotyczy to przede wszystkim rozwiązań, w których można zastosować ataki MITM oraz inne formy ataków aktywnych. Zapewnienie bezpieczeństwa komunikacji wymaga niezwykłej staranności podczas projektowania nowych rozwiązań i utrzymywania istniejących mechanizmów zabezpieczających.

18.14. Bibliografia

[802.1X-2010] „IEEE Standard for Port-Based Network Access Control", IEEE Std 802.1X-2010, luty 2010.

[AKNT04] Y. Amir, Y. Kim, C. Nita-Rotaru, G. Tsudik, *On the Performance of Group Key Agreement Protocols*, „ACM Transactions on Information and System Security", 7(3), sierpień. 2004.

[ANN02] N. Asokan, V. Niemi, K. Nyberg, „Man-in-the-Middle in Tunneled Authentication Protocols (Extended Abstract)", *Proc. 11th Security Protocols Workshop/LNCS 3364* (Springer, 2003).

[B06] M. Bellare, *New Proofs for NMAC and HMAC: Security without Collision-Resistance* (wersja wstępna z „CRYPTO 06"), czerwiec 2006.

[B09] D. Bernstein, *Breaking DNSSEC*, wykład otwierający na Workshop on Offensive Technologies (WOOT), sierpień 2009.

[BCK96] M. Bellare, R. Canetti, H. Krawczyk, *Keying Hash Functions for Message Authentication* (wersja skrócona z „CRYPTO 96/LNCS 1109"), czerwiec 1996.

[BM09] J. Bau and J. Mitchell, *A Security Evaluation of DNSSEC with NSEC3*, Network and Distributed System Security Symposium (NDSS), luty – marzec 2010.

[BOPSW09] R. Biddle i in., *Browser Interfaces and Extended Validation SSL Certificates: An Empirical Study*, sprawozdanie z konferencji ACM Cloud Security Workshop, listopad 2009.

[CABF09] CA/Browser Forum, *Guidelines for the Issuance and Management of Extended Validation Certificates (v1.2)*, 2009, *http://www.cabforum.org/Guidelines_v1_2.pdf*

[CHP] National Institute of Standards and Technology, Cryptographic Hash Project, Computer Security Division — Computer Security Resource Center, *http://csrc.nist.gov/groups/ST/hash*

[CHVV03] B. Canvel, A. Hiltgen, S. Vaudenay, M. Vuagnoux, *Password Interception in a SSL/TLS Channel*, „CRYPTO 2003/LNCS 2729".

[DH76] W. Diffie, M. Hellman, *New Directions in Cryptography*, „IEEE Transactions on Information Theory", IT-22, listopad 1976.

[DKPARAMS] *http://www.iana.org/assignments/dkim-parameters*

[DNSSECALG] *http://www.iana.org/assignments/dns-sec-alg-numbers*

[DOW92] W. Diffie, P. Oorschot, M. Wiener, *Authentication and uthenticated Key Exchanges*, „Designs, Codes and Cryptography", 2, czerwiec 1992.

[DSRRTYPES] *http://www.iana.org/assignments/ds-rr-types*

[FIPS186-3] National Institute for Standards and Technology, *Digital Signature Standard (DSS)*, FIPS PUB 186-3, czerwiec 2009.

[FIPS197] National Institute for Standards and Technology, *Advanced Encryption Standard (AES)*, FIPS PUB 197, listopad 2001.

[FIPS198] National Institute for Standards and Technology, *The Keyed-Hash Message Authentication Code (HMAC)*, FIPS PUB 198, marzec 2002.

[FIPS800-38B] National Institute for Standards and Technology, *Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication*, NIST Special Publication 800-38B, maj 2005.

[GGM86] O. Goldreich, S. Goldwasser, S. Micali, *How to Construct Random Functions*, „Journal of the ACM", 33(4), październik 1986.

[IDDCIN] S. Weiler , D. Blacka, *Clarifications and Implementation Notes for DNSSECbis*, Internet draft-ietf-dnsext-dnssec-bis-updates, lipiec 2011.

[IDDS2] B. Dickson, *DNSSEC Delegation Signature with Canonical Signer Name*, Internet draft-dickson-dnsext-ds2 (artykuł przedawniony), listopad 2010.

[IDDTLS] E. Rescorla, N. Modadugu, *Datagram Transport Layer Security Version 1.2*, Internet draft-ietf-tls-rfc4347-bis, lipiec 2011.

[IEAP] *http://www.iana.org/assignments/eap-numbers*

[IK03] T. Iwata and K. Kurosawa, *OMAC: One-Key CBC MAC*, obradyFast Software Encryption, marzec 2003.

[IKEPARAMS] *http://www.iana.org/assignments/ikev2-parameters*

[IPANA] *http://www.iana.org/assignments/pana-parameters*

[ITUOID] *http://www.itu.int/ITU-T/asn1*

[IWESP] *http://www.iana.org/assignments/wesp-flags*

[K87] N. Koblitz, *Elliptic Curve Cryptosystems*, „Mathematics of Computation", 48, 1987.

[L01] C. Landwehr, „Computer Security", Springer-Verlag Online, lipiec 2001.

[M85] V. Miller, *Uses of Elliptic Curves in Cryptography, Advances in Cryptology: CRYPTO '85*, „Lecture Notes in Computer Science", tom 218 (Springer-Verlag, 1986).

[MSK09] S. McClure, J. Scambray, G. Kurtz, *Hacking Exposed, Sixth Edition* (McGraw-Hill, 2009).

[MW99] U. Maurer i S. Wolf, *The Relationship between Breaking the Diffie-Hellman Protocol and Computing Discrete Logarithms*, „Siam Journal on Computing", 28(5), 1999.

[NAZ00] Network Associates P. Zimmermann, *Introduction to Cryptography*, fragment dokumentacji PGP 7.0, *http://www.pgpi.org/doc/guide/7.0/en*

[NIST800-38B] National Institute for Standards and Technology, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*, specjalna publikacja NIST, listopad 2005.

[NSEC3PARAMS] *http://www.iana.org/assignments/dnssec-nsec3-parameters*

[OM09] T. Ohigashi, M. Morii, *A Practical Message Falsification Attack on WPA*, Joint Workshop on Information Security, sierpień 2009.

[PY06] K. Paterson, A. Yau, *Cryptography in Theory and Practice: The Case of Encryption in IPsec*, EUROCRYPT 2006/LNCS 4004.

[RD09] M. Ray, S. Dispensa, *Renegotiating TLS*, „PhoneFactor Technical Report", listopad 2009.

[RFC1321] R. Rivest, *The MD5 Message-Digest Algorithm*, Internet RFC 1321 (informational), kwiecień 1992.

[RFC2104] H. Krawczyk, M. Bellare, R. Canetti, *HMAC: Keyed-Hashing for Message Authentication*, Internet RFC 2104 (informational), luty 1997.

[RFC2403] C. Madson, R. Glenn, *The Use of HMAC-MD5-96 within ESP and AH*, Internet RFC 2403, listopad 1998.

[RFC2404] C. Madson, R. Glenn, *The Use of HMAC-SHA-1-96 within ESP and AH*, Internet RFC 2404, listopad 1998.

[RFC2409] D. Harkins, D. Carrel, *The Internet Key Exchange (IKE)*, Internet RFC 2409 (obsolete), listopad 1998.

[RFC2410] R. Glenn, S. Kent, *The NULL Encryption Algorithm and Its Use with IPsec*, Internet RFC 2410, listopad 1998.

[RFC2451] R. Pereira, R. Adams, *The ESP CBC-Mode Cipher Algorithms*, Internet RFC 2451, listopad 1998.

[RFC2560] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol — OCSP*, Internet RFC 2560, czerwiec 1999.

[RFC2631] E. Rescorla, *Diffie-Hellman Key Agreement Method*, Internet RFC 2631, czerwiec 1999.

[RFC2671] P. Vixie, *Extension Mechanisms for DNS (EDNS0)*, Internet RFC 2671, sierpień 1999.

[RFC2845] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, B. Wellington, *Secret Key Transaction Authentication for DNS (TSIG)*, Internet RFC 2845, maj 2000.

[RFC2865] C. Rigney, S. Willens, A. Rubens, W. Simpson, *Remote Authentication Dial In User Service (RADIUS)*, Internet RFC 2865, czerwiec 2000.

[RFC2930] D. Eastlake 3rd, *Secret Key Establishment for DNS (TKEY RR)*, Internet RFC 2930, wrzesień 2000.

[RFC2931] D. Eastlake 3rd, *DNS Request and Transaction Signatures (SIG(0)s)*, Internet RFC 2931, wrzesień 2000.

[RFC3162] B. Aboba, G. Zorn, D. Mitton, *RADIUS and IPv6*, Internet RFC 3162, sierpień 2001.

[RFC3173] A. Shacham, B. Monsour, R. Pereira, M. Thomas, *IP Payload Compression Protocol (IPComp)*, Internet RFC 3173, wrzesień 2001.

[RFC3193] B. Patel, B. Aboba, W. Dixon, G. Zorn, S. Booth, *Securing L2TP Using IPsec*, Internet RFC 3193, listopad 2001.

[RFC3225] D. Conrad, *Indicating Resolver Support of DNSSEC*, Internet RFC 3225, grudzień 2001.

[RFC3447] J. Jonsson, B. Kaliski, *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*, Internet RFC 3447 (informational), luty 2003.

[RFC3526] T. Kivinen, M. Kojo, *More Modular Exponential (MODP) Diffie-Hellman Groups for Internet Key Exchange (IKE)*, Internet RFC 3526, maj 2003.

[RFC3547] M. Baugher, B. Weis, T. Hardjono, H. Harney, *The Group Domain of Interpretation*, Internet RFC 3547, lipiec 2003.

[RFC3566] S. Frankel, H. Herbert, *The AES-XCBC-MAC-96 Algorithm and Its Use with IPsec*, Internet RFC 3566, wrzesień 2003.

[RFC3588] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko, *Diameter Base Protocol*, Internet RFC 3588, wrzesień 2003.

[RFC3602] S. Frankel, R. Glenn, S. Kelly, *The AES-CBC Cipher Algorithm and Its Use with IPsec*, Internet RFC 3602, wrzesień 2003.

[RFC3645] S. Kwan, P. Garg, J. Gilroy, L. Esibov, J. Westhead, R. Hall, *Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)*, Internet RFC 3645, październik 2003.

[RFC3686] R. Housley, *Using Advanced Encryption Standard (AES) Counter Mode with IPsec Encapsulating Security Payload (ESP)*, Internet RFC 3686, styczeń 2004.

[RFC3713] M. Matsui, J. Nakajima, S. Moriai, *A Description of the Camellia Encryption Algorithm*, Internet RFC 3713 (informational), kwiecień 2004.

[RFC3715] B. Aboba, W. Dixon, *IPsec-Network Address Translation (NAT) Compatibility Requirements*, Internet RFC 3715 (informational), marzec 2001.

[RFC3740] T. Hardjono, B. Weis, *The Multicast Group Security Architecture*, Internet RFC 3740 (informational), marzec 2004.

[RFC3748] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, H. Levkowetz, ed., *Extensible Authentication Protocol (EAP)*, czerwiec 2004.

[RFC3749] S. Hollenbeck, *Transport Layer Security Protocol Compression Methods*, Internet RFC 3749, maj 2004.

[RFC3947] T. Kivinen, B. Swander, A. Huttunen, V. Volpe, *Negotiation of NAT-Traversal in the IKE*, Internet RFC 3947, styczeń 2005.

[RFC3948] A. Huttunen, B. Swander, V. Volpe, L. DiBurro, M. Stenberg, *UDP Encapsulation of IPsec ESP Packets*, Internet RFC 3948, styczeń 2005.

[RFC4016] M. Parthasarathy, *Protocol for Carrying Authentication and Network Access (PANA) Threat Analysis and Security Requirements*, Internet RFC 4016 (informational), marzec 2005.

[RFC4033] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, *DNS Security Introduction and Requirements*, Internet RFC 4033, marzec 2005.

[RFC4034] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, *Resource Records for the DNS Security Extensions*, Internet RFC 4034, marzec 2005.

[RFC4035] R. Arends, R. Austein, M. Larson, D. Massey, S. Rose, *Protocol Modifications for the DNS Security Extensions*, Internet RFC 4035, marzec 2005.

[RFC4058] A. Yegin, ed., Y. Ohba, R. Penno, G. Tsirtsis, C. Wang, *Protocol for Carrying Authentication for Network Access (PANA) Requirements*, Internet RFC 4058 (informational), maj 2005.

[RFC4086] D. Eastlake 3rd, J. Schiller, S. Crocker, *Randomness Requirements for Security*, Internet RFC 4086/BCP 0106, czerwiec 2005.

[RFC4120] C. Neuman, T. Yu, S. Hartman, K. Raeburn, *The Kerberos Network Authentication Service (V5)*, Internet RFC 4120, lipiec 2005.

[RFC4251] T. Ylonen, C. Lonvick, red., *The Secure Shell (SSH) Protocol Architecture*, Internet RFC 4251, styczeń 2006.

[RFC4301] S. Kent, K. Seo, *Security Architecture for the Internet Protocol*, Internet RFC 4301, grudzień 2005.

[RFC4302] S. Kent, *IP Authentication Header*, Internet RFC 4302, grudzień 2005.

[RFC4303] S. Kent, *IP Encapsulating Security Payload (ESP)*, Internet RFC 4303, grudzień 2005.

[RFC4307] J. Schiller, *Cryptographic Algorithms for Use in the Internet Key Exchange Version 2 (IKEv2)*, Internet RFC 4307, grudzień 2005.

[RFC4309] R. Housley, *Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)*, Internet RFC 4309, grudzień 2005.

[RFC4346] T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.1*, Internet RFC 4346 (przedawnione), kwiecień 2006.

[RFC4347] E. Rescorla, N. Modadugu, *Datagram Transport Layer Security*, Internet RFC 4347, kwiecień 2006.

[RFC4398] S. Josefsson, *Storing Certificates in the Domain Name System (DNS)*, Internet RFC 4398, marzec 2006.

[RFC4431] M. Andrews, S. Weiler, *The DNSSEC Lookaside Validation (DLV) DNS Resource Record*, Internet RFC 4431 (informational), luty 2006.

[RFC4434] P. Hoffman, *The AES-XCBC-PRF-128 Algorithm for the Internet Key Exchange Protocol (IKE)*, Internet RFC 4434, luty 2006.

[RFC4492] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, B. Moeller, *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*, Internet RFC 4492 (informational), maj 2006.

[RFC4493] JH. Song, R. Poovendran, J. Lee, T. Iwata, *The AES-CMAC Algorithm*, Internet RFC 4493 (informational), czerwiec 2006.

[RFC4509] W. Hardaker, *Use of SHA-256 in DNSSEC Delegation Signer (DS) Resource Records (RRs)*, Internet RFC 4509, maj 2006.

[RFC4535] H. Harney, U. Meth, A. Colegrove, G. Gross, *GSAKMP: Group Secure Association Key Management Protocol*, Internet RFC 4535, czerwiec 2006.

[RFC4555] P. Eronen, *IKEv2 Mobility and Multihoming Protocol (MOBIKE)*, Internet RFC 4555, czerwiec 2006.

[RFC4615] J. Song, R. Poovendran, J. Lee, T. Iwata, *The Advanced Encryption Standard-Cipher-Based Message Authentication Code-Pseudo-Random Function-128 (AES-CMAC-PRF-128) Algorithm for the Internet Key Exchange Protocol (IKE)*, Internet RFC 4615, sierpień 2006.

[RFC4635] D. Eastlake 3rd, *HMAC SHA (Hashed Message Authentication Code, Secure Hash Algorithm) TSIG Algorithm Identifiers*, Internet RFC 4635, sierpień 2006.

[RFC4681] S. Santesson, A. Medvinsky, J. Ball, *TLS User Mapping Extension*, Internet RFC 4681, październik 2006.

[RFC4739] P. Eronen, J. Korhonen, *Multiple Authentication Exchanges in the Internet Key Exchange (IKEv2) Protocol*, Internet RFC 4739 (experimental), listopad 2006.

[RFC4754] D. Fu, J. Solinas, *IKE and IKEv2 Authentication Using the Elliptic Curve Digital Signature Algorithm (ECDSA)*, Internet RFC 4754, styczeń 2007.

[RFC4835] V. Manral, *Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH)*, Internet RFC 4835, kwiecień 2007.

[RFC4851] N. Cam-Winget, D. McGrew, J. Salowey, H. Zhou, *The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)*, Internet RFC 4851 (informational), maj 2007.

[RFC4877] V. Devarapalli, F. Dupont, *Mobile IPv6 Operation with IKEv2 and the Revised IPsec Architecture*, Internet RFC 4877, kwiecień 2007.

[RFC4880] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, R. Thayer, *Open-PGP Message Format*, Internet RFC 4880, listopad 2007.

[RFC5011] M. StJohns, *Automated Updates of DNS Security (DNSSEC) Trust Anchors*, Internet RFC 5011, wrzesień 2007.

[RFC5054] D. Taylor, T. Wu, N. Mavrogiannopoulos, T. Perrin, *Using the Secure Remote Password (SRP) Protocol for TLS Authentication*, Internet RFC 5054 (informational), listopad 2007.

[RFC5055] T. Freeman, R. Housley, A. Malpani, D. Cooper, W. Polk, *Server-Based Certificate Validation Protocol (SCVP)*, Internet RFC 5055, grudzień 2007.

[RFC5056] N. Williams, *On the Use of Channel Bindings to Secure Channels*, Internet RFC 5056, listopad 2007.

[RFC5077] J. Salowey, H. Zhou, P. Eronen, H. Tschofenig, *Transport Layer Security (TLS) Session Resumption without Server-Side State*, Internet RFC 5077, styczeń 2008.

[RFC5106] H. Tschofenig, D. Kroeselberg, A. Pashalidis, Y. Ohba, F. Bersani, *The Extensible Authentication Protocol-Internet Key Exchange Protocol Version 2 (EAP-IKEv2) Method*, Internet RFC 5106 (experimental), luty 2008.

[RFC5114] M. Lepinski, S. Kent, *Additional Diffie-Hellman Groups for Use with IETF Standards*, Internet RFC 5114 (informational), styczeń 2008.

[RFC5116] D. McGrew, *An Interface and Algorithms for Authenticated Encryption*, Internet RFC 5116, styczeń 2008.

[RFC5155] B. Laurie, G. Sisson, R. Arends, D. Blacka, *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*, Internet RFC 5155, marzec 2008.

[RFC5191] D. Forsberg, Y. Ohba, red., B. Patil, H. Tschofenig, A. Yegin, *Protocol for Carrying Authentication for Network Access (PANA)*, Internet RFC 5191, maj 2008.

[RFC5193] P. Jayaraman, R. Lopez, Y. Ohba, red., M. Parthasarathy, A. Yegin, *Protocol for Carrying Authentication for Network Access (PANA) Framework*, Internet RFC 5193 (informational), maj 2008.

[RFC5216] D. Simon, B. Aboba, R. Hurst, *The EAP-TLS Authentication Protocol*, Internet RFC 5216, marzec 2008.

[RFC5238] T. Phelan, *Datagram Transport Layer Security (DTLS) over the Datagram Congestion Control Protocol (DCCP)*, Internet RFC 5238, maj 2008.

[RFC5246] T. Dierks, E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.2*, Internet RFC 5246, sierpień 2008.

[RFC5247] B. Aboba, D. Simon, P. Eronen, *Extensible Authentication Protocol (EAP) Key Management Framework*, Internet RFC 5247, sierpień 2008.

[RFC5280] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, Internet RFC 5280, maj 2008.

[RFC5281] P. Funk, S. Blake-Wilson, *Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)*, Internet RFC 5281 (informational), sierpień 2008.

[RFC5295] J. Salowey, L. Dondeti, V. Narayanan, M. Nakhjiri, *Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)*, Internet RFC 5295, sierpień 2008.

[RFC5296] V. Narayanan, L. Dondeti, *EAP Extensions for EAP Re-authentication Protocol (ERP)*, Internet RFC 5296, sierpień 2008.

[RFC5322] P. Resnick, red., *Internet Message Format*, Internet RFC 5322, październik 2008.

[RFC5374] B. Weis, G. Gross, D. Ignjatic, *Multicast Extensions to the Security Architecture for the Internet Protocol*, Internet RFC 5374, listopad 2008.

[RFC5386] N. Williams, M. Richardson, *Better-than-Nothing Security: An Unauthenticated Mode of IPsec*, Internet RFC 5386, listopad 2008.

[RFC5387] J. Touch, D. Black, Y. Wang, *Problem and Applicability Statement for Better-than-Nothing Security (BTNS)*, Internet RFC 5387 (informational), listopad 2008.

[RFC5406] S. Bellovin, *Guidelines for Specifying the Use of IPsec Version 2*, Internet RFC 5406/BCP 0146, luty 2009.

[RFC5585] T. Hansen, D. Crocker, P. Hallam-Baker, *DomainKeys Identified Mail (DKIM) Service Overview*, Internet RFC 5585 (informational), lipiec 2009.

[RFC5617] E. Allman, J. Fenton, M. Delany, J. Levine, *DomainKeys Identified Mail (DKIM) Author Domain Signing Practices (ADSP)*, Internet RFC 5617, sierpień 2009.

[RFC5652] R. Housley, *Cryptographic Message Syntax (CMS)*, Internet RFC 5652/STD 0070, wrzesień 2009.

[RFC5702] J. Jansen, *Use of SHA-2 Algorithms with RSA in DNSKEY and RRSIG Resource Records for DNSSEC*, Internet RFC 5702, październik 2009.

[RFC5723] Y. Sheffer, H. Tschofenig, *Internet Key Exchange Protocol Version 2 (IKEv2) Session Resumption*, Internet RFC 5723, styczeń 2010.

[RFC5739] P. Eronen, J. Laganier, C. Madson, *IPv6 Configuration in Internet Key Exchange Protocol Version 2 (IKEv2)*, Internet RFC 5739 (experimental), luty 2010.

[RFC5746] E. Rescorla, M. Ray, S. Dispensa, N. Oskov, *Transport Layer Security (TLS) Renegotiation Indication Extension*, Internet RFC 5746, luty 2010.

[RFC5753] S. Turner, D. Brown, *Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)*, Internet RFC 5753 (informational), styczeń 2010.

[RFC5755] S. Farrell, R. Housley, S. Turner, *An Internet Attribute Certificate Profile for Authorization*, Internet RFC 5755, styczeń 2010.

[RFC5764] D. McGrew, E. Rescorla, *Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-Time Transport Protocol (SRTP)*, Internet RFC 5764, maj 2010.

[RFC5840] K. Grewal, G. Montenegro, M. Bhatia, *Wrapped Encapsulating Security Payload (ESP) for Traffic Visibility*, Internet RFC 5840, kwiecień 2010.

[RFC5857] E. Ertekin, C. Christou, R. Jasani, T. Kivinen, C. Bormann, *IKEv2 Extensions to Support Robust Header Compression over IPsec*, Internet RFC 5857, maj 2010.

[RFC5879] T. Kivinen, D. McDonald, *Heuristics for Detecting ESP-NULL Packets*, Internet RFC 5879 (informational), maj 2010.

[RFC5903] D. Fu, J. Solinas, *Elliptic Curve Groups Modulo a Prime (ECP Groups) for IKE and IKEv2*, Internet RFC 5903 (informational), czerwiec 2010.

[RFC5933] V. Dolmatov, red., A. Chuprina, I. Ustinov, *Use of GOST Signature Algorithms in DNSKEY and RRSIG Resource Records for DNSSEC*, Internet RFC 5933, lipiec 2010.

[RFC5996] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, Internet RFC 5996, wrzesień 2010.

[RFC5998] P. Eronen, H. Tschofenig, Y. Sheffer, *An Extension for EAP-Only Authentication in IKEv2*, wrzesień 2010.

[RFC6024] R. Reddy, C. Wallace, *Trust Anchor Management Requirements*, Internet RFC 6024 (informational), październik 2010.

[RFC6040] B. Briscoe, *Tunnelling of Explicit Congestion Notification*, Internet RFC 6040, listopad 2010.

[RFC6066] D. Eastlake 3rd, *Transport Layer Security (TLS) Extensions: Extension Definitions*, Internet RFC 6066, styczeń 2011.

[RFC6071] S. Frankel, S. Krishnan, *IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap*, Internet RFC 6071 (informational), luty 2011.

[RFC6083] M. Tuexen, R. Seggelmann, E. Rescorla, *Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)*, Internet RFC 6083, styczeń 2011.

[RFC6091] N. Mavrogiannopoulos, D. Gillmor, *Using OpenPGP Keys for Transport Layer Security (TLS) Authentication*, Internet RFC 6091 (informational), luty 2011.

[RFC6101] A. Freier, P. Karlton, P. Kocher, *The Secure Socket Layer (SSL) Protocol Version 3.0*, Internet RFC 6101, sierpień 2011.

[RFC6147] M. Bagnulo, A. Sullivan, P. Matthews, I. van Beijnum, *DN64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers*, Internet RFC 6147, kwiecień 2011.

[RFC6176] S. Turner, S. Polk, *Prohibiting Secure Sockets Layer (SSL) Version 2.0*, Internet RFC 6176, marzec 2011.

[RFC6234] D. Eastlake 3rd, T. Hansen, *US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)*, Internet RFC 6234 (informational), maj 2011.

[RFC6345] P. Duffy, S. Chakrabarti, R. Cragie, Y. Ohba, red., A. Yegin, *Protocol for Carrying authentication for Network Access (PANA) Relay Element*, Internet RFC 6345, sierpień 2011.

[RFC6376] D. Crocker, red., T. Hansen, red., M. Kucherawy, red., *DomainKeys Identified Mail (DKIM) Signatures*, Internet RFC 6376, wrzesień 2011.

[RSA78] R. Rivest, A. Shamir, L. Adleman, *A Method for Obtaining Digital Signatures and Public Key Cryptosystems*, „Communications of the ACM", 21(2), luty 1978.

[TLD-REPORT] *http://stats.research.icann.org/dns/tld_report*

[TLSEXT] *http://www.iana.org/assignments/tls-extensiontype-values*

[TLSPARAMS] *http://www.iana.org/assignments/tls-parameters*

[TNMOC] The National Museum of Computing, *http://www.tnmoc.org*

[TSIGALG] *http://www.iana.org/assignments/tsig-algorithm-names*

[TWP07] E. Tews, R. Weinmann, A. Pyshkin, *Breaking 104 Bit WEP in Less than 60 Seconds*, sprawozdanie z konferencji 8th International Workshop on Information Security Applications (Springer, 2007).

[VENONA] R.L. Benson, National Security Agency Center for Cryptologic History, *The VENONA Story*, *http://www.nsa.gov/public_info/declass/venona*

[VK83] V. Voydock, S. Kent, *Security Mechanisms in High-Level Network Protocols*, „ACM Computing Surveys", 15, czerwiec 1983.

[WY05] X. Wang, H. Yu, *How to Break MD5 and Other Hash Functions*, EUROCRYPT, maj 2005.

[X9.62-2005] American National Standards Institute, *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Standard (ECDSA)*, ANSI X9.62, 2005.

[Z97] Y. Zheng, *Digital Signcryption or How to Achieve Cost(Signature &Encryption) << Cost(Signature) + Cost(Encryption)*, sprawozdanie z konferencji CRYPTO, „Lecture Notes in Computer Science", tom 1294 (Springer-Verlag, 1997).

Słownik akronimów

3GPP *3rd Generation Partnership Project* (organizacja definiująca standardy w zakresie telefonii komórkowej, odpowiedzialna za standardy GSM, WCDMA, LTE itd.)

3GPP2 *3rd Generation Partnership Project 2* (organizacja definiująca standardy w zakresie telefonii komórkowej, odpowiedzialna za standardy CDMA2000, EV-DO itd.)

6rd *IPv6 Rapid Deployment* (szybkie wdrożenie IPv6 — mechanizm przejścia do protokołu IPv6, umożliwiający przesyłanie ruchu IPv6 przez sieci IPv4, podobny do mechanizmu 6to4, ale korzystający z przypisań prefiksów IPv6 opartych na przypisanych adresach transmisji pojedynczej)

6to4 *Six to Four* (przesyłanie ruchu IPv6 w tunelach IPv4, w mechanizmie 6to4 występują pewne problemy funkcjonalne)

A *Address* (IPv4) (rekord zasobów DNS zawierający adres IPv4)

AAA *Authentication, Authorization and Accounting* (uwierzytelnianie, autoryzacja i rozliczanie — możliwości systemu zarządzania związane z pewnymi protokołami dostępu, takimi jak RADIUS i Diameter)

AAAA *Address* (IPv6) (rekord zasobów DNS zawierający adres IPv6)

ABC *Appropriate Byte Counting* (odpowiednie zliczanie bajtów — w kontroli przeciążenia protokołu TCP metoda uwzględniająca liczbę potwierdzonych bajtów zamiast stałego współczynnika przy wykonywaniu obliczeń dotyczących okna CWND; może złagodzić problem powolnego wzrostu rozmiaru okna związanego z opóźnionymi potwierdzeniami ACK)

AC *Attribute Certificate* (certyfikat atrybutu — typ certyfikatu używany do poświadczania atrybutów, takich jak uprawnienia, ale niezawierający klucza publicznego i dlatego różniący się od PKC)

ACCM *Asynchronous Control Character Map* (tablica znaków sterowania asynchronicznego — w protokole PPP wskazuje, które bajty powinny być cytowane w celu uniknięcia niepożądanych efektów)

ACD *Automatic Collision Detection* (automatyczne wykrywanie kolizji — procedura wykrywania i unikania kolizji w przypisywaniu adresów IP)

ACFC *Address and Control Field Compression* (kompresja pól adresu i sterowania — w protokole PPP wyeliminowanie pola adresu i pola sterowania w celu zmniejszenia narzutu)

ACK *Acknowledgement* (potwierdzenie — wskazanie, że dane dotarły pomyślnie do odbiorcy; ma zastosowanie w wielu warstwach stosu protokołów)

ACL *Access Control List* (lista kontroli dostępu — lista reguł filtrowania ruchu sieciowego, określająca, jaki ruch jest dozwolony, np. przez zaporę sieciową)

ADSP *Author Domain Signing Practices* (praktyki podpisywania dla domeny autora — rozszerzenie systemu DKIM, deklaracja polityki dotycząca tego, w jaki sposób system uwierzytelniania DKIM jest stosowany w konkretnej domenie)

AEAD *Authenticated Encryption with Associated Data* (uwierzytelnione szyfrowanie z danymi towarzyszącymi — algorytmy wykonujące szyfrowanie i uwierzytelnianie dla jednej części swoich danych wejściowych oraz uwierzytelnianie dla innej części)

AES *Advanced Encryption Standard* (zaawansowany standard szyfrowania — standard szyfrowania obecnej generacji obowiązujący w USA)

AF *Assured Forwarding* (przekazywanie gwarantowane — wariant PHB obsługujący klasy priorytetów oraz priorytetyzację wewnątrz klas)

AFTR *Address Family Transition Router* element (w DS-Lite, SPNAT umożliwiający współdzielenie małej liczby adresów IPv4 przez wielu klientów)

AH *Authentication Header* (nagłówek uwierzytelniający — opcjonalny protokół z rodziny IPsec umożliwiający uwierzytelnianie ruchu IP, zawierający w nagłówku informacje, które są niekompatybilne z NAT)

AIA *Authority Information Access* (dostęp do informacji o urzędach certyfikacyjnych — rozszerzenie certyfikatu X.509 wskazujące zasoby użyteczne dla weryfikacji certyfikatu)

AIAD *Additive Increase Additive Decrease* (addytywne zwiększenie, addytywne zmniejszenie — w protokole TCP metody, które łagodzą zmiany okna CWND przez dodanie do jego rozmiaru pewnej dodatniej wartości, kiedy poziom przeciążenia wydaje się być niski, i odjęcie od jego rozmiaru pewnej dodatniej wartości, kiedy przeciążenie zdaje się wzrastać; nie jest to standardowy algorytm protokołu TCP)

AIMD *Additive Increase Multiplicative Decrease* (addytywne zwiększenie, multiplikatywne zmniejszenie — w protokole TCP metody, które łagodzą zmiany okna CWND przez dodanie do jego rozmiaru pewnej dodatniej wartości, kiedy poziom przeciążenia wydaje się być niski, i pomnożenie jego rozmiaru przez pewien, mniejszy od 1 ułamek, kiedy przeciążenie zdaje się wzrastać)

ALG *Application Layer Gateway* (brama warstwy aplikacji — agent, zazwyczaj programowy, który wykonuje konwersję protokołów w warstwie aplikacji)

A-MPDU *Aggregated MPDU* (zagregowana MPDU — ramka zawierająca wiele jednostek MPDU, część standardu IEEE 802.11n)

A-MSDU *Aggregated MSDU* (zagregowana MSDU — ramka zawierająca wiele jednostek MSDU, część standardu IEEE 802.11n)

ANDSF *Access Network Discovery and Selection Function* (funkcja odkrywania i wyboru sieci dostępowych — część usług MoS dostarczająca informacji o sieciach, które mogą być użyte, wpływająca na przełączanie między stacjami bazowymi i wybór sieci)

AODV *Ad-hoc On-Demand Distance Vector routing protocol* (wczesny protokół routingu na żądanie dla sieci ad hoc, wykorzystujący wektor odległości)

AP *Access Point* (punkt dostępowy — wg standardu 802.11, stacja [STA] wykorzystywana zwykle do połączenia bezprzewodowego i przewodowego segmentu sieci)

API *Application Programming Interface* (interfejs programowania aplikacji — funkcje wywoływane przez aplikacje w celu uzyskania takich efektów jak wysyłanie i odbieranie ruchu sieciowego)

APIPA *Automatic Private IP Addressing* (automatyczne adresowanie prywatne IP — mechanizm, za pomocą którego węzeł samodzielnie konfiguruje swój własny adres IP z określonego zakresu; stosowany zwykle przez węzły IPv4)

APSD *Automatic Power Save Delivery* (dostarczanie z automatycznym oszczędzaniem energii — okresowe przetwarzanie wsadowe ramek 802.11 obsługujące tryb PSM)

AQM *Active Queue Management* (aktywne zarządzanie kolejką — metody zarządzania kolejką, które reagują na dynamikę ruchu, różne od „obcinania ogona" — typowego sposobu zarządzania kolejką FCFS/FIFO)

ARP *Address Resolution Protocol* (protokół rozpoznawania adresów — protokół powyżej warstwy łącza danych, który wykonuje konwersję adresów IPv4 na adresy warstwy MAC, korzystając z adresowania rozgłoszeniowego warstwy łącza danych)

ARQ *Automatic Repeat Request* (automatyczne żądanie ponownego przesłania — retransmisja informacji; zwykle po wywnioskowanej utracie danych)

AS *Authentication Server* (serwer uwierzytelniający — w kontekście protokołu PANA serwer, w którym wykonywane są czynności sprawdzające związane z uwierzytelnieniem)

AS *Autonomous System* (system autonomiczny — liczba 16- lub 32-bitowa używana w związku z routingiem między dostawcami ISP do identyfikacji zbioru prefiksów sieci i ich właściciela)

ASM *All-Source Multicast* (multiemisja z dowolnego źródła — multiemisja, w której każdy uczestnik może być źródłem ruchu)

ASN.1 *Abstract Syntax Notation One* (abstrakcyjna notacja składniowa numer jeden — standard ISO definiujący abstrakcyjną składnię dla informacji, ale bez odpowiadającego jej formatu kodowania; do metod kodowania informacji ASN.1 należą BER i DER)

AUS *Application Unique String* (unikalny łańcuch aplikacji — łańcuch wejściowy algorytmu DDDS)

AUTH *Authentication* (uwierzytelnienie — w związku z protokołem IKE, ładunek użyteczny zawierający informacje wymagane do wykonania uwierzytelnienia nadawcy)

AXFR *Zone Transfer* (transfer strefy — pełna wymiana informacji o strefie DNS; korzysta z protokołu TCP)

B4 *Basic Bridging BroadBand* element (w technologii DS-Lite router, który kapsułkuje ruch IPv4 w tunelach IPv6 zakończonych w routerze AFTR, router B4 nie wykonuje funkcji NAT)

BACP *Bandwidth Allocation Control Protocol* (protokół kontroli przydzielania pasma — w protokole PPP protokół do konfigurowania mechanizmu BoD)

BAP *Bandwidth Allocation Protocol* (protokół przydzielania pasma — protokół używany do konfigurowania łączy w wiązkę obsługiwaną przez MPPP)

BCMCS *Broadcast and Multicast Service* Controller (kontroler usług rozgłaszania i multiemisji — w sieciach telefonii komórkowej zarządza multiemisją)

BER *Basic Encoding Rules* (podstawowe reguły kodowania — składnia kodowania zgodna ze standardem ITU; podzbiór ASN.1)

BER *Bit Error Rate* (współczynnik błędów bitowych — oczekiwana liczba błędów bitowych w stosunku do ilości transmitowanych bitów)

BGP *Border Gateway Protocol* (protokół bram granicznych — protokół routingu międzydomenowego z obsługą strategii)

BIND9 *Berkeley Internet Name Domain* (version 9) (implementacja programowa serwera nazw, popularna w systemach uniksopodobnych)

BITS *Bump In the Stack* (guz na stosie — opcja implementacji protokołu IPsec na hoście)

BITW *Bump In the Wire* (guz na drucie — opcja implementacji protokołu IPsec w sieci)

BL *Bulk Leasequery* (w protokole DHCP protokół typu żądanie-odpowiedź służący do przekazywania bieżącej informacji dotyczącej przydziału adresów IP)

BoD *Bandwidth on Demand* (pasmo na żądanie — zdolność do dynamicznej regulacji dostępnego pasma łącza)

BOOTP *Bootstrap Protocol* (protokół ładowania — prekursor protokołu DHCP; używany do konfigurowania hostów)

BPDU *Bridge PDU* (PDU mostu — jednostka PDU używana przez protokół STP; wymieniana przez przełączniki i mosty)

BPSK *Binary Phase Shift Keying* (binarne kluczowanie z przesunięciem fazy — modulowanie binarne przy wykorzystaniu dwóch faz sygnału)

BSD *Berkeley Software Distribution* (wersja systemu UNIX Uniwersytetu Kalifornijskiego Berkeley, która zawierała pierwszą szeroko używaną implementację TCP/IP)

BSDP *Boot Server Discovery Protocol* (protokół odnajdowania serwera rozruchowego — rozszerzenie protokołu DHCP opracowane przez firmę Apple służące do odnajdowania serwera z obrazem rozruchowym)

BSS *Basic Service Set* (podstawowa grupa usługowa — zgodnie ze standardem IEEE 801.11 termin określający punkt dostępowy i powiązane z nim stacje bezprzewodowe)

BTNS *Better Than Nothing Security* (bezpieczeństwo lepsze niż nic — w związku z protokołem IPsec opcja używania certyfikatów bez pełnej infrastruktury PKI, która jednak jest nieodporna na ataki MITM)

BU *Binding Update* (aktualizacja powiązania — w protokole MIP ustanawia odwzorowanie między adresami CoA i HoA węzła mobilnego)

CA *Certificate Authority* (urząd certyfikacji — organizacja odpowiedzialna za generowanie i wydawanie par kluczy publiczny-prywatny oraz za podpisywanie i dystrybucję podpisanych kluczy publicznych i list CRL)

CALIPSO *Common Architecture Label IPv6 Security Option* (etykiety bezpieczeństwa dla pakietów IP; nie są szeroko używane)

CBC *Cipher Block Chaining* (wiązanie zaszyfrowanych bloków — metoda szyfrowania, która korzysta z operacji XOR do łączenia ze sobą zaszyfrowanych bloków w celu odparcia ataków przez zmianę kolejności bloków)

CBCP *Callback Control Protocol* (protokół kontroli wywołania zwrotnego — w protokole PPP ustala numer wywołania zwrotnego)

CCA *Clear Channel Assessment* (ocena czystego kanału — mechanizm warstwy fizycznej standardu 802.11, który wykrywa używanie kanału)

CCITT *Comité Consultatif International Téléphonique et Télégraphique* (Międzynarodowy Komitet Doradczy do spraw Telefonii i Telegrafii — obecnie ITU-T)

CCM *Counter Mode with CBC Message Authentication Code* (metoda licznikowa z kodem uwierzytelnienia wiadomości CBC — metoda uwierzytelnionego szyfrowania łącząca metodę szyfrowania CTR z CBC-MAC)

CCMP *Counter Mode with CBC-MAC Protocol* (protokół metody licznikowej z CBC-MAC — szyfrowanie użyte w implementacji WPA2 standardu IEEE 802.11i zastępującej standard WPA)

CCP *Compression Control Protocol* (protokół kontroli kompresji — w protokole PPP ustala używane metody kompresji)

ccTLD *Country Code TLD* (krajowa domena najwyższego poziomu — domena TLD na liście kodów krajowych ISO3661-2)

CDP *CRL Distribution Point* (punkt dystrybucji CRL — miejsce w sieci, skąd można uzyskać aktualną listę CRL danego urzędu certyfikacyjnego)

CERT *Certificate* (certyfikat — w protokole IKE ładunek użyteczny pokazujący tożsamość nadawcy)

CERT *Computer Emergency Response Team* (zespół ds. reagowania na przypadki naruszenia bezpieczeństwa teleinformatycznego — grupy, które zajmują się przypadkami naruszenia bezpieczeństwa teleinformatycznego, w tym pierwszy zespół CERT Uniwersytetu Carnegie Mellon i US-CERT rządu Stanów Zjednoczonych)

CERTREQ *Certificate Request* (żądanie certyfikatu — w protokole IKE ładunek użyteczny określający kotwicę zaufania jako wskazanie akceptowalnych certyfikatów)

CGA *Cryptographically Generated Address* (adres wygenerowany kryptograficznie — adres wygenerowany na podstawie skrótu klucza publicznego)

CHAP *Challenge-Handshake Authentication Protocol* (protokół uwierzytelniania wezwanie-uzgodnienie — protokół wymagający wezwania pasującego do odpowiedzi; podatny na ataki MITM)

CIA *confidentiality, integrity, and availability* (poufność, integralność i dostępność — zasady bezpieczeństwa informacji; tzw. triada CIA)

CIDR *Classless Inter-Domain Routing* (bezklasowy routing międzydomenowy — posunięcie zmierzające do zaradzenia problemowi ROAD przez usunięcie granic między klasami adresów IP, ale wymagające użycia specjalnej maski CIDR w routingu międzydomenowym)

CMAC *Cipher-based Message Authentication Code* (kod uwierzytelnienia wiadomości oparty na szyfrze — konkretny sposób użycia algorytmów szyfrowania jako MAC)

CN *Correspondent Node* (węzeł korespondujący — partner węzła mobilnego w komunikacji wg scenariusza MIP)

CNAME *Canonical Name* (nazwa kanoniczna — rekord zasobów DNS definiujący alias dla innej nazwy domeny)

CoA *Care-of Address* (adres pośredni — adres przypisany do węzła mobilnego w czasie, gdy rezyduje w sieci obcej)

CoS *Class of Service* (klasa usługi — ogólny termin odnoszący się do usług zróżnicowanych opartych na różnych klasach ruchu sieciowego; koncepcja obsługiwana przez architekturę DiffServ)

CoT *Care-of Test* (test adresu pośredniego — w sprawdzeniu routowalności odwrotnej komunikat wysłany do węzła mobilnego za pośrednictwem jego adresu CoA, w wyniku którego MN uzyskuje część klucza używanego do zabezpieczenia aktualizacji powiązania BU)

CoTI *Care-of Test Init* (inicjacja testu adresu pośredniego — w sprawdzeniu routowalności zwrotnej powoduje wysłanie komunikatu CoT przez odbiorcę)

CP *Configuration Payload* (w protokole IKE rozszerzalna struktura przekazująca parametry konfiguracyjne)

CPS *Certification Practice Statement* (kodeks postępowania certyfikacyjnego — dokument opisujący zasady działania urzędu certyfikacji dotyczące sposobu wydawania certyfikatów i zarządzania nimi)

CRC *Cyclic Redundancy Check* (cykliczny kod nadmiarowy — funkcje matematyczne używane do wykrywania błędów bitowych)

CRL *Certificate Revocation List* (lista anulowanych certyfikatów — lista nieważnych certyfikatów opublikowana przez CA)

CS *Cipher Suite* (zestaw szyfrów — w protokole TLS wybór zestawu algorytmów kryptograficznych)

CS *Class Selector* (selektor klasy — w protokole IP wartość DSCP zaprojektowana tak, aby była kompatybilna z wartościami bitowymi związanymi z obecnie niezalecanymi polami nagłówka IP „Typ usługi" i „Klasa ruchu")

CSMA/CA *Carrier-Sense Multiple Access/Collision Avoidance* (wielodostęp do łącza danych z badaniem stanu kanału/unikanie kolizji — protokół podwarstwy MAC standardu Wi-Fi, którego działanie wiąże się z wysyłaniem danych, gdy łącze jest bezczynne, i wycofywaniem się w przeciwnym wypadku)

CSMA/CD *Carrier-Sense Multiple Access/Collision Detection* (wielodostęp do łącza danych z badaniem stanu kanału/wykrywanie kolizji — klasyczny protokół podwarstwy MAC Ethernetu, którego działanie wiąże się z wysyłaniem danych, gdy łącze jest bezczynne, i wycofywaniem się, jeśli zostały wykryte kolizje)

CSPRNG *Cryptographycially Secure Preudo-Random Number Generator* (kryptograficznie bezpieczny generator liczb pseudolosowych — generator liczb pseudolosowych odpowiedni do użycia w kryptografii)

CSRG *Computer Systems Research Group* (grupa badawcza ds. systemów komputerowych — twórcy systemu BSD UNIX w Uniwersytecie Kalifornijskim Berkeley)

CTCP *Compound TCP* (złożone TCP — „skalowalny" wariant protokołu TCP zaimplementowany w nowszych systemach Windows, który łączy regulację rozmiaru okna na podstawie opóźnienia z regulacją na podstawie utraty pakietów)

CTR *Counter* (metoda licznikowa — sposób szyfrowania, który wykorzystuje wartość licznika w celu narzucenia wymaganego porządku zaszyfrowanych bloków, jednocześnie pozwalając na równoległe wykonywanie operacji szyfrowania i deszyfrowania na wielu blokach)

CTS *Clear To Send* (zgoda na wysłanie — komunikat upoważniający nadawcę komunikatu RTS do transmisji danych)

CW *Contention Window* (okno rywalizacji — przedział czasu, przez jaki stacja 802.11 będzie odczekiwać przed transmisją, zgodnie z DCF)

CWND *Congestion Window* (okno przeciążenia — w protokole TCP ograniczenie nałożone na rozmiar okna nadawcy w celu uniknięcia lub zmniejszenia przeciążenia)

CWR *Congestion Window Reducing* (lub *Reduced*) (redukcja okna przeciążenia — w protokole TCP redukcja rozmiaru użytecznego okna nadawcy)

CWV *Congestion Window Verification* (weryfikacja okna przeciążenia — w protokole TCP metoda kontroli i aktualizacji bieżącej wartości okna CWND, kiedy jest to konieczne)

DAD *Duplicate Address Detection* (wykrywanie zduplikowanych adresów — w mechanizmach ND i SLAAC protokołu IPv6 to DAD przez wysłanie komunikatu NS dla proponowanego adresu pomaga ustalić, czy kandydujący adres IPv6 jest już w użyciu)

DCCP *Datagram Congestion Control Protocol* (datagramowy protokół kontroli przeciążenia — protokół, który oferuje aplikacjom możliwie najlepszą usługę datagramową oraz zapewnia kontrolę przeciążenia)

DCF *Distributed Coordination Function* (rozproszona funkcja koordynacji — technika MAC zgodna z protokołem CSMA/CA dla sieci 802.11)

DDDS *Dynamic Delegation Discovery System* (system odkrywania dynamicznego delegowania — metody obsługujące leniwe wiązanie łańcuchów znaków z danymi; zazwyczaj używane w DNS do odnajdowania serwerów dla różnych protokołów warstwy aplikacji)

DDoS *Distributed DoS* (rozproszony DoS — atak z wykorzystaniem sieci, często przeprowadzany przez botnety)

DER *Distinguished Encoding Rules* (jednoznaczne reguły kodowania — standard ITU składni kodowania; podzbiór reguł BER standardu ASN.1, który wymaga użycia jednoznacznej reprezentacji dla każdej wartości)

DES *Data Encryption Standard* (standard szyfrowania danych — obowiązujący dawniej w USA standard szyfrowania danych z kluczem symetrycznym, wykorzystujący klucze 56-bitowe)

DF *Don't Fragment* (nie fragmentuj — bit nagłówka protokołu IPv4 wskazujący, że nie powinna być wykonywana żadna fragmentacja; ważny dla techniki PMTUD)

DH *Diffie-Hellman* (matematyczny protokół ustanowienia sekretnej wartości między dwoma stronami nawet w obecności podsłuchującego)

DHCP *Dynamic Host Configuration Protocol* (protokół dynamicznej konfiguracji hostów — wyewoluował z protokołu BOOTP; konfiguruje systemy, dostarczając im dane konfiguracyjne, takie jak dzierżawione adresy IP, adres IP domyślnego routera i serwera DNS)

DIFS *DCF Inter-Frame Space* (odstęp między ramkami dla DCF — czas między ramkami w technice DCF standardu 802.11)

DIX *Digital, Intel, Xerox* (nazwa wczesnego standardu Ethernetu utworzona od nazw jego twórców)

DKIM *Domain Keys Identified Mail* (uwierzytelnianie poczty e-mail za pomocą klucza domeny — protokół służący do kryptograficznego powiązania domeny nadawcy z wysyłającym wiadomość serwerem pocztowym)

DLNA *Digital Living Network Alliance* (grupa branżowa zajmująca się głównie interoperacyjnością i protokołami dla konsumpcyjnych urządzeń medialnych, takich jak telewizory, odtwarzacze DVD i DVR)

DMZ *De-Militarized Zone* (strefa zdemilitaryzowana — obszar sieci na zewnątrz wewnętrznej zapory sieciowej organizacji, zwykle wykorzystywany do umieszczania w nim hostów dostarczających usług klientom lub ogółowi użytkowników)

DNA *Detecting Network Attachment* (wykrywanie podłączenia do sieci — procedury wykrywające zmiany stanu połączenia z siecią)

DNAME *Non-Terminal DNS Name Redirection* (przekierowywanie nieterminalnej nazwy domeny — rekord zasobów DNS obsługujący generowanie wielu rekordów CNAME przy użyciu mechanizmu tworzenia aliasów dla poddrzewa DNS)

DNS *Domain Name System* (system nazw domenowych — odwzorowuje nazwy na adresy IP i wykonuje inne czynności)

DNS64 *DNS IPv4/IPv6 translation* (mechanizm wspierający koegzystencję protokołów IPv4 i IPv6, tłumaczący informacje DNS z serwerów korzystających z IPv4 w celu ich wykorzystania w DNS IPv6)

DNSKEY *Key for DNS* (klucz dla DNS — rekord zasobów DNS związany z DNSSEC, przechowujący klucz publiczny)

DNSSEC *DNS Security* (bezpieczeństwo DNS — zapewnienie uwierzytelniania źródeł i integralności danych DNS)

DNSSL *DNS Search List* (używana z komunikatami RA, wskazuje listę domyślnych rozszerzeń domeny)

DOI *Digital Object Identifier* (cyfrowy identyfikator dokumentu elektronicznego — metoda nazywania dokumentów elektronicznych i wiązania ich z rekordami bazy danych zawierającymi informacje o nich)

DoS *Denial of Service* (odmowa usługi — rodzaj ataku opartego na wyczerpaniu zasobów)

DPD *Delegated Path Discovery* (delegowane odkrywanie ścieżki — metoda delegowania zadania pozbierania wszystkich informacji wymaganych do walidacji ścieżki certyfikacji)

DPV *Delegated Path Validation* (delegowana walidacja ścieżki — metoda delegowania całej procedury walidacji certyfikatu)

DS *Delegation Signer* (w systemie DNS rekord zasobów używany w DNSSEC do zabezpieczenia delegowania)

DS *Differentiated Services* (usługi zróżnicowane — w zarządzaniu ruchem IP metody wprowadzające zróżnicowanie wydajności w dostarczaniu ruchu sieciowego)

DS *Distribution Service* (usługa dystrybucji — w sieciach LAN standardu 802.11 sieć lub usługa używana do wzajemnego łączenia punktów dostępowych, która jest najczęściej przewodową siecią 802.3/Ethernet)

DSA *Digital Signature Algorithm* (algorytm podpisu cyfrowego — algorytm służący do generowania podpisów cyfrowych oparty na trudności obliczania logarytmów dyskretnych)

DSACK *Duplicate SACK* (zduplikowana informacja SACK — w protokole TCP wariant opcji SACK, który zawiera opis odebranych zduplikowanych segmentów)

DSCP *DS Code Point* (punkt kodowy DS — wartość pola w pakiecie wskazująca wymagany sposób przekazywania pakietu)

DSL *Digital Subscriber Line* (cyfrowa linia subskrybencka — dedykowane szerokopasmowe łącze danych wykorzystujące linię POTS)

DS-Lite *Dual Stack Lite* (struktura dla dostawców usług opartych na IPv6, umożliwiająca dostęp do tych usług klientom z podwójnym lub pojedynczym stosem protokołów dzięki wykorzystaniu połączenia tunelowania IPv4-in-IPv6 z technologią NAT)

DSRK *Domain-Specific Root Key* (klucz wyprowadzony z klucza EMSK przeznaczony do używania przez systemy pozostające pod zarządem pojedynczego administratora)

DSS *Digital Signature Standard* (obowiązujący w USA standard podpisów elektronicznych oparty na algorytmie DSA)

DSUSRK *Domain-Specific USRK* (klucz łączący zasady użycia kluczy USRK i DSRK)

DTLS *Datagram TLS* (datagramowy TLS — wariant protokołu TLS używany z protokołami datagramowymi, takimi jak UDP)

DUID *DHCP Unique Identifier* (unikatowy identyfikator DHCP — wartość umieszczona w żądaniu DHCP służąca do dopasowania odpowiedzi)

DUP *Duplicate* (zduplikowany, -a, -e — termin używany w wielu kontekstach, np. DUP ACKs — zduplikowane potwierdzenia ACK)

EAP *Extensible Authentication Protocol* (rozszerzalny protokół uwierzytelniania — struktura obsługująca różne metody uwierzytelniania)

EAP-FAST *EAP — Flexible Authentication via Security Tunneling* (EAP — elastyczne uwierzytelnianie przy użyciu bezpiecznego tunelowania — metoda EAP firmy Cisco korzystająca z TLS, która zastąpiła wcześniejszą metodę tej firmy znaną jako LEAP EAP)

EAPOL *EAP over LAN* (np. EAP over Ethernet w standardzie IEEE 802.1X)

EAP-TTLS *EAP — Tunneled Transport Layer Security* (metoda EAP oparta na wcześniejszej metodzie TLS EAP, ale wymagająca do uzyskania certyfikatu tylko serwera)

EC2N *Elliptic Curve groups modulo a power of 2* (grupy krzywych eliptycznych modulo potęga liczby 2 — grupy oparte na krzywych eliptycznych, w sensie algebry abstrakcyjnej, nad ciałem Galois $GF(2^N)$)

ECC *Error Correcting Code* (kod korekcji błędów — nadmiarowe bity dodane do bitów informacji używane do korygowania błędów)

ECDSA *Elliptic Curve Digital Signature Algorithm* (algorytm podpisu cyfrowego z wykorzystaniem krzywych eliptycznych — wariant algorytmu DSA wykorzystujący kryptografię krzywych eliptycznych)

ECE *ECN Echo* (w protokole TCP odzwierciedlenie informacji ECN przekazywane do nadawcy)

ECN *Explicit Congestion Notification* (jawne powiadomienie o przeciążeniu — bezpośrednia metoda wskazywania przeciążenia — np. hostom przez routery)

ECP *Elliptic Curve groups modulo a Prime* (grupy krzywych eliptycznych modulo liczba pierwsza — grupy oparte na krzywych eliptycznych, w sensie algebry abstrakcyjnej, nad ciałami Galois GF(P), gdzie P jest liczbą pierwszą)

ECT *ECN-Capable Transport* (protokół transportowy zdolny do interpretacji wskaźników ECN)

EDCA *Enhanced Distributed Channel Access* (rozszerzony rozproszony dostęp do kanału — funkcja koordynująca standardu 801.11, obsługująca QoS, wprowadzona w standardzie 802.11e)

EDNS0 *Extension mechanisms for DNS (version 0)* (mechanizmy rozszerzeń DNS, wersja 0 — metoda rozszerzenia rekordów zasobów DNS, wersja 0, wymagana przez DNSSEC)

EF *Expedited Forwarding* (przekazywanie przyśpieszone — wariant PHB oferujący klasę usług niezależną od przeciążenia, z czego ogólnie wynika jej najwyższy priorytet, a więc i wymaganie kontroli jej dostępności w celu uniknięcia skutków nadmiernego ruchu w tej klasie)

EFO *Expanded Flags Option* (opcja rozszerzonego zbioru flag — wykorzystywana w protokole DHCP, wskazuje obecność dodatkowych opcji)

EIFS *Extended IFS* (rozszerzony odstęp między ramkami — używany po odebraniu nierozpoznanej ramki w protokole 802.11 DCF)

EMSK *Extended MSK* (rozszerzony MSK — dodatkowy klucz wygenerowany oprócz klucza MSK przez protokół EAP po operacji wyprowadzania kluczy)

ENUM *E.164 to URI DDDS Application* (aplikacja DDDS konwersji E.164 do URI — konkretna aplikacja DDDS używana do odwzorowywania numerów telefonicznych zgodnych ze standardem E.164 na adresy URI)

EP *Enforcement Point* (punkt wymuszania — w protokole PANA punkt, w którym wymuszane są zasady kontroli dostępu)

EQM *Equal Modulation* (jednakowa modulacja — używanie tego samego mechanizmu modulacji w różnych strumieniach danych jednocześnie)

ERE *Eligible Rate Estimate* (oszacowanie odpowiedniej szybkości — część protokołu TCP Westwood+; oszacowanie ilości pasma, które może być wykorzystane przez połączenie)

ERP *EAP Re-authentication Protocol* (protokół ponownego uwierzytelnienia przy użyciu EAP — rozszerzenie protokołu EAP służące do zmniejszenia opóźnienia przy ponownym przeprowadzaniu uwierzytelnienia)

ESN *Extended Sequence Number* (rozszerzony numer sekwencyjny — w protokole IPsec rozszerzony 64-bitowy numer sekwencyjny używany do zwalczania ataków przez powtórzenie; normalne numery sekwencyjne mają 32 bity)

ESP *Encapsulating Security Payload* (obowiązkowy protokół w zbiorze IPsec, zapewniający uwierzytelnianie i (lub) poufność danych)

ESSID *Extended Service Set Identifier* (rozszerzony identyfikator grupy usługowej — nazwa sieci IEEE 802.11)

EUI *Extended Unique Identifier* (rozszerzony identyfikator unikatowy — format prefiksu adresu warstwy MAC zdefiniowany przez IEEE, rozszerzenie OUI)

EV *Extended Validation* (rozszerzona walidacja — forma certyfikatu z rozszerzoną walidacją identyczności wykonywaną przed jego wystawieniem)

EV-DO *Evolution, Data Optimized* (lub *Only*) (standard szerokopasmowego bezprzewodowego dostępu do internetu, jeden ze standardów 3GPP2; ewolucja standardu CDMA2000)

FACK *Forward Acknowledgement* (potwierdzenie generowane w przód — w protokole TCP numer sekwencyjny o jeden większy od najwyższego numeru sekwencyjnego, o którym wiadomo, że dotarł do odbiorcy; ustalany przy użyciu opcji SACK)

FCFS *First Come, First Served* (pierwszy nadszedł, pierwszy obsłużony — dyscyplina szeregowania z obsługą według kolejności przyjścia; bez stosowania priorytetów)

FCS *Frame Check Sequence* (ciąg kontrolny ramki — ogólny termin dotyczący bitów służących do wykrywania błędów bitowych)

FEC *Forward Error Correction* (korekcja błędów z wyprzedzeniem — wykorzystanie bitów nadmiarowych do skorygowania błędów w bitach danych)

FIFO *First In, First Out* (pierwszy na wejściu — pierwszy na wyjściu — dyscyplina zarządzania kolejką z obsługą według kolejności; bez zmian w uporządkowaniu)

FIN *Finish* (zakończ — bit nagłówka protokołu TCP i typ ostatniego segmentu przesyłanego w połączeniu TCP)

FMIP *Mobile IP with Fast Handovers* (mobilne IP z szybkimi zmianami routera dostępowego — modyfikacja protokołu MIPv6 z szybszymi zmianami routera dostępowego)

FQDN *Fully Qualified Domain Name* (pełna, jednoznaczna nazwa domenowa — nazwa domenowa zawierająca pełne rozszerzenie domeny)

F-RTO *Forward RTO* (w protokole TCP metoda wnioskowania pozwalająca rozstrzygnąć, czy retransmisja była zbędna, i przez to ułatwiająca unikanie niepotrzebnych retransmisji)

FTP *File Transfer Protocol* (oparty na protokole TCP protokół transferu plików używający odrębnych połączeń do sterowania i transmisji danych)

GARP *Generic Attribute Registration Protocol* (ogólny protokół rejestracji atrybutów — zastąpiony przez MRP)

GCKS *Group Controller/Key Server* (kontroler grupy/serwer kluczy — w protokole IPsec wykorzystywany w systemie GKM, przechowuje i wydaje klucze dla skojarzeń GSA)

GCM *Galois/Counter Mode* (Galois/metoda licznikowa — metoda uwierzytelnionego szyfrowania łącząca metodę szyfrowania CTR z metodą uwierzytelniania Galois)

GDOI *Group Domain of Interpretation* (grupowa domena interpretacji — w zbiorze IPsec protokół zarządzania kluczami grupy oparty na protokołach ISAKMP i IKE)

GENA *General Event Notification Architecture* (ogólna architektura powiadomień o zdarzeniach — oparta na języku XML ramowa struktura powiadomień używająca protokołu HTTP na bazie protokołu UDP z rozsyłaniem grupowym; stosowana z architekturą UPnP)

GI *Guard Interval* (okres ochronny — w technice telekomunikacji minimalny czas między transmisjami służący do uniknięcia interferencji międzysymbolowych)

GKM *Group Key Management* (zarządzanie kluczami grupy — w protokole IPsec metody dystrybucji kluczy do grupy, obsługujące tworzenie grupowych skojarzeń bezpieczeństwa)

GMAC *Galois Message Authentication Code* (kod uwierzytelniania wiadomości Galois — wariant GCM ograniczający się do uwierzytelniania)

GMI *Group Membership Interval* (okres członkostwa grupy — w protokołach IGMP i MLD ilość czasu odczekiwana przez router multicastowy przed zdecydowaniem, że nie ma żadnego konkretnego źródła albo nie ma już członków w grupie; otrzymuje wartość *QRV·QI+QRI*)

GMRP *Generic Multicast Registration Protocol* (ogólny protokół rejestracji multicastowej — zastąpiony przez protokół MMRP)

GPAD *Group PAD* (grupowe PAD — w protokole IPsec abstrakcja bazy danych zawierającej dane uwierzytelniające dla wszystkich jednostek GCKS)

GRE *Generic Routing Encapsulation* (ogólne kapsułkowanie dla routingu — ogólne kapsułkowanie w datagramach IP)

GSA *Group Security Association* (grupowe skojarzenie bezpieczeństwa — w protokole IPsec skojarzenie bezpieczeństwa ustanowione między członkami grupy używającymi protokołu rozgłaszania grupowego)

GSAKMP *Group Secure Association Key Management Protocol* (protokół zarządzania kluczami grupowych skojarzeń bezpieczeństwa — struktura ramowa do tworzenia grup ze wspólną informacją kryptograficzną, polityką dystrybucji, wykonywaniem kontroli dostępu, generowaniem kluczy grupy i odtwarzaniem po dynamicznych zmianach w grupie)

GSPD *Group SPD* (grupowe SPD — w protokole IPsec baza SPD zdolna do przechowywania informacji dotyczących zarówno skojarzeń SA, jak i GSA)

GSS-API *Generic Security Services API* (ogólny interfejs API usług bezpieczeństwa — interfejs API umożliwiający dostęp do niezliczonych usług bezpieczeństwa, takich jak uwierzytelnianie, poufność itd., zazwyczaj używany z systemem uwierzytelniania Kerberos)

gTLD *Generic TLD* (domena funkcjonalna najwyższego poziomu — domena TLD, taka jak COM, EDU, MIL — nieoparta na kodzie krajowym)

GVRP *GARP VLAN Registration Protocol* (protokół GARP z rejestracją sieci VLAN — zastąpiony przez MVRP)

HA *Home Agent* (agent domowy — system oferujący usługę pomocniczą MIP dla węzła mobilnego)

HAIO *Home Agent Information Option* (opcja informacji o agencie domowym — w protokole ICMPv6 opcja obsługująca protokół MIPv6 we wskazaniu adresu HA)

HCF *Hybrid Coordination Function* (funkcja koordynacji hybrydowej — funkcja koordynująca obsługująca dostęp do kanału w standardzie 802.11 oparty zarówno na priorytecie, jak i na rywalizacji)

HDLC *High-level Data Link Control* (wysokopoziomowe sterowanie łączem danych — popularny protokół warstwy łącza danych w standardzie ISO, podstawa najpopularniejszego wariantu protokołu PPP)

HELD *HTTP-Enabled Location Delivery* (protokół dostarczania informacji LCI, używający stosu protokołów HTTP/TCP/IP)

HIP *Host Identity Protocol* (protokół identyfikacji hostów — służąca do celów badawczych architektura protokołu koncentrująca się na mobilności i bezpieczeństwie)

HMAC *Hash-based Message Authentication Code* (kod uwierzytelniania wiadomości z wmieszanym kluczem tajnym — szczególny sposób wykorzystania algorytmów haszujących w tworzeniu kodu MAC)

HoA *Home Address* (adres domowy — w protokole MIP adres węzła mobilnego z jego sieci domowej)

HOPOPT *IPv6 Hop-by-Hop Option* (opcja tranzytowa protokołu IPv6 — typ opcji protokołu IPv6 odnoszący się do każdego przeskoku w ścieżce)

HoT *Home Test* (test adresu domowego — w sprawdzeniu routowalności odwrotnej komunikat wysłany do węzła mobilnego za pośrednictwem agenta HA, w wyniku którego MN uzyskuje część klucza używanego do zabezpieczenia aktualizacji powiązania BU)

HoTI *Home Test Init* (inicjacja testu adresu domowego — w sprawdzeniu routowalności zwrotnej powoduje wysłanie komunikatu HoT przez odbiorcę)

HSPA *High-Speed Packet Access* (szybka transmisja pakietowa — standard 3GPP odnoszący się do bezprzewodowych połączeń szerokopasmowych; wyewoluował z technologii WCDMA)

HSTCP *Highspeed TCP* (szybkie TCP — „skalowalny" wariant protokołu TCP, w którym okno CWND jest korygowane częściowo na podstawie swojej aktualnej wartości; zaprojektowany do efektywniejszego działania w środowiskach o dużej pojemności)

HT *High Throughput* (wysoka przepustowość — wyższe szybkości transmisji związane ze standardem IEEE 802.11n)

HTML *Hyper-Text Markup Language* (hipertekstowy język znaczników — podstawowy język systemu WWW)

HTTP *Hyper-Text Transfer Protocol* (protokół przesyłania dokumentów hipertekstowych — podstawowy protokół systemu WWW; często przesyła dokumenty HTML)

HTTPMU *HTTP using UDP* (metoda przesyłania ruchu HTTP przy wykorzystaniu protokołu UDP z adresowaniem multicastowym; używany do przesyłania komunikatów SSDP w architekturze UPnP)

HTTPS *HTTP over SSL/TLS* (standard bezpiecznej wymiany informacji WWW)

HWRP *Hybrid Wireless Routing Protocol* (hybrydowy protokół routingu bezprzewodowego — protokół routingu proponowany dla standardu IEEE 802.11s)

IA *Identity Association* (skojarzenie tożsamości — w protokole DHCP zbiór adresów)

IAB *Internet Architecture Board* (rada ds. architektury Internetu — jedno z ciał zarządzających IETF; odpowiedzialna za nadzór nad architekturą i ustanawianie związków z innymi organizacjami SDO)

IAID *IA Identifier* (identyfikator IA — w protokole DHCP ID odnoszące się do konkretnego skojarzenia IA)

IANA *Internet Assigned Numbers Authority* (organizacja utrzymująca numery protokołów i wartości pól)

IBSS *Independent Basic Service Set* (niezależny zbiór podstawowych usług — sieć ad hoc z rodziny 802.11)

ICANN *Internet Corporation for Assigned Names and Numbers* (Internetowa Korporacja ds. Nadawania Nazw i Numerów — ciało zarządzające typu non profit ds. nazw domen i związanej z tym strategii)

ICE *Interactive Connectivity Establishment* (ustanawianie łączności interaktywnej — struktura ramowa dla wykonywania trawersowania NAT, która obejmuje podejmowanie prób połączeń bezpośrednich, użycie mechanizmu STUN i w końcu mechanizmu TURN w celu umożliwienia komunikacji przy obecności urządzeń NAT)

ICMP *Internet Control Message Protocol* (internetowy protokół komunikatów kontrolnych — protokół przekazywania informacji i raportowania błędów uważany za część protokołu IP)

ICS *Internet Connection Sharing* (współdzielenie połączeń internetowych — alternatywna nazwa mechanizmu NAT; używana w systemach Windows Microsoftu)

ICV *Integrity Check Value* (wartość kontrolna integralności — wartość używana do sprawdzenia integralności komunikatu, np. skrót kryptograficzny)

ID *Identification* (identyfikacja — w protokole IKE ładunek użyteczny wskazujący tożsamość nadawcy)

IDN *Internationalized Domain Name* (umiędzynarodowiona nazwa domeny — nazwa domeny zawierająca znaki spoza kodu ASCII)

IEEE *Institute of Electrical and Electronics Engineers* (Instytut Inżynierów Elektryków i Elektroników — organizacja SDO ds. protokołów warstwy łącza danych i nie tylko)

IESG *Internet Engineering Steering Group* (internetowa grupa nadzorcza ds. technicznych — ciało zarządzające IETF z kompetencją zatwierdzania dokumentów RFC)

IETF *Internet Engineering Task Force* (internetowy zespół ds. technicznych — organizacja SDO ds. standardów w Internecie)

IGD, IGDDC *Internet Gateway Device/Discovery and Control* (odnajdowanie urządzeń bramy internetowej i sterowanie nimi — protokół technologii UPnP służący do odnajdowania i konfigurowania urządzeń bramy internetowej, takich jak domowe routery NAT)

IGMP *Internet Group Message Protocol*, *Internet Group Management Protocol* (internetowy protokół zarządzania grupami — protokół służący do zarządzania grupami rozsyłania grupowego IPv4; używany przez routery oraz hosty końcowe)

IHL *Internet Header Length* (długość nagłówka internetowego — pole nagłówka IPv4 zawierające długość nagłówka wyrażoną w słowach 32-bajtowych)

IID *Interface Identifier* (identyfikator interfejsu — liczbowy identyfikator, zwykle oparty na adresie MAC; używany przy wyborze adresów IPv6, ale nieużywany do tego celu wtedy, gdy włączone są rozszerzenia prywatności)

IKE *Internet Key Exchange* (internetowa wymiana kluczy — część protokołu IPsec; protokół służący do dynamicznego ustanawiania skojarzeń bezpieczeństwa obejmujących klucze i parametry operacyjne)

IMAP *Internet Message Access Protocol* (internetowy protokół dostępu do wiadomości — używany do pobierania nagłówków poczty e-mail i samych wiadomości z serwerów)

IMAPS *IMAP over SSL/TLS* (bezpieczny protokół pobierania poczty e-mail, obsługiwany przez większość programów pocztowych)

IN *Internet* (w systemie DNS nazwa klasy wskazująca na informację dotyczącą Internetu)

IND *Inverse Neighbor Discovery* (odwrotne odkrywanie sąsiada — dostarcza podobnej funkcji dla IPv6 jak protokół RARP dla IPv4)

IP *Internet Protocol* (protokół internetowy — standardowy pakietowy protokół internetowy, dostarczający dane zgodnie z zasadą najlepszego starania, implementujący wspólny abstrakcyjny datagram w każdej sieci warstwy łącza danych)

IPCP *IP Control Protocol* (protokół kontroli IP — w protokole PPP wariant protokołu NCP używany do konfigurowania łącza sieci IPv4)

IPG *Inter-Packet Gap* (przerwa między pakietami — minimalny odstęp między ramkami w protokole MAC)

IPsec *IP Security* (bezpieczeństwo IP — struktura ramowa zabezpieczeń ruchu IP, obejmująca protokoły IKE, AH i ESP)

IPV6CP *IPv6 Control Protocol* (protokół kontroli IPv6 — w protokole PPP wariant protokołu NCP używany do konfigurowania łącza sieci IPv6)

IRIS *Internet Registry Information Service* (usługa informacyjna rejestrów internetowych — baza danych zawierającą informacje dotyczące zakresów adresów, związanych z nimi numerów AS, danych kontaktowych i serwerów nazw)

IRTF *Internet Research Task Force* (zespół ds. badań nad Internetem — grupy badawcze stowarzyszone z IETF poprzez IAB)

ISAKMP *Internet Security Association and Key Management Protocol* (internetowy protokół skojarzeń bezpieczeństwa i zarządzania kluczami — w protokole IPsec protokół ustanawiania skojarzeń bezpieczeństwa, poprzedzający IKE)

ISATAP *Intra-Site Automatic Tunnel Addressing Protocol* (protokół lokalnego automatycznego adresowania tuneli — technologia automatycznego tunelowania pakietów IPv6 w IPv4, obsługiwana przez Microsoft)

ISDN *Integrated Services Digital Network* (sieć cyfrowa z integracją usług — połączenie usług przesyłania danych: z przełączaniem obwodów i z przełączaniem pakietów)

IS-IS *Intermediate System to Intermediate System* (system pośredni-system pośredni — protokół routingu typu stan łącza oparty na standardzie ISO)

ISL Cisco's *Inter-Switch Link* (protokół łącza między przełącznikami — należący do firmy Cisco protokół przekazywania informacji o sieci VLAN między przełącznikami)

ISM *Industrial, Scientific, and Medical* ([zastosowania] przemysłowe, naukowe i medyczne — pasma częstotliwości wolne od licencji w dużej części świata, wykorzystywane przez Wi-Fi)

ISN *Initial Sequence Number* (początkowy numer sekwencyjny — w protokole TCP pierwszy numer sekwencyjny w połączeniu; przypisany do żądania SYN)

ISO *International Organization for Standardization* (Międzynarodowa Organizacja Normalizacyjna — organizacja SDO odpowiedzialna za definiowanie różnych protokołów i sposobów kodowania, które miały niegdyś zastąpić TCP/IP)

ISOC *Internet Society* (towarzystwo internetowe — korporacja non profit, przewodząca w zakresie standardów internetowych)

ISP *Internet Service Provider* (dostawca usługi internetowej — jednostka, często firma, która przydziela adresy, dostarcza usługi DNS i routingu oraz współpracuje z innymi jednostkami ISP)

ITU *International Telecommunications Union* (Międzynarodowy Związek Telekomunikacyjny — organizacja SDO ds. standardów radiowych i telefonicznych)

ITU-T ITU *Telecommunication Standardization Sector* (Sektor Normalizacji Telekomunikacji ITU — dawniej CCITT; jeden z trzech „sektorów" ITU odpowiedzialny za standardy i „rekomendacje", takie jak ASN.1, X.25, DSL)

IW *Initial Window* (okno początkowe — w protokole TCP początkowa wartość okna CWND)

IXFR *Incremental Zone Transfer* (przyrostowy transfer strefy — przyrostowa wymiana informacji dotyczącej strefy DNS, korzysta z protokołu TCP)

KE *Key Exchange* (wymiana kluczy — w protokole IKE ładunek użyteczny służący do ustanowienia kluczy; na ogół korzysta z protokołu DH)

KSK *Key Signing Key* (klucz do podpisywania kluczy — klucz używany w DNSSEC do podpisywania innych kluczy; na ogół ma ustawiony bit SEP)

L2TP *Layer 2 Tunneling Protocol* (protokół tunelowania warstwy 2. — protokół tunelowania warstwy łącza danych będący standardem IETF)

LACP *Link Aggregation Control Protocol* (protokół sterowania agregacją łączy — część standardu IEEE 802.1AX dotycząca zarządzania zagregowanymi łączami)

LAG *Link Aggregation Group* (grupa agregacji łączy — zbiór łączy działających razem jako jedno wirtualne łącze wyższej wydajności)

LAN *Local Area Network* (sieć lokalna — sieć w obrębie małego obszaru geograficznego, takiego jak pojedyncza siedziba, biuro lub dom)

LCG *Linear Congruential Generator* (liniowy generator kongruencyjny — deterministyczny typ popularnego generatora liczb pseudolosowych, który nie jest generatorem CSPRNG)

LCI *Location Configuration Information* (dane reprezentujące położenie systemu — geograficzne lub administracyjne)

LCI *Logical Channel Identifier* (identyfikator kanału logicznego — w przełączaniu obwodów identyfikator dla wirtualnego kanału)

LCN *Logical Channel Number* (numer kanału logicznego — w przełączaniu obwodów numer wirtualnego kanału)

LCP *Link Control Protocol* (protokół kontroli łącza — w protokole PPP używany do ustanowienia łącza)

LDAP *Lightweight Directory Access Protocol* (lekki protokół usług katalogowych — protokół wyszukiwania oparty na protokole DAP standardu ISO X.500)

LDRA *Lightweight DHCP Relay Agent* (lekki agent przekazywania DHCP — mechanizmy umożliwiające urządzeniom warstwy 2. pełnienie roli agentów przekazywania DHCP)

LEAP *Lightweight Extensible Authentication Protocol* (lekki rozszerzalny protokół uwierzytelniania — metoda EAP firmy Cisco używająca kluczy WEP lub TKIP)

LLA *Link Layer Address* (adres warstwy łącza danych — w protokole FMIPv6 opcja nagłówka mobilności służąca do wskazania adresu warstwy łącza danych)

LLC *Logical Link Control* (sterowanie połączeniem logicznym — podwarstwa warstwy łącza danych związana ze sterowaniem łączem)

LLMNR *Link Local Multicast Name Resolution* (lokalne dla łącza rozpoznawanie nazw, oparte na rozgłaszaniu grupowym — oparty na multiemisji wariant systemu DNS,

zaprojektowany do stosowania w ramach łącza, który używa innego numeru portu niż DNS; stosowany do lokalnego odkrywania usług i węzłów)

LMQI *Last Member Query Interval* (w protokołach IGMP i MLD czas między komunikatami „group-specific query")

LMQT *Last Member Query Time* (w protokołach IGMP i MLD całkowity czas „stracony" po wysłaniu komunikatu „last member query" i możliwych transmisjach; reprezentuje opóźnienie związane z opuszczeniem grupy przez host — „leave latency")

LNP *Local Network Protection* (ochrona sieci lokalnej — zbiór technik sugerowanych do wykorzystania we wdrożeniach IPv6, które sprawiają, że niepotrzebne staje się stosowanie urządzeń NAT)

LoST *Location-to-Service Translation* (system lokalizacji usług — struktura ramowa służąca do oferowania usług opartych na lokalizacji — np. wskazanie najbliższego szpitala)

LQR *Link Quality Reports* (raporty jakości łącza — w protokole PPP raporty dotyczące pomiarów jakości łącza, zawierające liczby pakietów: odebranych, wysłanych i odrzuconych ze względu na błędy)

LTE *Long-Term Evolution* (długofalowy rozwój — standard bezprzewodowej szerokopasmowej transmisji danych rozwijany przez konsorcjum 3GPP; ewolucja standardu HSPA)

LW-MLD *Lightweight MLD* (lekkie MLD — wariant protokołu MLD z prostszą semantyką dołączania lub opuszczania grupy multiemisji)

MAC *Media Access Control* (kontrola dostępu do nośnika — zasady negocjowania dostępu do współdzielonego nośnika sieciowego, zazwyczaj część protokołu warstwy łącza danych)

MAC *Message Authentication Code* (kod uwierzytelniania wiadomości — funkcja matematyczna używana jako pomoc w weryfikacji integralności wiadomości)

MAN *Metropolitan Area Network* (sieć metropolitarna — sieć obejmująca umiarkowany obszar geograficzny, taki jak duże miasto lub region)

MCS *Modulation and Coding Scheme* (schemat modulacji i kodowania — połączenie modulacji i kodowania, wiele kombinacji jest dostępnych w standardzie 802.11n)

MD *Message Digest* Algorithms (algorytmy skrótu wiadomości — funkcje matematyczne dające krótki liczbowy „odcisk palca" dla większej wiadomości)

mDNS *Multicast DNS* (DNS na bazie multiemisji — lokalny wariant usługi rozpoznawania nazw opracowany przez firmę Apple)

MIH *Media-Independent Handoff* (przekazywanie niezależne od nośnika — mechanizmy obsługujące zmianę punktu dołączenia do sieci między sieciami heterogenicznymi; standard IEEE 802.21 obejmuje MIH dla następujących typów sieci: 802.3, 802.11, 802.15, 802.16, 3GPP i 3GPP2)

MII *Media-Independent Interface* (interfejs niezależny od nośnika — w obszarze sprzętu interfejs między implementacją MAC a implementacją protokołu warstwy fizycznej, który jest niezależny od warstwy fizycznej)

MIME *Multipurpose Internet Mail Extensions* (uniwersalne rozszerzenia poczty internetowej — metoda etykietowania i kodowania różnych typów obiektów w poczcie elektronicznej)

MIMO *Multiple Input, Multiple Output* (wielokrotne wejście, wielokrotne wyjście — system antenowy w sieci bezprzewodowej z wieloma antenami, dający wyższą wydajność niż systemy z pojedynczą anteną, ale wymagający bardziej złożonego przetwarzania sygnału)

MIP *Mobile IP* (mobilne IP — rozszerzenia dotyczące adresowania i routingu służące do obsługi zmiany położenia punktu dołączenia do sieci bez zmiany adresu)

MITM *Man-in-the-Middle attack* (atak typu człowiek pośrodku — typowa forma ataku MSM, przeprowadzana przez ukrytego pośrednika)

MLD *Multicast Listener Discovery* (odkrywanie odbiorców multiemisji — protokół używany przez routery IPv6 do wykrywania odbiorców multiemisji w łączu; oferuje funkcjonalność podobną do funkcjonalności protokołu IGMP dla IPv4)

MLPP *Multilevel Precedence and Preemption* (wielopoziomowy system pierwszeństwa i wywłaszczania — stosowany w telefonii system priorytetyzacji rozmów — np. do użytku militarnego)

MMRP *Multiple MAC Registration Protocol* (protokół wielokrotnej rejestracji adresów grupowych MAC — część protokołu MRP używana do rejestracji zainteresowania multiemisją)

MN *Mobile Node* (węzeł mobilny — ruchomy węzeł w scenariuszu MIP)

MOBIKE *Mobile version of IKE* (rozszerzenia protokołu IKE obsługujące mobilność i zmiany informacji adresowych)

MODP *Modulo-P groups* (grupy modulo P — grupy oparte na arytmetyce modularnej, w rozumieniu algebry abstrakcyjnej, wykorzystywane w protokołach ustanawiania kluczy)

MoS *Mobility Services* (usługi mobilności — część standardu IEEE 802.21 obsługująca usługi przekazywania niezależnego od nośnika)

MP *Mesh Point* (punkt kratowy — nazwa węzła IEEE 802.11s działającego w konfiguracji kratowej)

MP, MPPP, MLP, MLPPP *Multi-link PPP* (wielołączowe PPP — użycie protokołu PPP obejmujące wiele łączy jednocześnie)

MPDU *MAC Protocol Data Unit* (jednostka danych protokołu MAC — nazwa ramki używana w standardach 802.11)

MPE *Manchester Phase Encoding* (kodowanie fazowe Manchester — system kodowania bitów, w którym przejście między poziomami napięcia wskazuje jeden bit)

MPLS *Multi-Protocol Label Switching* (wieloprotokołowe przełączanie etykiet — architektura, która przełącza ramki na podstawie wartości etykiet, a nie adresów IP)

MPPC *Microsoft's Point-to-Point Compression* (metoda kompresji dla połączeniu dwupunktowego firmy Microsoft — używana w protokole PPP)

MPPE *Microsoft's Point-to-Point Encryption* (metoda szyfrowania dla połączenia dwupunktowego firmy Microsoft — używana w protokole PPP)

MPV *Maximum Pad Value* (maksymalna wartość dopełnienia — w protokole PPP maksymalna liczba bajtów dopełnienia)

MRD *Multicast Router Discovery* (protokół odkrywania sąsiednich, bezpośrednio dostępnych routerów multiemisyjnych)

MRP *Multiple Registration Protocol* (protokół wielokrotnej rejestracji — standard IEEE 802.1ak dotyczący rejestrowania atrybutów)

MRRU *Multilink Maximum Received Reconstructed Unit* (maksymalna zrekonstruowana jednostka odbioru dla łącza wielokrotnego — jednostka MRU po rekonstrukcji z części odebranych za pośrednictwem wielu łączy MP)

MRU *Maximum Receive Unit* (maksymalna jednostka odbioru — rozmiar największego pakietu lub komunikatu, który zostanie przyjęty przez odbiorcę)

MS-CHAP *Microsoft's Challenge-Handshake Authentication Protocol* (protokół uwierzytelniania wezwanie-uzgodnienie firmy Microsoft — protokół uwierzytelniania obejmujący wezwanie, odpowiedź i sprawdzenie odpowiedzi, występujący w dwóch wersjach: MS-CHAPv1 i MS-CHAPv2)

MSDU *MAC Services Data Unit* (jednostka danych usługi MAC — typ ramki 802.11 dostępny dla warstw powyżej MAC)

MSK *Master Session Key* (główny klucz sesji — klucz wyprowadzony po sesji EAP przy użyciu metod obsługujących wyprowadzanie kluczy)

MSL *Maximum Segment Lifetime* (maksymalny czas życia segmentu — w protokole TCP maksymalny czas, przez jaki segment może istnieć w sieci przed uznaniem go za nieważny)

MSM *Message Stream Modification* (modyfikacja strumienia komunikatów — aktywna modyfikacja komunikatów; zwykle rodzaj ataku)

MSS *Maximum Segment Size* (maksymalny rozmiar segmentu — w protokole TCP rozmiar największego segmentu, jaki odbiorca jest gotów odebrać; zwykle przekazywany w opcji podczas ustanawiania połączenia)

MTU *Maximum Transmission Unit* (maksymalna jednostka transmisji — maksymalny rozmiar ramki, jaką można przesłać przez sieć)

MVRP *Multiple VLAN Registration Protocol* (protokół wielokrotnej rejestracji sieci VLAN — część protokołu MRP używana do rejestrowania sieci VLAN)

MX *Mail Exchanger* (rekord wymiany poczty — rekord zasobów DNS wskazujący porządek pierwszeństwa hostów, które chcą użyć protokołu SMTP do wymiany poczty)

NAC *Network Access Control* (kontrola dostępu do sieci — proces wykorzystywany do ustalenia, czy urządzenie powinno otrzymać prawa dostępu do używania sieci)

NACK *Negative Acknowledgment* (potwierdzenie negatywne — wskazanie braku odbioru lub braku akceptacji)

NAP *Network Access Protection* (ochrona dostępu do sieci — wariant NAC firmy Microsoft; po raz pierwszy dostępny w systemie Windows Server 2008)

NAPT *NAT with Port Translation* (NAT z translacją portów — NAT z podmianą numeru portu, najbardziej rozpowszechniona forma NAT)

NAPTR *Name Authority Pointer* (autorytatywny wskaźnik nazw — rekord zasobów DNS używany w systemie DDDS opartym na DNS do przechowywania reguł przepisywania)

NAR *New Access Router* (nowy router dostępowy — w protokole FMIPv6 router, który przypuszczalnie zostanie wkrótce użyty)

NAT *Network Address Translation* (translacja adresów sieciowych — mechanizm zamiany adresów w datagramach IP; używany przede wszystkim do zmniejszenia zużycia globalnie routowalnych adresów IP; zazwyczaj używany w połączeniu z prywatnymi adresami IP; dostarcza również pewnego rodzaju funkcjonalności zapory sieciowej)

NAT64 *IPv6/IPv4 NAT* (mechanizm NAT, który wykonuje translację adresów między protokołami IPv4/ICMPv4 a IPv6/ICMPv6 i na odwrót; proponowany do zapewnienia koegzystencji i interoperacyjności między IPv6 i IPv4)

NAT-PMP *NAT Port Mapping Protocol* (protokół mapowania portów dla urządzeń NAT — alternatywa dla protokołu IGD, opracowana przez firmę Apple i służąca do konfigurowania pewnych urządzeń NAT; dostarcza możliwości zdalnej konfiguracji przekazywania numerów portów)

NAT-PT *NAT with Protocol Translation* (NAT z translacją protokołów — aktualnie niezalecane podejście do translacji IPv4/IPv6)

NAV *Network Allocation Vector* (wektor przydziału sieci — opóźnienie czasowe przed transmisją ze względu na używanie kanału przez inne stacje, występujące w funkcji DCF standardu 802.11)

NBMA *Non-Broadcast Multiple Access* (wielodostęp bez rozgłaszania — sieci z wieloma użytkownikami pozbawione możliwości rozgłaszania)

NCoA *New Care-of Address* (nowy adres pośredni — w protokole FMIPv6 adres CoA, który zostanie uzyskany od routera NAR)

NCP *Network Control Protocol* (protokół sterowania siecią — w protokole PPP używany do ustalenia protokołu warstwy sieci)

ND, NDP *Neighbor Discovery* (odkrywanie sąsiada — stosowana w protokole IPv6 metoda odkrywania i uzyskiwania adresu MAC sąsiadów w tym samym łączu; działa podobnie do protokołu ARP; zaimplementowana jako część protokołu ICMPv6)

NEMO *Network Mobility* (mobilność sieci — mobilność, w której router i sieć zmieniają punkt dołączenia)

NIC *Network Interface Card* (karta sieciowa — urządzenie stanowiące interfejs między komputerem a siecią)

NONCE *number used once* (identyfikator jednorazowy — wartość losowa używana w wielu protokołach kryptograficznych do zwalczania ataków przez powtórzenie)

NPT66 *IPv6-to-IPv6 NAPT* (NAT z algorytmiczną translacją adresów i portów)

NRO *Number Resource Organization* (organizacja zasobów numerów — *Address Supporting Organization*, organizacja obsługi adresów, afiliowana do ICANN)

NS *Name Server* (serwer nazw — rekord zasobów DNS zawierający nazwę innego serwera nazw)

NS *Neighbor Solicitation* (zapytanie o adres sprzętowy sąsiada — część metody ND protokołu iPv6; podobne do zapytania protokołu ARP związanego z IPv4, ale używa adresowania grupowego protokołu IPv6; zaimplementowane przy użyciu protokołu ICMPv6)

NSCD *Name Services Cache Daemon* (demon buforowania usług nazw — proces dostarczający buforowanie dla DNS i innych systemów rozpoznawania nazw popularnych w systemach UNIX)

NSEC *Next Secure* (rekord zasobów DNS używany w DNSSEC do wskazania następnego rekordu zasobów w uporządkowanej liście; wykorzystywany do uwierzytelnionego zaprzeczenia istnienia)

NSEC3 *Next Secure (version 3)* (rekord zasobów DNS podobny do NSEC, ale zawierający funkcję skrótu w celu odparcia ataku przez wyliczenie nazw)

NSEC3PARAM *NSEC Parameters* (parametry NSEC — rekord zasobów DNS używany w DNSSEC, przechowujący parametry funkcji skrótu rekordu NSEC3)

NTN *Non-Terminal NAPTR* (nieterminalny NAPTR — wskaźnik NAPTR, który wskazuje inną domenę z rekordami)

NTP *Network Time Protocol* (sieciowy protokół czasu — protokół służący do synchronizacji zegarów)

NUD *Neighbor Unreachability Detection* (wykrywanie niedostępności sąsiada — w metodzie ND protokołu IPv6 służy do ustalenia, czy sąsiad jest nadal dostępny)

OCSP *Online Certificate Status Protocol* (protokół aktualnego stanu certyfikatów — protokół sprawdzania ważności certyfikatu; alternatywa dla przeszukiwania list CRL)

OFDM *Orthogonal Frequency Division Multiplexing* (multipleksowanie z ortogonalnym podziałem częstotliwości — skomplikowany system modulacji, w którym podnośne wielu częstotliwości są jednocześnie modulowane w określonym paśmie w celu osiągnięcia wysokiej przepustowości; wykorzystywane w DSL, 802.11a/g/n, 802.16e i zaawansowanych standardach przesyłu danych telefonii komórkowej, w tym w LTE)

OID *Object Identifier* (identyfikator dokumentu elektronicznego — numeryczny identyfikator dokumentu elektronicznego; stosowany w kodowaniu certyfikatów)

OLSR *Optimized Link State Routing* (zoptymalizowany routing z użyciem stanów połączeń — standardowy protokół dla routingu na żądanie w sieciach ad hoc)

OOB *Out Of Band* (poza pasmem — informacja dostarczana poza głównym kanałem komunikacyjnym)

ORO *Option Request Option* (opcja żądania opcji — w protokole DHCP opcja wskazująca zainteresowanie systemu wiedzą, które opcje są obsługiwane)

OSI *Open System Interconnect* (łączenie systemów otwartych — abstrakcyjny model odniesienia zdefiniowany przez ISO dla systemów otwartych, który pomógł w kształtowaniu podstaw projektowania warstwowego w dziedzinie protokołów)

OUI *Organizationally Unique Identifier* (unikalny identyfikator organizacji — format oryginalnego prefiksu adresu warstwy MAC zdefiniowany przez IEEE)

P2P *Peer-to-Peer* (sieć równorzędna — uczestniczące systemy są zarówno klientami, jak i serwerami)

PA *Provider-Aggregatable* (agregowalna w ramach dostawcy — przestrzeń adresów IP, w której prefiks klienta jest nadany przez jego dostawcę usługi Internetu)

PAA *PANA Authentication Agent* (agent uwierzytelniający PANA — agent protokołu PANA wykonujący uwierzytelnianie, takie jak serwer AAA)

PaC *PANA Client* (klient PANA — agent protokołu PANA zgłaszający żądanie uwierzytelnienia)

PAD *Peer Authentication Database* (baza danych uwierzytelniania węzłów równorzędnych — w protokole IPsec abstrakcja bazy danych zawierająca informacje dotyczące uwierzytelniania dla każdego węzła równorzędnego, takie jak używanie IKE lub PSK i odnośne dane uwierzytelniające)

PANA *Protocol for Carrying Authentication for Network Access* (protokół transportu uwierzytelnienia dostępu do sieci — oparty na UDP/IP nośnik dla protokołu EAP)

PAP *Password Authentication Protocol* (protokół uwierzytelniania za pomocą hasła — protokół, który przesyła niezaszyfrowane hasło; nieodporny na ataki MITM i podsłuchiwanie)

PAWS *Protection Against Wrapped Sequence Numbers* (ochrona przed przepełnieniem numeru sekwencyjnego — w protokole TCP metoda wykorzystująca wartości TSOPT w celu wychwycenia powtarzania się numerów sekwencyjnych po przepełnieniu pola numeru sekwencyjnego)

PCF *Point Coordinating Function* (funkcja koordynacji punktowej — protokół MAC dla sieci 802.11 stosujący kombinację dostępu do nośnika opartego na rywalizacji i dostępu do nośnika nieograniczonego zasadami konkurencji; nieużywany w szerokim zakresie)

PCO *Phased Coexistence Operation* (działanie w trybie okresowej koegzystencji — metoda przełączania szerokości kanału stosowana przez punkty dostępowe sieci 802.11 w celu zmniejszenia negatywnego wpływu na działanie starszego sprzętu)

PCoA *Previous Care-of Address* (poprzedni adres pośredni — w protokole FMIPv6 bieżący lub poprzedni adres CoA uzyskany od routera PAR)

PCP *Port Control Protocol* (protokół kontroli portów — projekt protokołu IETF współczesnej generacji dotyczący konfigurowania urządzeń NAT, w tym SPNAT i NAT64)

PDU *Protocol Data Unit* (jednostka danych protokołu — opisuje komunikat na poziomie wybranej warstwy protokołów; czasem nieformalnie używana zamiennie z takimi terminami jak pakiet, ramka, datagram, segment lub komunikat)

PEAP *Protected Extensible Authentication Protocol* (chroniony rozszerzalny protokół uwierzytelniania — popularna metoda enkapsulacji protokołu EAP w TLS; podobny do protokołu EAP-TTLS)

PEN *Private Enterprise Number* (prywatny numer przedsiębiorstwa — numery przydzielane przez IANA używane przez przedsiębiorstwa przy tworzeniu identyfikatorów obiektów OID)

PFC *Protocol Field Compression* (kompresja pola *Protokół* — w protokole PPP wyeliminowanie pola *Protokół* w celu zmniejszenia narzutu)

PFS *Perfect Forward Secrecy* (doskonała poufność w przód — w kryptografii klucza publicznego właściwość, dzięki której naruszenie bezpieczeństwa jednego klucza prowadzi co najwyżej do naruszenia bezpieczeństwa danych zaszyfrowanych tym właśnie kluczem, nie zagrażając innym danym lub kluczom)

PHB *Per-Hop Behavior* (abstrakcyjny sposób obsługi pakietów w routerze używany do implementacji DS)

PHY *Physical* (warstwa fizyczna — warstwa w modelu OSI; zazwyczaj opisuje złącza, częstotliwości, sposób kodowania i modulacji)

PI *Provider-Independent* (niezależna od dostawcy — przestrzeń adresów IP posiadana przez klienta; niepochodząca od prefiksu adresu ISP)

PIM *Protocol Independent Multicast* (multiemisja niezależna od protokołu — protokół routingu nielokalnej multiemisji, który może wykorzystać dane i operacje protokołów routingu emisji pojedynczej)

PIO *Prefix Information Option* (opcja informacji o prefiksie — w protokole ICMPv6 opcja zawierająca prefiks adresu IP)

PKC *Public Key Certificate* (certyfikat klucza publicznego — obiekt cyfrowy zawierający klucz publiczny i podpis urzędu certyfikacji razem z różnymi zasadami użycia i parametrami)

PKCS *Public Key Cryptography Standards* (standardy kryptografii klucza publicznego — metody kodowania i przedstawiania klucza publicznego oraz powiązanego materiału)

PKI *Public Key Infrastructure* (infrastruktura klucza publicznego — system zarządzania kluczami publicznymi i ich dystrybucją)

PLCP *Physical Layer Convergence Procedure* (procedura konwergencji warstwy fizycznej — metoda standardu 802.11 dotycząca kodowania i ustalenia typu ramki i parametrów transmisji radiowej)

PMTU *Path MTU* (MTU ścieżki — najmniejsze MTU we wszystkich łączach ścieżki od nadawcy do odbiorcy)

PMTUD *PMTU Discovery* (odkrywanie PMTU — proces ustalania wartości PMTU; zwykle polega na komunikatach PTB protokołu ICMP)

PNAC *Port-Based NAC* (NAC oparte na porcie — wersja NAC, w której w podejmowaniu decyzji autoryzacji wykorzystywany jest fizyczny port dołączenia do sieci)

PoE *Power over Ethernet* (zasilanie przez Ethernet — przesyłanie energii zasilającej urządzenia przy wykorzystaniu okablowania Ethernetu)

POTS *Plain Old Telephone Service* (zwykła tradycyjna telefonia — konwencjonalna analogowa usługa telefoniczna)

PPP *Point-to-Point Protocol* (protokół typu punkt-punkt — protokół konfiguracyjny i kapsułkujący dane warstwy łącza danych zdolny do obsługi wielu protokołów warstwy sieciowej i wykorzystujący wiele leżących poniżej łączy fizycznych)

PPPoE *PPP over Ethernet* (PPP nad Ethernetem — metody służące do ustanowienia skojarzenia PPP na bazie łącza ethernetowego)

PPTP *Point-to-Point Tunneling Protocol* (protokół tunelowania typu punkt-punkt — protokół tunelowania warstwy łącza danych firmy Microsoft)

PRF *Pseudo-Random Function Family* (rodzina funkcji pseudolosowych — zbiór funkcji, które nie mogą być odróżnione od prawdziwie losowych funkcji przy użyciu algorytmu wielomianowego; termin używany także czasem mniej formalnie w odniesieniu do pojedynczej takiej funkcji)

PRNG, PRG *Pseudo-Random Generator* (generator liczb pseudolosowych — funkcja matematyczna używana do wygenerowania ciągu pojawiających się losowo wartości)

PSK *Pre-Shared Key* (klucz współdzielony — wstępne rozdzielenie kluczy szyfrowania; nie jest używany żaden protokół dynamicznej wymiany kluczy)

PSM *Power Save Mode* (tryb oszczędzania energii — tryb standardu 802.11, w którym urządzenia mogą znajdować się w stanie „uśpienia", kiedy nie są aktywne, a w późniejszym czasie wykonać sondowanie w celu otrzymania dotyczących ich informacji od punktu dostępowego)

PSMP *Power-Save Multi-Poll* (dwukierunkowa wersja techniki APSD, część standardu 802.11n)

PTB *Packet Too Big* (za duży pakiet — ogólna nazwa obejmująca komunikat protokołu ICMP *Destination Unreachable Fragmentation Required* oraz komunikat protokołu IPv6 *Packet Too Big*, które wskazują, że pakiet jest za duży w stosunku do rozmiaru MTU następnego przeskoku)

QAM *Quadrature Amplitude Modulation* (kwadraturowa modulacja amplitudowo-fazowa — kombinacja modulacji amplitudy i fazy)

QBSS *QoS BSS* (grupa BSS sieci 802.11 wzbogacona o mechanizmy QoS standardu 802.11e lub 802.11n)

QI *Query Interval* (interwał zapytań — w protokołach IGMP i MLD czas między zapytaniami ogólnymi)

QoS *Quality of Service* (jakość usługi — ogólny termin opisujący, w jaki sposób może być różnicowana obsługa ruchu sieciowego, zwykle pod względem mniejszego lub większego opóźnienia lub kolejności odrzucania, na podstawie parametrów konfiguracyjnych)

QPSK *Quadrature Phase Shift Keying* (kwadraturowe kluczowanie z przesuwem fazy — zazwyczaj modulowanie dwóch bitów na symbol przy użyciu czterech faz sygnału, chociaż są możliwe bardziej zaawansowane wersje, z większą ilością bitów na symbol)

QQI *Querier's Query Interval* (interwał zapytań routera odpytującego — w protokołach IGMP i MLD czas między wysyłaniem komunikatów z zapytaniem ogólnym; aktualnie nieodpytujące routery multiemisji przyjmują ostatnio odebraną wartość QQI jako swoją wartość QI)

QQIC *Querier's Query Interval Code* (kod interwału zapytań routera odpytującego — w komunikatach protokołu IGMP i MLD kod wartości QQI)

QRI *Query Response Interval* (okres odpowiedzi na zapytanie — w protokołach IGMP i MLD maksymalna ilość czasu, w jakim odbiorcy wolno wysłać odpowiedź na zapytanie)

QRV *Querier Robustness Variable* (zmienna niezawodności routera odpytującego — w protokołach IGMP i MLD ustala liczbę retransmisji)

QS *Quick Start* (szybki start — w protokole TCP eksperymentalna modyfikacja prowadząca do szybszego działania w fazie początkowej, pod warunkiem uzyskania zgody urządzeń znajdujących się w ścieżce)

QSTA *QoS STA* (stacja w sieci 802.11 obsługująca mechanizm QoS)

RA *Router Advertisement* (ogłoszenie routera — komunikat wskazujący obecność sąsiadującego routera w sieci lokalnej; używa protokołu ICMP)

RADIUS *Remote Authentication Dial-In User Service* (usługa zdalnego uwierzytelniania wdzwaniających się użytkowników — popularny protokół przesyłania danych AAA)

RAIO *Relay Agent Information Option* (opcja informacji agenta przekazywania — w protokole DHCPv6 opcja wykorzystywana przez agenty przekazywania do umieszczania różnych informacji)

RARP *Reverse ARP* (odwrotne ARP — protokół obsługujący określanie adresów sieciowych na podstawie adresów warstwy MAC)

RAS *Remote Access Server* (serwer zdalnego dostępu — serwer, który obsługuje zdalnych użytkowników — uwierzytelnianie, kontrola dostępu itd.)

RC4 *Rivest Cipher #4* (szyfr Rivesta, wersja 4. — popularna metoda szyfrowania z kluczem symetrycznym opracowana przez Rona Rivesta)

RD *Router Discovery* (odkrywanie routerów — procedura lokalizacji najbliższego routera; używa protokołu ICMP)

RDATA *Returned Data* (zwrócone dane — część protokołu DNS służąca do przechowywania zwróconych danych)

RDNSS *Recursive DNS Server* (rekurencyjny serwer DNS — opcja używana w komunikatach RA; wskazuje adres serwera DNS)

RED *Random Early Detection* (losowe wczesne wykrywanie — mechanizm AQM, który zaznacza lub odrzuca pakiety z rosnącym prawdopodobieństwem, kiedy trwałe przeciążenie wydaje się narastać)

RFC *Request for Comments* (prośba o komentarze — dokumenty publikowane przez IETF; niektóre z nich są standardami)

RGMP *Router-port Group Management Protocol* (protokół zarządzania grupami typu router-port — protokół firmy Cisco służący do uaktywniania podsłuchu IGMP)

RH *Routing Header* (nagłówek routingu — nagłówek rozszerzający protokołu IPv6, który zmienia ścieżkę dostarczania ruchu sieciowego)

RHBP *Rate Halving with Bounded Pacing* (redukcja szybkości transmisji o połowę z parametrami ograniczającymi — w protokole TCP przekształcona wersja algorytmu FACK pomagająca w bardziej równomiernym rozłożeniu retransmisji na przestrzeni okresu RTT po wywnioskowanej utracie pakietu)

RIP *Routing Information Protocol* (protokół informowania o trasach — protokół routingu dla małych organizacji; oryginalna wersja nie obsługuje masek podsieci)

RIR *Regional Internet Registry* (regionalny rejestr Internetu — przydziela przestrzeń adresów dla określonego regionu świata)

RO *Route Optimization* (optymalizacja tras — poprawianie tras przez eliminowanie okrężnych, załamujących się ścieżek używanych w prostym protokole MIP)

ROAD *Running Out of Address Space* (wyczerpywanie się przestrzeni adresów — problem motywujący utworzenie protokołu IPv6 i skutkujący utworzeniem metody przydzielania adresów CIDR)

ROHC *Robust Header Compression* (niezawodna kompresja nagłówków — standardy obecnej generacji dotyczące kompresji nagłówków protokołów)

RP *Rendezvous Point* (miejsce spotkań — termin używany w routingu multiemisyjnym do wymiany informacji dotyczących grup)

RPC *Remote Procedure Call* (zdalne wywołanie procedury — struktura umożliwiająca zdalną obsługę wywołań procedur w programie)

RPF *Reverse Path Forwarding* (przekazywanie z braniem pod uwagę odwrotnej ścieżki — aby uniknąć pętli, routery multiemisji wykonują sprawdzenie RPF mające na celu upewnienie się, że datagram multiemisji przychodzi na ten sam interfejs, który odpowiada ścieżce do nadawcy)

RPSL *Routing Policy Specification Language* (język specyfikacji polityki routingu — język używany do formułowania zasad routingu, takich jak ustalenie, do którego systemu autonomicznego należy określony prefiks sieci)

RR *Resource Record* (rekord zasobów — blok informacji określonego typu posiadany przez domenę i dystrybuowany przez system DNS)

RRP, **RR** *Return Routability/Procedure* (procedura routowalności zwrotnej — sprawdzenie używane w protokole MIPv6 w celu upewnienia się, że węzeł mobilny jest autentyczny, zawiera kontrolę HoA i kontrolę CoA)

RRset *Resource Record Set* (zestaw rekordów zasobów — zbiór rekordów zasobów DNS z tą samą nazwą domeny — właściciela rekordu i klasą)

RRSIG *Resource Record Signature* (podpis rekordów zasobów — rekord zasobów DNS używany w DNSSEC zawierający podpis dla zestawu RRset)

RS *Router Solicitation* (żądanie routera — komunikat protokołu ICMP, który powoduje wysłanie odpowiedzi przez router)

RSA *Rabin, Shamir, Adelman* (najpopularniejszy algorytm kryptografii klucza publicznego)

RSN *Robust Security Network* (sieć z mocnym bezpieczeństwem — poprawione bezpieczeństwo w standardzie IEEE 802.11i/WPA; rozwiązanie włączone do standardu 802.11)

RSNA *RSN Association* (skojarzenie RSN — pełne wykorzystanie/implementacja RSN)

RST *Reset* (resetuj — bit nagłówka protokołu TCP i typ segmentu, który powoduje awaryjne przerwanie połączenia TCP)

RSTP *Rapid Spanning Tree Protocol* (szybki protokół drzewa rozpinającego — wersja protokołu STP ze zmniejszonym opóźnieniem)

RTO *Retransmission Timeout* (czas oczekiwania na retransmisję — czas odczekiwania przed retransmisją danych, które uważa się za utracone)

RTS *Request To Send* (żądanie wysyłania — komunikat pokazujący chęć wysłania następnego komunikatu)

RTT *Round Trip Time* (czas podróży w obie strony — minimalny czas, w którym należy się spodziewać odpowiedzi od partnera komunikacji)

RTTM *RTT Measurement* (pomiar RTT — chwilowe oszacowanie czasu RTT)

RTTVAR *RTT Variance* (wariancja RTT — w protokole TCP uśrednione w czasie oszacowanie odchylenia czasu RTT dla połączenia)

RTX *Retransmission* (retransmisja — ponowne wysłanie danych)

RW *Restart Window* (okno restartu — w protokole TCP wartość okna CWND w momencie, gdy TCP wznawia wysyłanie po okresie bezczynności)

SA *Security Association* (skojarzenie bezpieczeństwa — w protokole IPsec stan odnoszący się do jednokierunkowego związku między stronami komunikacji; zawiera uzgodnione klucze, algorytmy itd.; SA może odnosić się do transmisji pojedynczej lub do multiemisji)

SACK *Selective Acknowledgment* (potwierdzenie selektywne — w protokole TCP opcja wskazująca poprawnie odebrane dane poza kolejnością)

SAD *Security Association Database* (baza danych skojarzeń bezpieczeństwa — w protokole IPsec abstrakcja bazy danych zawierającej informacje o każdym aktywnym SA; indeksowana logicznie przez SPI)

SAE *Simultaneous Authentication of Equals* (jednoczesne uwierzytelnianie węzłów równorzędnych — forma uwierzytelniania stosowana w standardzie 802.11s)

SAP *Session Announcement Protocol* (protokół ogłaszania sesji — przesyła eksperymentalne multiemisyjne ogłoszenia sesji; patrz także SDP)

SCSV *Signaling Cipher Suite Value* (sygnalizacyjna wartość zestawu szyfrów — w protokole TLS wartość CS, która zamiast zestawu szyfrów wskazuje konkretny zbiór alternatywnych funkcji lub opcji)

SCTP *Stream Control Transport Protocol* (transportowy protokół sterowania strumieniem — niezawodny protokół transportowy stanowiący alternatywę dla protokołu TCP, który nie wymusza ścisłego uporządkowania i obsługuje wiele podstrumieni oraz zmiany adresów punktów końcowych)

SCVP *Server-Based Certificate Verification Protocol* (oparty na serwerze protokół walidacji certyfikatów — protokół stosujący metody DPD i DPV do obsługi certyfikatów)

SDID *Signing Domain Identifier* (identyfikator domeny podpisującej — w protokole DKIM nazwa domenowa podpisującego)

SDLC *Synchronous Data Link Control* (synchroniczne sterowanie łączem danych — prekursor protokołu HDLC, warstwa łącza danych architektury SNA)

SDO *Standards-Defining Organization* (organizacja definiująca standardy — pojęcie obejmujące IEEE, IETF, ISO, ITU, 3GPP, 3GPP2)

SDP *Session Description Protocol* (protokół opisu sesji — protokół, który opisuje sesje multimedialne)

SEND *Secure Neighbor Discovery* (bezpieczne odkrywanie sąsiadów — bezpieczny wariant ND wykorzystujący adresy CGA)

SEP *Secure Entry Point* (bezpieczny punkt wejściowy — w protokole DNSSEC wskazuje, że rekord zasobów DNSKEY zawiera klucz KSK)

SFD *Start Frame Delimiter* (znacznik początkowy ramki — układ bitów wskazujący początkową część ramki w PDU łącza)

SG *Security Gateway* (brama bezpieczeństwa — w protokole IPsec system końcowy protokołów IPsec, często na brzegu sieci)

SHA *Secure Hash Algorithm* (algorytm bezpiecznego skrótu — jeden ze zbioru algorytmów haszujących nadających się do zapewnienia integralności wiadomości)

SIFS *Short Inter-Frame Space* (krótki odstęp międzyramkowy — najmniejsza ilość czasu między ramką standardu 802.11 a jej potwierdzeniem ACK)

SIIT *Stateless IP/ICMP Translation* (bezstanowa translacja IP/ICMP — struktura ramowa dla translacji między protokołami IPv4 i IPv6, obejmująca specjalne reguły translacji ICMP oraz techniki NAT64 i DNS64)

SIP *Session Initiation Protocol* (protokół inicjowania sesji — ogólny protokół sygnalizacyjny; używany w technologii VoIP)

SLAAC *Stateless Address Autoconfiguration* (bezstanowa automatyczna konfiguracja adresu — mechanizm, za pomocą którego węzeł samodzielnie konfiguruje swój własny adres IP; zwykle ma zastosowanie do węzłów IPv6)

SLLAO *Source Link-Layer Address Option* (opcja właściwego dla warstwy łącza danych adresu źródła — w protokole ICMPv6 opcja zawierająca właściwy dla warstwy łącza danych adres nadawcy)

SMSS *Senders MSS* (MSS nadawcy — wartość MSS dla połączenia z punktu widzenia nadawcy)

SMTP *Simple Mail Transfer Protocol* (prosty protokół transferu poczty — protokół przekazywania poczty elektronicznej między agentami transferu poczty)

SNA *Systems Network Architecture* (architektura sieciowa systemów — architektura sieciowa firmy IBM)

SNAP *Subnetwork Access Protocol* (protokół dostępu do podsieci — terminologia wprowadzona przez IEEE dla kapsułkowania w sieci 802.2; rzadko stosowana w przypadku sieci TCP/IP)

S-NAPTR *Straightforward NAPTR* (prosty NAPTR — uproszczony NAPTR, gdzie łańcuch AUS jest bezpośrednio odwzorowywany na wynik bez podstawień wyrażeń regularnych)

SNMP *Simple Network Management Protocol* (prosty protokół zarządzania siecią — raportowanie stanu i ustawienia konfiguracyjne dla sprzętu sieciowego; używany zwykle na bazie UDP/IP)

SOA *Start of Authority* (początek strefy DNS — rekord zasobów DNS wskazujący metadane dotyczące strefy)

SOAP (dawniej) *Simple Object Access Protocol* (prosty protokół dostępu do obiektów — protokół aplikacyjny usług WWW używający języka XML, który dostarcza możliwości podobnych do RPC; nazwa SOAP nie jest już akronimem)

SPD *Security Policy Database* (baza danych polityki bezpieczeństwa — w protokole IPsec abstrakcja bazy danych zawierającej zasady bezpieczeństwa odnoszące się do sposobu obsługiwania ruchu sieciowego — np. odrzucanie, omijanie lub ochrona)

SPI *Security Parameter Index* (indeks parametrów bezpieczeństwa — w protokole IPsec indeks logiczny bazy SAD wskazujący parametry bezpieczeństwa, 32- lub 64-bitowy)

SPNAT, CGN, LSN *Service-Provider („large scale") NAT* (NAT dostawcy usługi/NAT „na dużą skalę" — sposób wdrożenia NAT, w którym translacja adresów jest wykonywana przez dostawcę usługi Internetu, a nie przez klienta)

SRP *Secure Remote Password* Protocol (bezpieczny zdalny protokół haseł — silny protokół uzgadniania kluczy oparty na hasłach; jest obsługiwany przez różne protokoły bezpieczeństwa, takie jak TLS i EAP)

SRTP *Secure Real-Time Transport Protocol* (bezpieczny protokół transmisji w czasie rzeczywistym — bezpieczny wariant protokołu czasu rzeczywistego opartego na protokołach UDP/IP; zazwyczaj używany do przesyłania informacji multimedialnych)

SRTT *Smoothed RTT* (wygładzone RTT — w protokole TCP uśrednione w czasie oszacowanie czasu RTT połączenia)

SSDP *Simple Service Discovery Protocol* (prosty protokół odkrywania usług — zdefiniowany przez IETF protokół odkrywania usług rozproszonych przeznaczony dla sieci LAN i sieci domowych, wykorzystywany przez technologię UPnP)

SSH *Secure Shell* Protocol (protokół bezpiecznej powłoki — protokół bezpiecznego logowania/bezpiecznej pracy zdalnej; obsługuje także tunelowanie innych protokołów)

SSID *Service Set Identifier* (identyfikator grupy usługowej — nazwa sieci 802.11)

SSL *Secure Sockets Layer* (warstwa bezpiecznych gniazd — warstwa stosująca szyfrowanie i ochronę integralności, znajdująca się powyżej protokołu TCP; prekursor protokołu TLS)

SSM *Single-Source Multicast* (multiemisja z pojedynczym źródłem — multiemisja, w której tylko pojedynczy nadawca może być źródłem ruchu sieciowego do konkretnej grupy)

STA *Station* (stacja — terminologia standardu IEEE 802.11 odnosząca się do punktu dostępowego lub skojarzonego z siecią bezprzewodowego hosta)

STP *Spanning Tree Protocol* (protokół drzewa rozpinającego — protokół używany w komunikacji między mostami i przełącznikami w celu uniknięcia pętli)

STUN *Session Traversal Utilities for NAT* (narzędzia sesji do trawersowania NAT — protokół typu klient-serwer pomagający w ustaleniu adresów i numerów portu ruchu sieciowego przy przechodzeniu przez NAT)

SWS *Silly Window Syndrome* (syndrom głupiego okna — w protokołach korzystających ze sterowania przepływem opartego na oknie danych niepożądana sytuacja, w której wymieniane są małe ilości danych z powodu używania małych rozmiarów okna)

SYN *Synchronize* (synchronizuj — bit nagłówka protokołu TCP i typ pierwszego segmentu przesyłanego w połączeniu TCP)

TCP *Transmission Control Protocol* (protokół sterowania transmisją — zorientowany na połączenie, niezawodny, strumieniowy protokół bez oznaczania granic komunikatów, który zawiera sterowanie przepływem i kontrolę przeciążenia)

TCP-AO *TCP Authentication Option* (opcja uwierzytelniania TCP — w protokole TCP mechanizm zwalczania ataków MSM, który może korzystać z wielu algorytmów)

TDES, 3DES *Triple DES* (potrójny DES — kodowanie przy użyciu trzech rund szyfrowania DES dające w rezultacie efektywną długość klucza wynoszącą 112 bitów)

TDM *Time Division Multiplexing* (multipleksowanie z podziałem czasu — współdzielenie kanału komunikacyjnego przez przydział oddzielnych szczelin czasowych do wyłącznego użytku)

TFC *Traffic Flow Confidentiality* (poufność przepływu ruchu — w protokole IPsec metody maskowania przepływu ruchu nawet zaszyfrowanego, w tym używanie dopełnień i generowanie fikcyjnych pakietów)

TFRC *TCP Friendly Rate Control* (kontrola szybkości transmisji przyjazna dla TCP, metody kontroli szybkości transmisji protokołu, tak aby uniknąć nieuczciwego współzawodnictwa z przepływem pod kontrolą TCP w podobnym środowisku operacyjnym)

TFTP *Trivial File Transfer Protocol* (trywialny protokół przesyłu plików — oparty na protokołach UDP/IP prosty protokół transferu)

TKIP *Temporal Key Integrity Protocol* (protokół integralności klucza tymczasowego — zastosowany w WPA, zastąpił algorytm szyfrowania WEP)

TLD *Top-Level Domain* (domena najwyższego poziomu — nazwa domeny najwyższego poziomu, taka jak EDU, COM, UK, ZA)

TLS *Transport Layer Security* (bezpieczeństwo warstwy transportowej — protokół oparty na SSL opracowany przez firmę Netscape)

TLV *Type/Length Value* (typ/długość wartość — schemat używany w protokołach; wskazuje typ, długość wartości o zmiennej długości i samą wartość)

ToS *Type of Service* (typ usługi — starsza nazwa bajta nagłówka IPv4 wskazującego typ usługi; zastąpiony przez pole DS i bity ECN)

TS *Traffic Selector* (selektor ruchu — w protokole IKE specyfikacje służące do identyfikowania ruchu sieciowego, takie jak zakres adresów IP, numer portu itd.)

TSER, **TSecr** *Timestamp Echo Reply* (echo znacznika czasu — w protokole TCP część opcji TSOPT używana do przekazywania echa wartości TSV drugiej stronie połączenia)

TSF *Time Synchronization Function* (funkcja synchronizacji czasu — ustala wspólny czas w grupie BSS sieci 802.11)

TSIG *Transaction Signatures* (podpisy transakcji — podpisy używane do zabezpieczenia indywidualnych transakcji DNS, lecz nie zawartości od jej powstania)

TSOPT *Timestamps Option* (opcja znaczników czasu — w protokole TCP opcja zawierająca wartości TSV i TSER)

TSPEC *Traffic Specification* (specyfikacja ruchu — struktura wskazująca parametry ruchu istotne dla QoS w sieci 802.11)

TSV *Timestamp Value* (wartość znacznika czasu — w protokole TCP część opcji TSOPT służąca do identyfikacji czasu nadawcy; wykorzystywana w mechanizmach RTTM i PAWS)

TTL *Time-to-Live* (pole nagłówka IPv4 pokazujące aktualną liczbę przeskoków przez router, która jeszcze przysługuje datagramowi)

TURN *Traversal Using Relay NAT* (trawersowanie NAT przy użyciu przekazywania — protokół, w którym trzecia strona przekazuje informacje między hostami niezdolnymi, w innym wypadku, do komunikowania się z powodu obecności jednego lub większej liczby urządzeń NAT)

TWA *Time-Wait Assassination* („zamach" na stan `TIME-WAIT` — w protokole TCP nieprawidłowa sytuacja spowodowana przez odebranie pewnych segmentów w czasie trwania stanu `TIME-WAIT`)

TXOP *Transmission Opportunity* (okazja do transmisji — w sieciach 802.11 forma „kredytu" pozwalająca stacji na wysłanie jednej ramki lub większej ich ilości)

TXT *Text* (tekst — rekord zasobów DNS zawierający tekst opisowy; wykorzystywany przez protokół DKIM)

UBM *Unicast Prefix-based Multicast addressing* (adresowanie multiemisji oparte na prefiksie emisji pojedynczej; tworzenie adresów multiemisji na podstawie przypisanych prefiksów emisji pojedynczej)

UDL *Unidirectional Link* (łącze jednokierunkowe — łącze obsługujące komunikację tylko w jednym kierunku)

UDP *User Datagram Protocol* (protokół datagramów użytkownika — protokół dostarczający komunikaty na zasadzie najlepszego starania, określający granice komunikatów i niezawierający kontroli przeciążenia ani sterowania przepływem)

UEQM *Unequal Modulation* (niejednakowa modulacja — jednoczesne używanie odmiennych typów modulacji na różnych strumieniach danych)

ULA *Unique Local IPv6 Unicast Addresses* (unikalne lokalne adresy emisji pojedynczej protokołu IPv6 — prywatne adresy używane w protokole IPv6, zaczynające się od prefiksu `fc00::/7`)

U-NAPTR *URI-enabled NAPTR* (NAPTR dopuszczający URI jako wynik reguły — uproszczony NAPTR dopuszczający ograniczony zakres podstawień przy użyciu wyrażeń regularnych)

U-NII *Unlicensed National Information Infrastructure* (nielicencjonowana narodowa infrastruktura informacji — nielicencjonowany zakres częstotliwości radiowych w dużej części świata)

UNSAF *Unilateral Self-Address Fixing* (jednostronne ustalanie własnego adresu — heurystyka stosowana w próbie ustalenia, w jaki sposób przepływ ruchu jest identyfikowany po przejściu przez NAT; słaba metoda, dla której rekomendowanymi alternatywami są techniki, takie jak ICE)

UP *User Priority* (priorytet użytkownika — priorytety w standardzie 802.11; oparte na takiej samej terminologii ze standardu 802.11d)

UPnP *Universal Plug and Play* (uniwersalna architektura typu „podłącz i używaj" — ramowy protokół odkrywania urządzeń i usług przeznaczony dla użytkownika indywidualnego; ustandaryzowany przez UPnP Forum)

URG *Urgent Mechanism* (mechanizm pilnych danych — w protokole TCP metoda oznaczania i identyfikowania informacji jako „pilnych"; niezalecana do użytku)

URI *Universal Resource Identifier* (uniwersalny identyfikator zasobów — łańcuch znaków identyfikujący nazwę lub zasób w Internecie, obejmujący adresy URL i nazwy URN)

URL *Uniform Resource Locator* (jednolity lokalizator zasobów — nieformalnie „adres WWW")

URN *Universal Resource Name* (uniwersalna nazwa zasobów — identyfikator URI używający schematu *urn* nieimplikującego dostępności zasobu)

USRK *Usage-Specific Root Key* (klucz główny dla określonego sposobu użycia — klucz wyprowadzony z klucza EMSK przeznaczony do używania w określonych celach)

UTC *Coordinated Universal Time* (uniwersalny czas koordynowany — standardowy czas używany przez NTP i inne protokoły; w praktyce zamienny z czasem GMT, ale z pewnymi technicznymi różnicami)

UTO *User Timeout* (czas oczekiwania użytkownika — w protokole TCP maksymalny czas, przez jaki nadawca będzie czekać, próbując retransmisji, przed rezygnacją z połączenia)

VC *Virtual Circuit* (obwód wirtualny — symulowana dedykowana ścieżka komunikacyjna)

VLAN *Virtual LAN* (wirtualna sieć LAN — rozwiązanie używane najczęściej do symulowania wielu oddzielnych sieci LAN korzystających ze wspólnego okablowania)

VLSM *Variable-Length Subnet Masks* (zmiennej długości maski podsieci — lokalne użycie masek podsieci o zróżnicowanej długości w tym samym środowisku)

VoIP *Voice over IP* (głos przez IP — przenoszenie ruchu zawierającego zapis głosu przez sieci IP, zwykle związane z sygnalizacją SIP)

VPN *Virtual Private Network* (wirtualna sieć prywatna — wirtualnie odizolowana sieć; często zaszyfrowana)

W3C *World Wide Web Consortium* (konsorcjum sieci ogólnoświatowej — organizacja SDO definiująca standardy sieci WWW, takie jak XML)

WAN *Wide Area Network* (sieć rozległa — sieć łącząca geograficznie rozproszone miejsca; zwykle zarządzana przez wielu administratorów)

WEP *Wired Equivalent Privacy* (prywatność na poziomie sieci kablowych — oryginalne szyfrowanie w sieciach Wi-Fi; okazało się, że było katastrofalnie słabe)

WESP *Wrapped ESP* (opakowany ESP — w protokole IPsec metoda poprzedzania enkapsulacji ESP nagłówkiem wskazującym, czy następujące po nim dane są

zaszyfrowane, czy tylko uwierzytelnione; przydatna w inspekcji wykonywanej przez urządzenia pośredniczące [middleboxes])

Wi-Fi *Wireless Fidelity* (bezprzewodowa wierność — standard IEEE 802.11 bezprzewodowych sieci LAN)

WiMAX *Worldwide Interoperability for Microwave Access* (ogólnoświatowa interoperacyjność dostępu mikrofalowego — standard IEEE 802.16 bezprzewodowego, szerokopasmowego dostępu do Internetu)

WKP *Well-Known Prefix* (dobrze znany prefiks — neutralny względem sumy kontrolnej prefiks IPv6 o wartości `64:ff9b::/96`, stosowany w algorytmicznych odwzorowaniach między adresami IPv4 i IPv6)

WLAN *Wireless LAN* (bezprzewodowy LAN — bezprzewodowa sieć LAN, taka jak Wi-Fi)

WMM *Wi-Fi Multimedia* (podzbiór funkcji QoS standardu 802.11e obecnie dostępny w standardzie 802.11n)

WoL *Wake on LAN* (budzenie przez sieć — metoda polegająca na pozostawaniu komputera w trybie „uśpienia", dopóki nie zostanie odebrany szczególnego rodzaju pakiet)

WPA *Wi-Fi Protected Access* (chroniony dostęp Wi-Fi — metoda szyfrowania dla sieci 802.11)

WPAD *Web Proxy Autodiscovery Protocol* (protokół automatycznego wykrywania serwera proxy WWW — protokół służący do odkrywania obecności najbliższego serwera pośredniczącego WWW)

WRED *Weighted RED* (RED z użyciem wag — algorytm RED, w którym prawdopodobieństwo zaznaczenia lub odrzucenia pakietu jest funkcją klasy ruchu i przypisanej wagi)

WSCALE, WOPT, WSOPT *Window Scale Option* (opcja skalowania okna — w protokole TCP opcja zawierająca czynnik skalujący, który powinien być zastosowany do zawartości pola *Rozmiar okna*)

WWW *World Wide Web* (sieć ogólnoświatowa — sieciowe środowisko udostępniania danych używające zestawu protokołów HTTP/TCP/IP)

X.25 *ITU-T recommendation X.25* (rekomendacja X.25 organizacji ITU-T — standard ITU-T sieci z przełączaniem pakietów obejmujący warstwy od 1. do 3. modelu OSI; najpopularniejsza technologia oparta na przełączaniu pakietów przed wejściem do szerokiego użycia protokołów TCP/IP)

XML *Extensible Markup Language* (rozszerzalny język znaczników — zbiór reguł kodowania dokumentów w postaci czytelnej dla komputerów; szeroko używany przez usługi WWW)

XMPP *Extensible Messaging and Presence Protocol* (rozszerzalny protokół przesyłania wiadomości i statusu dostępności — otwarty, rozszerzalny, oparty na języku HTML protokół wymiany wiadomości oraz przekazywania informacji o dostępności i listach kontaktów)

ZSK *Zone Signing Key* (klucz podpisujący strefę — klucz używany w protokole DNSSEC do podpisywania zawartości strefy, zwykle podpisany za pomocą klucza KSK)

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

Q

R

S

U

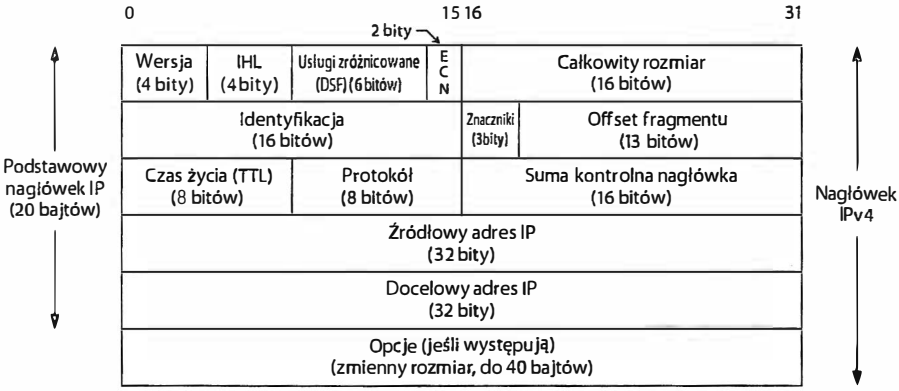
V

W

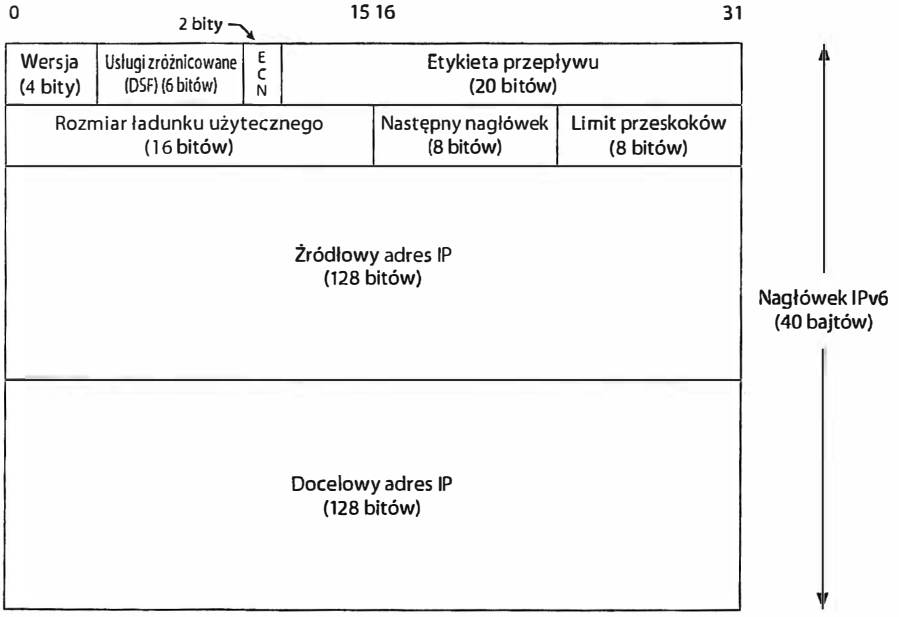
Z

Ż

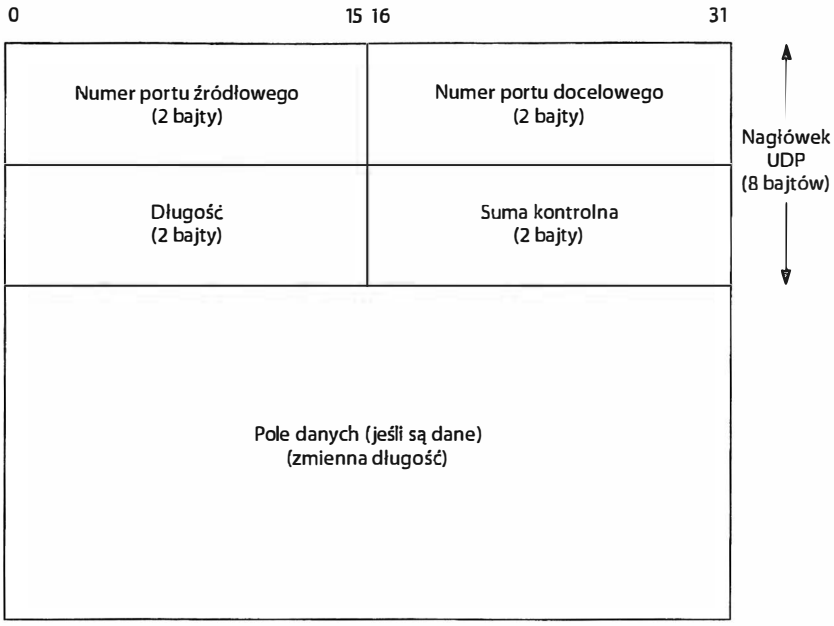
Nagłówek IPv4



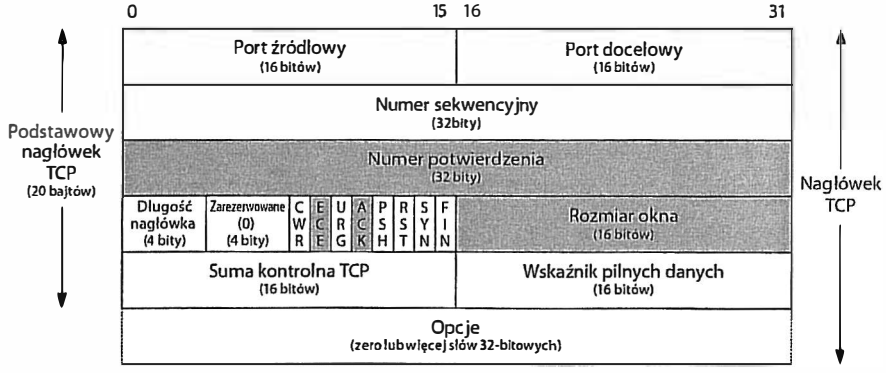
Nagłówek IPv6



Nagłówek UDP



Nagłówek TCP



Wydanie II

TCP/IP od środka Protokoły

Vademecum profesjonalisty

Kompletne źródło informacji na temat możliwości TCP/IP!

TCP/IP to model, bez którego nie byłoby sieci Internet – takiej, jaką dziś znamy. Pomimo słusznego wieku (pierwsze próby odbywały się w latach 70.) jest nadal w pełni wystarczający. Główne założenie modelu TCP/IP to podział całego procesu komunikacji na współpracujące ze sobą warstwy. Na tej podstawie zbudowane są różne protokoły transmisji danych, takie jak FTP, HTTP czy SMTP.

TCP/IP od środka. Protokoły. Wydanie II to szczegółowy, opatrzony wieloma ilustracjami przewodnik po współczesnych protokołach grupy TCP/IP. Uwzględnia najnowsze wersje tych protokołów i pokazuje ich funkcjonowanie „na żywo", w środowisku popularnych systemów operacyjnych, takich jak Windows, Linux i Mac OS X. Nie ma lepszego sposobu na wyjaśnienie, dlaczego właśnie tak wyglądają poszczególne aspekty działania TCP/IP, jak zmienia się ono w różnych okolicznościach oraz jak wykorzystać jego różne możliwości. To wyjątkowe opracowanie stanowi obowiązkową lekturę dla wszystkich osób chcących dowiedzieć się więcej o podwalinach współczesnej sieci. W trakcie lektury poznasz założenia architektoniczne, architekturę adresów internetowych oraz znaczenie i rolę poszczególnych warstw modelu TCP/IP. Dowiesz się, jak korzystać z komunikatów ICMP, rozgłaszać informacje w sieci, kontrolować przeciążenia w protokole TCP oraz korzystać z mechanizmów kryptograficznych. Znajdziesz tu dogłębne i intuicyjne wyjaśnienie wielu mechanizmów TCP/IP i Internetu, co pozwoli Ci bardziej efektywnie zarządzać swoimi sieciami i tworzyć lepsze aplikacje internetowe.

Kevin R. Fall zajmuje się protokołami TCP/IP od ponad ćwierćwiecza. Jest członkiem organizacji Internet Architecture Board oraz współprzewodniczącym grupy roboczej IETF Delay Tolerant Networking Research (DTNRG), zajmującej się problematyką wydajnego funkcjonowania sieci w warunkach ekstremalnych. Należy również do IEEE.

W. Richard Stevens był jednym z tych pionierskich autorów, na których książkach wychowało się całe pokolenie specjalistów od sieci TCP/IP, sukcesywnie sprowadzających Internet z wyżyn akademickich katedr do codziennego życia każdego człowieka. Wśród bestsellerów jego autorstwa można wymienić wszystkie trzy tomy *TCP/IP Illustrated* (Addison-Wesley) oraz *UNIX Network Programming* (Prentice Hall).

W tym znakomitym podręczniku znajdziesz informacje na temat:

- ***modelu TCP/IP***
- ***bezprzewodowych sieci LAN***
- ***architektury adresów internetowych***
- ***protokołu PPP***
- ***możliwości autokonfiguracji z wykorzystaniem DHCP***
- ***datagramów użytkownika — UDP***

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-4815-3

Cena 129,00 zł