Zaawansowany routing IP w Linuxie

Paweł Krawczyk
kravietz@echelon.pl

30 czerwca 2002

Spis treści

1 Wstęp

1.1 O czym jest ten artykuł?

Tekst ten omawia większość nowości dotyczących obsługi sieci IP, które pojawiły się w jądrze Linuxa 2.2. Nie jest to *vademecum* dla osób początkujących ani HOWTO. Jego zadaniem jest zapoznanie z użytecznymi funkcjami osób zajmujących się ruterami działającymi na Linuxie i posiadających podstawowe pojęcie o działaniu protokołu IP.

1.2 Nowości w jądrze

Początki wielkich zmian w Linuxie, dotyczących protokołu IP, to koniec 1996 roku, kiedy pojawiła się wersja 2.1.17. Zawierała ona pierwszą wersję przepisanego praktycznie od nowa kodu obsługującego IP. Nowością była zgodność z RFC 1812 [12] i routing rozszerzony (*policy routing*). Do konfiguracji nowych funkcji służył program `iproute`, potem pojawił się pakiet `iproute2`, zawierający program `ip`. Kolejne wersje jądra wprowadziły także mechanizmy sterowania przepływem danych (*traffic control*), obsługiwane za pomocą oddzielnego programu `tc` i opisane w osobnym artykule [1]. Rozwijanie tej części jądra Linuxa oraz pakietu `iproute2` to przede wszystkim zasługa Aleksieja N. Kuźniecowa *¡kuznet@ms2.inr.ac.ru¿* z moskiewskiego Instytutu Fizyki Jądrowej.

1.3 Nowości w iproute2

Obsługa `iproute2` różni się zasadniczo od dotychczas stosowanych narzędzi – `ifconfig` i `route`. W programach tych adres IP oraz maska podsieci były podawane oddzielnie, przy czym ta ostania była tradycyjnie zapisywana w postaci *aaa.bbb.ccc.ddd*. Program `ip` przyjmuje natomiast maskę podsieci w formacie skróconym *aaa.bbb.ccc.ddd/nn*, gdzie *nn* jest którą określa się jako liczbę bitów jedynkowych w masce. Przykładowo, maska sieci C zapisywana tradycyjnie jako 255.255.255.0 ma 24 bity jedynkowe, podsieci 255.255.255.240 – 28 itp.

[1] Paweł Krawczyk, „Sterowanie przepływem danych w Linuxie 2.2”

Wiąże się z tym także możliwość używania skróconych postaci adresów IP, gdzie brakujace bajty są zastępowane zerami. Przykładowo:

Notacja skrócona	Notacja pełna
127/8	127.0.0.0/8
192.168/16	192.168.0.0/16
0/0	0.0.0.0/0

Należy podkreślić, że jest to notacja zgodna z filozofią CIDR ([2], [3]) oraz IPv6 i zalecana przez RFC 1812 ([12]), gdzie tradycyjną konstrukcję adresu IP (adres sieci, adres podsieci, adres hosta) zastępuje się przez adres prefiksu i adres hosta. W konsekwencji wyżej opisana notacja maski podsieci odpowiada po prostu długości prefiksu w bitach.

1.4 Konfiguracja adresów interfejsów

Każdy interfejs może posiadać więcej niż jeden adres IP. Dodatkowe adresy są po prostu adresami, różniącymi się od podstawowego adresu tylko flagą **secondary**, a nie samodzielnymi pseudointefejsami, jak to było w jądrach 2.0 i wcześniejszych.

Do ustawiania adresu intefejsu służy polecenie **ip addr**:

```
ip addr add ADRES dev INTERFEJS
[ ( peer | remote ) P-T-P ]
[ ( broadcast | brd ) BROADCAST ]
[ anycast ANYCAST ] [ label NAZWA ]
[ scope ZASIĘG ]
```

Parametr `ADRES` jest adresem dodawanym do interfejsu. Należy zaznaczyć, że w odróżnieniu od programu `ifconfig` adresy IPv4 podaje się razem z maską podsieci jako *aaa.bbb.ccc.ddd/nn*.

Adresy IPv6 podaje się standardowo jako *aa:bb:. . . :zz/nnn*, gdzie */nnn* to długość prefiksu (maski podsieci).

W przypadku intefejsów *point-to-point* (na przykład PPP) parametr *ADRES* określa adres lokalny intefejsu, natomiast adres drugiego końca należy podać po parametrze **peer** (akceptowane jest również słowo **remote**).

Do ustawiania adresu rozgłoszeniowego (*broadcast*) służy parametr **broadcast** (lub krócej **brd**). W nowych wersjach `iproute` parametr `+` spowoduje adresu obliczonego automatycznie na podstawie długości prefiksu.

Adres *anycast* stosowany w IPv6 ustawia sie parametrem **anycast**. [2]

[2]Patrz http://www.whatis.com/anycast.htm

Zasięg adresu (*scope*) określa się parametrem **scope**, który może być jednym ze standardowych zasięgów, albo nazwą (lub numerem) zasięgu zdefiniowanego przez użytkownika. [3]

Interfejsowi można nadawać nazwy za pomocą parametru **label**, co jest przydatne w przypadku dodawania kolejnych adresów do interfejsu (*eth0:1, eth0:2,...*). W ten sposób do dodatkowych adresów można się także odwoływać za pomocą starszych wersji polecenia *ifconfig*.

2 Konfiguracja parametrów interfejsu

Dodanie adresu do interfejsu nie powoduje jego automatycznej aktywacji (flaga *UP*) Jest to zachowanie odmienne od znanego z jąder 2.0 Do konfigurowania tego – i innych – parametrów intefejsu służy polecenie **ip link**:

```
ip link set INTERFEJS [ ( up | down ) ]
[ arp ( on | off ) ]
[ multicast ( on | off ) ]
[ ( txqueuelen | txqlen ) PKT ]
[ name NAZWA ]
```

- Flagi **up** i **down** powodują odpowiednio aktywację lub deaktywację interfejsu. W momencie aktywacji interfejsu kernel dodaje do tablicy routingu trasy kierujące na ten interfejs pakiety do sieci do niego przyłączonej. Jest ona umieszczana w specjalnej tablicy `local`. Warto zauważyć, że w jądrach 2.0 i wcześniejszych taką trasę trzeba było dodać *explicite* za pomocą polecenie `route`, obecnie nie jest to już konieczne.

- Opcja **arp** włącza lub wyłącza protokół ARP dla danego interfejsu. Oznacza to, że interfejs nie będzie odpowiadał na pytania ARP, nawet jeśli pytanie dotyczy jego adresu IP. [4]

- Opcja **multicast** włącza lub wyłącza obsługę pakietów *multicast* na danym interfejsie.

- Parametr **txqueulen** (akceptowany jest skrót **txqlen**) określa wielkość kolejki wyjściowej danego interfejsu w pakietach. Wielkość ta jest ustawiana przez system automatycznie na podstawie domyślnej wartości – innej dla każdego rodzaju interfejsu – a zależnej od jego przepustowości. Z reguły ustawienie domyślne jest

[3] Więcej na temat zasięgów można przeczytać w Dodatku B.

[4] Flagę **NOARP** mają ustawioną automatycznie na przykład intefejsy typu *point-to-point* (PPP, SLIP), interfejs *dummy* itp.

wystarczające, jego zmiana może być czasem korzystna na przykład do poprawienia współdzielenia wolnego łącza PPP. [5]

3 Komunikacja hostów w obrębie sieci lokalnej

3.1 Wprowadzenie

W IPv4 informacja o adresie w warstwie łącza (*link layer address*) interfejsu posiadającego dany adres IP jest uzyskiwana za pomocą protokołu ARP (*Address Resolution Protocol*, RFC 826 [1]), korzystającego z rozgłaszania w warstwie łącza i różniącego się implementacją dla każdego rodzaju sieci fizycznej (ARP dla Ethernetu różni się od ARP dla ATM). W IPv6 służy do tego protokół wyszukiwania hostów sąsiadujących (*Neighbor Discovery*), oparty o pakiety *multicast* i działający nad warstwą IP. Dokładny jego opis – oraz terminologię – można znaleźć w RFC 2461 [6].

W związku z tym, kernel tablica sąsiednich hostów, przechowywana przez kernel, jest niezależna od protokołu i zawiera informacje uzyskane albo za pomocą ARP albo za pomocą ND. Służy ona także jako baza danych do wprowadzonego jako standard przez IPv6, ale działającego także dla IPv4, mechanizmu weryfikacji osiągalności hosta (*Neighbor Unreachability Discovery*).

Tablica sąsiednich hostów zawiera następujące informacje: adres IP hosta, adres hosta w warstwie łącza (*link layer address*) oraz aktualny stan rekordu. Dodatkowo są w niej przechowywane takie informacje jak ilość odwołań do danego rekordu oraz czas ostatniego odwołania. Główne stany rekordów to: **incomplete** (żaden host nie odpowiedział na pytanie o dany adres), **reachable** (host jest osiągalny) oraz *stale* (host był osiągalny, lecz rekord jest przeterminowany). Ponadto istnieją dwa stany specjalne: **noarp** (rekord nie jest uaktualniany przez ARP ani ND) oraz **permanent** (rekord dodany ręcznie przez administatora). Oba nie są nigdy zmieniane przez kernel, nie jest w stosunku do nich używany ARP, ND ani NUD, a rekordy w stanie `permanent` nie są usuwane podczas okresowego czyszczenia tablicy (*garbage collecting*).

3.2 Obsługa tablicy sąsiadów

Do manipulacji tablicą zawierającą informacje o adresach fizycznych hostów i odpowiadających im adresach IP służy polecenie **ip neigh**:

```
ip neigh ( add | del )
(ADRES [ lladdr LLADDR ] | proxy PROXY )
[ nud ( permanent | noarp | stale | reachable ) ]
dev INTERFEJS
```

[5] Dla intefejsów PP wielkością domyślną jest 10 pakietów. Alan Cox zaleca w takim wypadku zmniejszenie tej wartości do 3.

Polecenie `ip neigh` może dodawać dwa typy rekordów do tablicy:

- **Adres rzeczywisty** Parametr *ADRES* jest wtedy adresem IPv4 lub IPv6, którego rekord chcemy dodać lub usunąć z tablicy hostów sąsiadujących. Adres warstwy łącza związany z dodawanym adresem IP podajemy natomiast w parametrze **lladdr**.
- **Adres Proxy ARP** Parametr *PROXY* określa adres IP hosta, dla którego dany interfejs ma pośredniczyć w protokole ARP. Więcej informacji o technice „Proxy ARP", której nie będziemy tutaj opisywać, można znaleźć na przykład w „Network Administrator's Guide" [10] w rozdziale *Configuring TCP/IP Networking*, sekcja *Checking the ARP Tables*.

4 Ruting

4.1 Wstęp.

W kernelach 2.0 była tylko jedna podstawowa tablica routingu. W kernelach 2.2 tablic tych może być do 250 zdefiniowanych tablic, z czego domyślnie aktywne są trzy:

- local (255)
- main (254)
- default (253)

Tablica **local** zawiera trasy dodawane automatycznie przez kernel, takie jak trasy do lokalnych interfejsów oraz trasy broadcastowe. Trasy w tej tablicy mają z reguły zasięg *host* lub *link*.

Tablica **main** odpowiada starej tablicy routingu w kernelach 2.0 i do niej trafiają trasy dodawane przez użytkownika, jeśli nie poda on innej tablicy. Do niej dodawane są również trasy tworzone automatycznie przez kernel w momencie aktywacji interfejsu.

Tablica **default** jest domyślnie pusta.

Ponadto istnieje także tablica **cache**, która jest tworzona automatycznie i traktowana w odmienny sposób niż wyżej wymienione tablice. W szczególności, jej zawartość jest uzupełniana automatycznie przez kernel i nie jest ona dostępna do zapisu przez użytkownika. Jej zawartość można natomiast przeglądać, co jest opisane poniżej.

Tablice są przeglądane w kolejności: *local*, *main*, *default*. Pozostałe tablice **nie są** przeszukiwane automatycznie, znajdują natomiast zastosowanie w routingu rozszerzonym (*policy routing*).

Zasady nazewnictwa poszczególnych tablic nieco się zmieniły w kolejnych wersjach `iproute2`. W chwili obecnej odwzorowanie nazw na numery tablic jest definiowane w pliku `/etc/iproute2/rt_tables`. Domyślnie znajdują się tam własnie take nazwy jak powyżej, nic nie stoi jednak na przeszkodzie, by definiować i dodawać własne.

4.2 Obsługa tablic routingu.

Do operacji na tablicach tras służy polecenie **ip route**. Jego składnia jest bardzo rozbudowana, więc w tej sekcji ograniczymy się tylko do omówienia głowych funkcji polecenia `ip route`. Szczegółowe omówienie parametrów znajduje się w dodatku C.

Dodawanie oraz usuwanie tras z tablicy odbywa się za pomocą polecenia `ip route add` (lub `del`), którego argumentem jest specyfikacja trasy. W najprostszym przypadku składa się ona z adresu sieci docelowego oraz adresu rutera do tej sieci prowadzącego (*next hop*). Poniżej ograniczymy się do omówienia mniej lub bardziej typowych przypadków.

```
ip route add 192.168.2.0/24 via 192.168.1.1
```

Powyższe polecenie dodaje do głównej tablicy routingu trasę prowadzącą do sieci `192.168.2.0/24` przez ruter o adresie `192.168.1.1`.

```
ip route add 192.168.2.0/24
  via 192.168.1.1 table default
```

Dodaje taką samą trasę, ale do tablicy *default*.

```
ip route add 0/0 via 192.168.0.1
```

Dodaje trasę domyślną (*default*) przez ruter `192.168.0.1`. Warto zwrócić uwagę na skrótową formę zapisu sieci przeznaczenia `0.0.0.0/0`, oznaczającej trasę domyślną i zapisanej skrótowo jako `0/0`.

```
ip route add 10.1/16 via 192.168.1.1
  src 10.2.2.1
```

Trasa, którą to polecenie dodaje nie jest tak interesująca jak występujący w nim parametr `src`. Powoduje on, że pakiety wychodzące z hosta tą trasą, będą miały adres źródłowy ustawiony jako `10.2.2.1`. Dotyczy to tylko pakietów inicjowanych przez host lokalny (tj. przy połączeniach wychodzących).

Taka konfiguracja może być przydatna na przykład jeśli nasza sieć jest podłączona do kilku providerów (*multi-homed*) bez routingu dynamicznego i chcemy by pakiety wysyłane do jednego z nich wracały tą samą trasą. W przeciwnym razie wracałyby one trasą podyktowaną przez adres źródłowy wynikający z adresu naszego głównego interfejsu.

```
ip route add unreachable 10.1.2/24
```

Specjalny rodzaj trasy. Pakiety kierowane do sieci docelowej `10.1.2.0/24` zostaną odrzucone, a w odpowiedzi zostanie odesłany komunikat ICMP „Host unreachable”. Dostępne są także inne tego rodzaju trasy: `prohibit` i `blackhole`. Pierwsza z nich zwraca pakiet „Packet administratively prohibited”, a druga usuwa pakiet bez zwracania żadnej informacji.

Trasy te można efektywnie wykorzystać przynajmniej na trzy sposoby:

- Jako znacznie szybsze wersje części regułek filtrujących *firewall*, co jest dokładniej opisane w rozdziale 6 (str. 10).

- Do translacji adresów (*Network Address Translation*) przez trasę `nat`. Patrz rozdział 5.3 (str. 9).
- Wykorzystanie trasy `unreachable` dla uniknięcia pętli routingu powstających przy dynamicznie tworzonych interfejsach typu PPP. Przypadek ten jest dokładniej opisany w dodatku 7.2 (str. 12).

5 Ruting rozszerzony

5.1 Wstęp

W tradycyjnym modelu ruter dokonuje wyboru trasy tylko na podstawie adresu przeznaczenia pakietu. Kryteria wyboru trasy dla konkretnego pakietu, z kilku o różnym zasięgu, określa RFC 1812 [12]. Na przykład, jeśli w tablicy routingu znajdują się dwie trasy – jedna na podsieć i druga na hosta w tej podsieci – to pierwszeństwo będzie miała trasa bardziej szczegółowa, na hosta. Jest to tzw. reguła „najdłuższego dopasowania" trasy (*longest match*). Pozostałe reguły wyboru tras przez ruter można znaleźć w sekcji 5.2.4.3 RFC 1812 [12].

Linux 2.2 spełnia wszystkie zdefiniowane przez ten dokument wymagania wobec ruterów internetowych. Poza podstawowymi funkcjami posiada on potężny mechanizm w postaci routingu rozszerzoneg (*policy routing*).

5.2 Ruting rozszerzony

Routing rozszerzony pozwala na wybranie trasy według wymienionych poniżej selektorów, które mogą być łączone w dowolnych kombinacjach. Selektory te, jak widać, odpowiadają poszczególnym polom pakietu IP:

Parametr	**Selektor**
Adres źródłowy	`from`
Adres docelowy	`to`
Typ usługi (*Type of Service*)	`tos`
Interfejs z którego przychodzi pakiet	`iif`
Oznakowanie nadane przez firewall	`fwmark` [6]

Do każdej regułki, będącej grupą selektorów , przypisany jes priorytet (`pref`) oraz akcja. Priorytet określa kolejność sprawdzania regułek – regułki o niższym priorytecie są sprawdzane wcześniej. Akcja jest to sposób w jaki zostanie potraktowany pakiet pasujący do danej regułki. Dostępne są następujące podstawowe akcje:

- `table TABLICA` Pakiet jest kierowany według tradycyjnego algorytmu wyboru trasy, na podstawie tablicy routingu określanej identyfikatorem `TABLICA`.

- `nat SIEC/MM` Adres **źródłowy** pakietu jest tłumaczony na adres z sieci podanej w parametrze `SIEC`. Ponieważ tłumaczenie jest w stosunku 1:1, więc rozmiar nowej sieci określany maską `MM` musi być taki sam jak rozmiar sieci tłumaczonej. Należy podkreślić, że za pomocą routingu rozszerzonego tłumaczy się jedynie adresy źródłowe pakietów. Translacji w drugą stronę dokonuje się za pomocą odpowiedniego wpisu w tablicy routingu, co jest opisane w sekcji 5.3 (str. 9).

- `unreachable, prohibit, reject` Odrzucają pakiety na różne sposoby, analogicznie jak trasy specjalne opisane w sekcji 4.2 (str. 7).

- `realms`

Należy pamiętać o dwóch rzeczach. Po pierwsze, regułka nakazująca przeszukiwanie tablicy **local** ma zawsze pierwszeństwo i nie jest możliwe zmiana jej priorytetu. Z tego powodu nie będą działać regułki w których selektorem jest adres docelowy jednego z interfejsów danego hosta. Takie przypadki można jednak rozwiązać dodając stosowną trasę do tablicy *local*.

Po drugie, pierwszeństwo mają także trasy znajdujące się w tablicy **cache**. W związku z tym, po dodaniu nowych regułek należy tę tablice wyczyścić poleceniem `ip route flush table cache`.

5.3 Translacja adresów

Linux umożliwia translację adresów IP na dwa sposoby:

- dynamicznie, w stosunku „wiele do 1”,

- statycznie, w stosunku „1 do 1”

Pierwszy algorytm był dostępny już w Linuxie 2.0 jako maskowanie adresów (*IP masquerading*). Jest to zaawansowana funkcja, pozwalająca na schowanie nawet bardzo dużej sieci komputerów posiadających adresy prywatne (zdefiniowane w RFC 1918 [4]) za ruterem maskującym. Istnieje wiele opisów działania i konfiguracji maskowania adresów IP [7].

Linux 2.2 umożliwia także statyczne tłumaczenie adresów w stosunku 1 do 1. Oznacza to, że określonej podsieci adresów prywatnych musi być przyporządkowana podsieć adresów publicznych o tej samej wielkości. NAT jest realizowany na poziomie tablicy routingu oraz rutingu rozszerzonego. Tłumaczenie adresów źródłowych i docelowych konfiguruje się oddzielnie.

Tłumaczenie adresów docelowych konfiguruje się za pomocą polecenia `ip route` i polega ono na stworzeniu specjalnej trasy `nat`. Sieć docelowa określa zakres adresów,

[7] Na przykład „IP Masquerade mini HOWTO”, *http://www.jtz.org.pl/Html/mini/IP-Masquerade.pl.html*

które mają być tłumaczone, a adres rutera (`via`) jest pierwszym adresem z sieci na którą mają być tłumaczone adresy docelowe. Trasa ta musi znaleźć się w tablicy `local`:

```
ip route add nat 192.168.1.0/24 via 195.116.211.0 table local
```

Tłumaczenie adresów źródłowych ustawia się za pomocą odpowiedniej regułki routingu rozszerzonego. W tym wypadku selektor `from` określa sieć, która ma być tłumaczona, a akcja `nat` – adres pierwszego adresu sieci, na którą mają być tłumaczone adresy źródłowe.

```
ip rule add from 195.116.211.0/24 nat 192.168.1.0
```

Trzeba pamiętać, że translacja adresów skonfigurowana w ten sposób działa wyłacznie dla pakietów rutowanych przez dany system, czyli wchodzących jednym interfejsem i wychodzących innym. W szczególności nie obejmuje ona pakietów generowanych przez sam ruter, czyli na przykład połączeń wychodzących z tego rutera. Adres źródłowy takich połączeń można zdefiniować natomiast przy pomocy parametru `src` polecenia `ip route`, który jest opisany w rozdziale 4.2 na stronie 7.

6 Optymalizacja

Podczas projektowania oraz konfiguracji ruterów, mających obsługiwać duży ruch należy brać pod uwagę wydajność systemu oraz opóźnienia powstające podczas przeszukiwania dużych tablic routingu lub długich list filtra pakietów. Generalnie, przeszukiwanie tablic routingu jest znacznie szybsze niż przeszukiwanie regułek filtra. To pierwsze odbywa się z pomocą funkcji haszujących, podczas gdy drugie jest zwykłym przeszukiwaniem liniowym [8].

6.1 Filtrowanie za pomocą tablicy routingu

Filtrowanie pakietów przy pomocy `ipchains` zużywa więcej czasu procesora niż przeszukiwanie tablic routingu. Jeśli jedynymi kryteriami filtrowania są adresy źródłowe lub docelowe pakietów IP, to optymalniejsze będzie skorzystanie ze specjalnej trasy `prohibit` (opisanej w sekcji 4.2, str. 7). Na przykład, regułkę `ipchains`:

```
ipchains -A input -d 192.168.0.0/24 -j REJECT
```

Można zastąpić przez:

```
ip route add prohibit 192.168.0.0/24
```

W przypadku regułek odrzucających pakiety na podstawie adresu źródłowego należy skorzystać z routingu rozszerzonego i polecenia `ip rule add from`.

[8] Interesujące opracowanie na ten temat można znaleźć w pracy [13]

6.2 Optymalizacja filtra pakietów

Czas przeszukiwania listy regułek filtra pakietów można zredukować układając je w odpowiedniej kolejności. Dla każdego sprawdzanego pakietu lista jest przeszukiwana od początku aż do pierwszej pasującej do niego regułki.

Po pierwsze, przy pomocy zliczania pakietów należy określić które regułki mają najwięcej trafień i umieścić je na początku listy.

Po drugie, w przypadku protokołu TCP można zaszczędzić czas przeszukując tablicę tylko dla pakietów rozpoczynając połączenie (wyróżnionych flagą `SYN`). Można to zrealizować dodając na początku listy regułkę przepuszczającą wszystkie pakiety dla protokołu TCP i **nie** opatrzone flagą `SYN`:

```
ipchains -A forward -p tcp ! -y
```

6.3 Optymalizacja tablicy routingu

Jeśli tablica routingu danego rutera posiada lub ma w założeniu posiadać więcej niż 100 pozycji, to podczas konfiguracji kernela należy włączyć opcję `CONFIG_IP_ROUTE_LARGE_TABLES` (`Networking options`, `IP: Advanced router`, `IP: large routing tables`). Spowoduje to przebudowanie struktur odpowiedzialnych za szybkie wyszukiwanie pozycji w tablicy podczas każdego dodawania do niej nowych pozycji.

7 Dodatki

7.1 Zasięgi adresów

Zasięg (*scope*) można rozpatrywać w kontekście adresów interfejsów oraz tras. Jako standardowy element protokołu IP zasięg adresu jest zdefiniowany jednoznacznie w IPv6 w RFC 2373 [8]. W adresie IPv6 zasięg determinują pierwsze 4 bity adresu. I tak, adres IPv6 może posiadać następujący zasięg:

node–local

host–local Adres lokalny oznaczający tę samą maszynę, tak jak adresy z klasy 127/8 w IPv4. Adres ten jest nadawany interfejsowi lokalnemu i nie może być przesyłany przez rutery.

link–local Adres lokalny, obowiązujący wyłącznie w sieci lokalnej, do której należy dany host. Każdy host IPv6 musi posiadać taki adres, konfigurowany automatycznie przez system w momencie aktywacji interfejsu sieciowego. Istnienie adresów typu `link-local` jest bardzo wygodne, ponieważ pozwala na komunikację między hostami leżącymi w jednej sieci lokalnej bez uprzedniego przydzielania i konfiguracji adresów. W oparciu o adresy `link-local` można także tworzyć sieci prywatne, do których w IPv4 służyły klasy wydzielone z publicznej puli adresów

(192.168/16 itp.). Pakiety z adresami o tym zasięgu nie mogą być przekazywane przez rutery.

site–local Adres lokalny, obowiązujący w ramach administratcyjnie określonej sieci prywatnej. Adresy o tym zasięgu nie są konfigurowanie automatycznie i muszą być zdefiniowane przez administratora sieci. Jednak pakiety z adresami `site-local` mogą być przekazywane przez rutery, co pozwala na tworzenie sieci prywatnych obejmujących wiele segmentów LAN.

global Adresy publiczne.

W przypadku IPv4 jądro Linuxa również rozróżnia zasięgi adresów, nie są one jednak częścią standardu tylko raczej przeniesieniem wygodnej metody rozróżniania adresów z IPv6. Dla IPv4 jądro rozróżnia następujące zasięgi:

host–local Odpowiednik zasięgu `host--local` w IPv6, zarezerwowany dla adresów z klasy 127/8.

link–local Odpowiednik zasięgu `link--local` w IPv6. W przypadku IPv4 zasięg ten jest nadawany adresowi warstwy łącza danego interfejsu, czyli np. adresowi MAC karty sieciowej w Ethernecie.

site–local Odpowiednik zasięgu `site-local` w IPv6. Ponieważ jednak rezerwacja określonych klas adresów IPv4 dla sieci prywatnych (RFC 1918 [4]) jest tylko umowna, zasięg `site-local` można nadać każdemu adresowi, chociaż sens ma nadawanie go tylko adresom z klas zarezerwowanych.

universe, global Adresy publiczne.

Możliwe jest definiowanie własnych wartości zasięgów. [11] podaje przykład zasięgu dla tras wewnątrzsieciowych (*interior*), którym przypisuje się wartość zasięgu pomiędzy *universe* i *link*.

Więcej informacji na temat zasięgów można znaleźć w dokumentach RFC 2373 [8] , RFC 2374 [5] oraz RFC 2462 [7].

7.2 Wykorzystanie tras `unreachable`

Jeśli mamy serwer dostępowy z pulą modemów, na które poświęcamy Podsieć określonej wielkości, to z reguły na głównym ruterze jest ustawiona trasa kierująca pakiety do tej podsieci na adres serwera dostępowego.

Jeśli dany modem jest zajęty to odpowiadający mu adres IP posiada trasę w tablicy routingu, kierującą pakiety do niego na przydzielony dynamicznie interfejs. Natomiast w momencie gdy jakiś adres nie jest przydzielony wdzwaniającemu się klientowi, pakiet przeznaczony na dany adres IP zostanie skierowany według trasy

domyślnej i wróci do rutera. Ten z powrotem odbije go na serwer dostępowy, i tak aż do wyczerpania czasu życia (TTL) pakietu.

Jeśli na serwerze dostępowym stworzymy trasę `unreachable` dla danej podsieci z – co istotne – dużą miarą `metric`, to pakiety kierowane na nieaktywne w danym momencie adresy IP zostaną odrzucone z komunikatem „Host unreachable", co będzie zgodne z prawdą.

7.3 Znakowanie pakietów

Nowy kod filtra pakietów Linuxa posiada przydatną funkcję znakowania pakietów pasujących do danej regułki. Oznakowanie jest liczbą 32–bitową, której wartość określa parametr `-m` polecenia `ipchains`. Najprościej opisać to na przykładzie. Tworzymy odpowiedni łańcuch:

```
# ipchains -A forward -p tcp -d 192.168.1.1/32 25 -j ACCEPT -m 0x1234
# ipchains -vL forward
Chain forward (policy ACCEPT: 315108 packets, 107748450 bytes):
pkts bytes target prot tosa tosx mark source destination ports
0 0 ACCEPT tcp 0xFF 0x00 0x1234 anywhere 192.168.1.1 any -> smtp
```

Wszystkie pakiety wysyłane na port 25 hosta 192.168.1.1 zostaną przepuszczone (docelowy łańcuch `ACCEPT`) oraz opatrzone znakiem `0x1234`. Jak można wykorzystać takie oznakowanie? Jest to bardzo użyteczny mechanizm, pozwalający na zmianę zachowania rutera (działającego w warstwie sieci) wobec pakietu w zależności od cech charakterystycznych dla protokołów warstwy transportowej. Czyli przede wszystkim numerów portów protokołów TCP oraz UDP, jak również typów pakietów ICMP. Można to wykorzystać przynajmniej na dwa sposoby:

- W routingu rozszerzonym, do kierowania różnymi trasami pakietów należących do różnych protokołów, za pomocą parametru **fwmark** (patrz rozdział 5.2, str. 8).
- W kontroli przepływu danych, do przydziału pasma oraz priorytetów w zależności od protokołu. Służy do tego filtr **fw** oraz odpowiednio zdefiniowane oznaczenia pakietów: identyfikatorowi przepływu 1:1 odpowiada oznaczenie 0x10001, 2:3 – 0x20003 itd.

7.4 Weryfikacja adresów

Sekcja RFC 1812 [12] sugeruje, by rutery posiadały możliwość weryfikacji, czy pakiet przychodzący danym interfejsem posiada adres źródłowy z sieci, która jest przez ten interfejs rutowana.

Wyobraźmy sobie, że dany ruter posiada dwa interfejsy `eth0` i `eth1`. Pierwszy z nich jest wyjściem na świat i skierowana jest na niego trasa domyślna. Na drugi interfejs jest ustawiona trasa 195.116.2111.0/24, kierująca do znajdującej się za tym ruterem sieci LAN. W normalnej sytuacji pakiet z adresem **źródłowym** z sieci 195.116.211.0/24 nie ma prawa pojawić się na zewnętrznym interfejsie `eth0`. Pakiety z tej sieci mogą bowiem być generowane wyłącznie w sieci lokalnej, znajdującej się za intefejsem `eth1`.

Ruter posiadający filtr zgodny z RFC 1812 powinien takie pakiety przechwytywać i kasować. Z reguły są one efektem błędnej konfiguracji gdzieś w odległych zakątkach sieci lub – co gorsza – próby przeprowadzenia ataku przez sfałszowanie adresów IP (*address spoofing*).

W Linuxie weryfikację adresów włącza się za pomocą interfejsu `systcl`, czyli zapisując odpowiednią wartość w pliku znajdującym się w systemie plików `/proc`. Są dostępne dwa poziomy weryfikacji – silniejsza, w pełni zgodna z RFC 1812 (wartość 2) i słabsza, dotycząca tylko sieci bezpośrednio przyłączone do danego hosta (wartość 1). By zupełnie wyłączyć weryfikację adresów należy zapisać wartość 0, tak jak w poniższym przykładzie:

```
echo 0 >/proc/sys/net/ipv4/conf/all/rp_filter
```

Weryfikacji adresów nie należy włączać w określonych wypadkach, kiedy mogłaby ona w ogóle uniemożliwić łączność. Na przykład, jeśli routing do jakiejś sieci jest asymetryczny (pakiety wchodzą jednym interfejsem, wychodzą innym).

7.5 Ograniczenia prędkości odpowiedzi ICMP

Linux 2.2 posiada możliwość ograniczenia prędkości wysyłania na określone pakiety ICMP. Ma to na celu zmiejszenie przeciążenia sieci, zasobów rutera oraz ograniczenie ataków polegających na zalewaniu ofiary potokiem odpowiedzi na fałszywy pakiet ICMP. Jest to funkcja zalecana przez RFC 1812 [12] w punkcie 4.3.2.8.

W przypadku Linuxa limity określa się przez podanie minimalnego odstępu czasowego pomiędzy dwoma odpowiedziami ICMP określonego typu. Czas ten określa się w *jiffies*, czyli podstawowej wewnętrznej jednostce czasu jądra Linuxa. W systemach opartych o procesory Intela i kompatybilne jest to 1/100 sekundy, a w przypadku procesorów Alpha – 1/1024 sekundy.

Limitowaniu podlegają następujące pakiety ICMP wyliczone poniżej wraz z odpowiadającymi im pozycjami w `sysctl`:

- *Destination unreachable*
  `/proc/sys/net/ipv4/icmp_destunreach_rate`
- *Parameter problem*
  `/proc/sys/net/ipv4/icmp_paramprob_rate`

- *Time exceeded*
  `/proc/sys/net/ipv4/icmp_timeexceed_rate`

- *Echo reply*
  `/proc/sys/net/ipv4/icmp_echoreply_rate`

Wszystkie wyżej wymienione limity są domyślnie ustawione na 1 sekundę, za wyjątkiem `icmp_echoreply_rate`, które ma wartość 0 – czyli brak limitu.

W praktyce działanie limitów objawia się na przykład brakiem odpowiedzi na jedną z próbek polecenia `traceroute`, jak w poniższym przykładzie:

```
$ traceroute tau2
traceroute to tau2.ceti.com.pl (195.116.211.16), 30 hops max, 40 byte packets
 1  tau2 (195.116.211.16)  0.282 ms *  0.256 ms
```

Literatura

[1] David C. Plummer, „An Ethernet Address Resolution Protocol", November 1982 (RFC 826)

[2] Y. Rekhter, T. Li, „An Architecture for IP Address Allocation with CIDR", September 1993 (RFC 1518)

[3] V. Fuller, T. Li, J. Yu, K. Varadhan, „Classless Inter-Domain Routing (CIDR)", September 1993 (RFC 1519)

[4] Y. Rekhter *et al.*, „Address Allocation for Private Internets", February 1996 (RFC 1918)

[5] R. Hinden, M. O'Dell, S. Deering, „An IPv6 Aggregatable Global Unicast Address Format", July 1998 (RFC 2374)

[6] W. Thomson, E. Nordmark, T. Narten, „Neighbor Discovery for IP Version 6 (IPv6)", December 1998 (RFC 2461)

[7] W. Thomson, T. Narten, „IPv6 Stateless Address Autoconfiguration", December 1998 (RFC 2462)

[8] R. Hinden, S. Deering, „IP Version 6 Addressing Architecture", July 1998 (RFC 2373)

[9] Alexey N. Kuznetsov, „IP Command Reference", April 14, 1999

[10] Olaf Kirch, „The Network Administrators' Guide"

[11] include/linux/rtnetlink.h

[12] F. Baker, „Requirements for IP Version 4 Routers", June 1995, (RFC 1812)

[13] H. Endre, „CPU consumption of Linux firewalls", June 1999, *http://dawn.elte.hu/ endre/meres/index-en.rxml*

UWAGA: obecna wersja ma jeszcze masę błędów, niedoróbek i nieścisłości, więc proszę czytać ją z krytycznym nastawieniem i nie wierzyć we wszystko co napisałem.

Uwagi, poprawki i rozszerzenia mile widziane.