Kształtowanie Ruchu i Zaawansowany Routing HOWTO

Bert Hubert
Netherlabs BV

bert.hubert%netherlabs.nl

Thomas Graf (Autor sekcji)

tgraf%suug.ch

Gregory Maxwell (Autor sekcji)
Remco van Mook (Autor sekcji)

remco%virtu.nl

Martijn van Oosterhout (Autor sekcji)

kleptog%cupid.suninternet.com

Paul B Schroeder (Autor sekcji)

paulsch%us.ibm.com

Jasper Spaans (Autor sekcji)

jasper%spaans.ds9a.nl

Pedro Larroy (Autor sekcji)

piotr%omega.resa.es

Polskie tłumaczenie: Łukasz Bromirski

lbromirski%mr0vka.eu.org

Kształtowanie Ruchu i Zaawansowany Routing HOWTO

Bert Hubert
Thomas Graf (Autor sekcji)

tgraf%suug.ch

Gregory Maxwell (Autor sekcji)
Remco van Mook (Autor sekcji)

remco%virtu.nl

Martijn van Oosterhout (Autor sekcji)

kleptog%cupid.suninternet.com

Paul B Schroeder (Autor sekcji)

paulsch%us.ibm.com

Jasper Spaans (Autor sekcji)

jasper%spaans.ds9a.nl

Pedro Larroy (Autor sekcji)

piotr%omega.resa.es

Polskie tłumaczenie: Łukasz Bromirski

lbromirski%mr0vka.eu.org

Bardzo praktyczne podejście do iproute2, kontroli ruchu oraz po części do netfilter.

Historia zmian
Zmiana $Revision: 1.35 $ $Date: 2003/07/26 19:57:49 $
Wydanie DocBook

Spis treści

Rozdział 1. Dedykacja

Dokument ten dedykowany jest całemu mnóstwu ludzi i jest moją próbą dania czegoś w zamian. By wymienić zaledwie parę osób:

- Rusty Russell
- Aleksiej N. Kuzniecow
- dobrzy ludzie z Google
- Załoga Casema Internet

Rozdział 2. Wprowadzenie

Witam, Szanowny Czytelniku.

Ten dokument ma nadzieję rozjaśnić trochę zagadnienia routingu w Linuksie 2.2/2.4. Większość użytkowników nie zdaje sobie nawet sprawy, że posługuje się narzędziami, które potrafią naprawdę spektakularne rzeczy. Komendy takie jak **route** czy **ifconfig** są zaledwie namiastkami rozbudowanej i potężnej infrastrukturę iproute2.

Mam nadzieję, że to HOWTO będzie tak czytelne jak te napisane przez Rusty Russell'a z zespołu netfilter.

Możesz zawsze skontaktować się z nami, pisząc na adres zespołu HOWTO (mailto:HOWTO%ds9a.nl). Prosimy jednak byś rozważył wysłanie postu na listę pocztową (zajrzyj do odpowiedniej sekcji) jeśli masz pytania nie dotyczące bezpośrednio tego HOWTO. Nie stanowimy darmowego helpdesku, ale zwykle odpowiadamy na pytania zadawane na liście.

Zanim zgubisz się czytając ten dokument, a wszystko co chcesz robić to kształtowanie ruchu, pomiń wszystko i przejdL' bezpośrednio do rozdziału *Other possibilities* by zapoznać się z CBQ.init.

Disclaimer & Licencja

Dokument ten rozpowszechniany jest w nadziei, że będzie użyteczny, ale BEZ ŻADNEJ GWARANCJI; nawet bez implikowanej gwarancji RĘKOJMI lub PRZYDATNOŚCI DO KONKRETNEGO ZASTOSOWANIA.

Krótko mówiąc, jeśli twoja sieć STM-64 padnie albo posłuży do rozesłania pornografii do najbardziej cenionych klientów - to nie była nasza wina. Przykro nam.

Namawiamy do darmowego kopiowania i dystrybuowania (sprzedawania lub rozdawania) tego dokumentu w dowolnym formacie. Wymagamy jednak by poprawki i/lub komentarze przekazywać do koordynatora dokumentu.

Wymagamy również, żeby w przypadku publikacji tego HOWTO w wersji drukowanej, autorzy otrzymali próbki na „potrzeby recenzji" :-).

Wymagana wiedza

Tak jak wskazuje tytuł, to „Zaawansowane" HOWTO. O ile nie oznacza to w żadnym przypadku wiedzy z dziedziny technologii rakietowych, zakładamy że posiadasz pewną wiedzę.

Poniżej trochę odwołań, które mogą pomóc w nauczeniu się czegoś:

Zagadnienia sieciowe HOWTO, Rusty Russell'a (http://netfilter.samba.org/unreliable-guides/networking-concepts-HOWTO/index.html)

> Bardzo ładne wprowadzenie, wyjaśniające co to jest sieć i jak łączy się z innymi sieciami.

Sieć Linuksa (poprzednio Net-3 HOWTO)

> Doskonała rzecz, choć bardzo szczegółowa. Uczy wielu rzeczy, które są już skonfigurowane jeśli możesz połączyć się z Internetem. Powininno znajdować się w `/usr/doc/HOWTO/NET3-4-HOWTO.txt`, ale można je również znaleL'ć pod adresem online (http://www.linuxports.com/howto/networking).

Co Linux może zrobić dla Ciebie

Krótka lista rzeczy, które można uzyskać:

- kształtować pasmo sieciowe (ang. bandwidth) dla określonych komputerów
- kształtować pasmo sieciowe do określonych komputerów
- pomóc ci sprawiedliwie dzielić twoje pasmo sieciowe
- bronić sieć przed atakami typu DoS
- bronić Internet przed twoimi klientami
- wykorzystywać wiele serwerów jak jeden, by uzyskać równoważenie obciążenia i zwiększoną dostępność
- ograniczyć dostęp do twoich komputerów
- ograniczyć dostęp twoich użytkowników do innych komputerów
- prowadzić routing w oparciu o identyfikator użytkownika (tak!), adres MAC, L'ródłowy adres IP, port, typ usługi, czas dnia i zawartość

Obecnie niewiele osób używa tych zaawansowanych możliwości. Dzieje się tak z wielu powodów. O ile dokumentacja jest bardzo szczegółowa, nie jest zbyt przyjazna czy zrozumiała. A kształtowanie ruchu jest prawie w ogóle nieudokumentowane.

Notatki gospodyni

Warto zaznaczyć parę rzeczy dotyczących tego dokumentu. Mimo, że jestem autorem większości teksu, nie chcę by tak zostało. Jestem wielkim zwolennikiem ruchu Open Source, więc zachęcam do nadsyłania przemyśleń, uaktualnień, poprawek i tak dalej. Nie zwlekajcie z informowaniem mnie o literówkach lub po prostu błędach. Jeśli mój Angielski jest trochę drewniany, proszę zwrócić uwagę, że nie jest to mój język ojczysty. Zapraszam do nadsyłania sugestii.

Jeśli uważasz, że masz lepsze kwalifikacje by zajmować się którąś sekcją, lub myślisz że mógłbyś napisać a potem zajmować się nowymi sekcjami, zapraszam do zrobienia tego. Źródło tego dokumentu dostępne jest przez CVS w formacie SGML. Namawiam do dołączenia się do projektu.

By wam pomóc, znajdziecie tu wiele uwag FIXME. Poprawki są zawsze mile widziane! Za każdym razem gdy znajdziesz notatkę FIXME, powinieneś wiedzieć, że wchodzisz na nieznane terytorium. Nie chodzi o to, że wszędzie będą błędy, ale należy być uważnym. Jeśli udało Ci się coś potwierdzić lub wręcz przeciwnie - wskazać błąd, proszę, daj nam znać byśmy mogli popracować nad fragmentem zawierającym notkę FIXME.

W tym HOWTO pozwolę sobię na pewną swobodę. Na przykład, zakładam że posiadasz 10Mbit-owe połączenie z Internetem, mimo że nie jest to zbyt popularna konfiguracja.

Dostęp, CVS i wysyłanie uaktualnień

To HOWTO znajduje się pod tym adresem (http://www.ds9a.nl/lartc).

Na potrzeby projektu utrzymujemy serwer CVS z anonimowym dostępem z całego świata. Jest to przydatne z wielu powodów. Uaktalnienie twojej kopii do najnowszej wersji czy też wysłanie nam poprawek nie stanowi dzięki takiemu rozwiązaniu żadnego problemu.

Co więcej, system ten umożliwia wielu autorom na pracę nad tekstem L'ródłowym niezależnie.

```
$ export CVSROOT=:pserver:anon@outpost.ds9a.nl:/var/cvsroot
$ cvs login
CVS password: [wprowadL' 'cvs' (bez apostrofów)]
$ cvs co 2.4routing
cvs server: Updating 2.4routing
U 2.4routing/lartc.db
```

Jeśli zauważysz błąd, lub chciałbyś coś dodać, popraw to na swojej lokalnej kopii, następnie uruchom **`cvs -z3 diff -uBb`** i wyślij rezultat pod adres <howto%ds9a.nl> - łatwiej będzie nam się nim zająć. Dziękujemy! Przy okazji, upewnij się że edytowałeś plik .db - pozostałe są z niego generowane.

Razem z dokumentem L'ródłowym dostępny jest również plik Makefile, który powinien pomóc Ci wygenerować dokumenty w formacie postscript, dvi, pdf, html i czystym tekście. Być może będziesz musiał zainstalować docbook, docbook-utils, ghostscript oraz tetex by otrzymać wszystkie formaty.

Nie edytuj pliku 2.4routing.sgml! Zawiera starszą wersję tego HOWTO. Obecny plik zawierający HOWTO to lartc.db.

Lista pocztowa

Autorzy otrzymują rosnącą liczbę poczty dotyczącej tego HOWTO. Z uwagi na interes ogółu, zdecydowaliśmy o stworzeniu listy pocztowej, na której ludzie mogą dyskutować między sobą o zagadnieniach zaawansowanego routingu i kształtowania pasma. Możesz zapisać się na tą listę pod tym adresem (http://mailman.ds9a.nl/mailman/listinfo/lartc).

Należy zwrócić uwagę, że autorzy są bardzo powściągliwi jeśli chodzi o odpowiadanie na pytania nie zadane na liście. Chcielibyśmy ją archiwizować i utrzymywać jako swego rodzaju bazę wiedzy. Jeśli masz pytanie, proszę przeszukaj archiwum, a następnie wyślij post na listę.

Układ tego dokumentu

Zaczniemy robić interesujące rzeczy praktycznie zaraz, co oznacza, że na początku wiele rzeczy możesz uznać za słabo lub wcale nie wytłumaczone. Przeczytaj je zakładając, że wszystko stanie się jasne potem.

Routing i filtrowanie to dwie różne rzeczy. Filtrowanie zostało bardzo dobrze udokumentowane w HOWTO Rusty'ego, które dostępne są pod adresem:

- Rusty's Remarkably Unreliable Guides (http://netfilter.samba.org/unreliable-guides/)

My skupimy się głównie na pokazaniu co możliwe jest przy połączeniu infrastruktury netfilter i iproute2.

Rozdział 3. Wprowadzenie do iproute2

Dlaczego iproute2?

Większość dystrybucji Linuksa, i większość UNIXów, używa szacownych komend **arp**, **ifconfig** i **route**. O ile narzędzia te działają, powodują trochę nieoczekiwanych rezultatów od wersji Linuksa 2.2 w górę. Na przykład, tunele GRE są integralną częścią routingu a wymagają oddzielnych narzędzi.

Jeśli chodzi o iproute2, tunele są integralną częścią zestawu.

Kernele Linuksa od wersji 2.2 zawierają kompletnie przeprojektowany podsystem sieciowy. Wydajność nowego kodu i zestaw jego możliwości powoduje, że Linuks nie ma zbyt wielu konkurentów na arenie systemów operacyjnych. Tak naprawdę, nowy kod routujący, filtrujący i klasyfikujący jest potężniejszy niż ten dostarczany w większości dedykowanych routerów, ścian ogniowych i produktów zajmujących się kształtowaniem pasma.

W trakcie rozwijania nowych koncepcji sieciowych, ludzie znajdowali zawsze sposoby by dołożyć je do istniejących szkieletów już stworzonych systemów operacyjnych. To ciągłe dodawanie kolejnych warstw doprowadziło do tego, że kod sieciowy może czasami ciekawie się zachowywać lub wyglądać, tak jak to ma miejsce w przypadku większości ludzkich języków. W przeszłości, Linux emulował obsługę większości zagadnień sieciowych w sposób zgody z SunOS, który sam nie jest zbyt idealny.

Nowy szkielet umożliwia jasne wyrażanie potrzeb i pragnień niedostępnych wcześniej w Linuksie.

Przewodnik po iproute2

Linuks posiada bardzo wyrafinowany system kształtowania pasma, nazywany Traffic Control. System ten udostępnia wiele metod klasyfikowania, priorytetowania, współdzielenia i ograniczania zarówno ruchu wychodzącego jak i przychodzącego.

Zaczniemy od krótkiej wycieczki po możliwościach iproute2.

Wymagania wstępne

Powinieneś upewnić się, że wszystkie narzędzia z przestrzeni użytkownika są zainstalowane. Paczka nazywa się ’iproute’ zarówno w przypadku RedHat’a jak i Debiana, można ją znaleL’ć pod adresem `ftp://ftp.inr.ac.ru/ip-routing/iproute2-2.2.4-now-ss??????.tar.gz"`.

Możesz również spróbować pobrać i użyć najnowszą wersję, znajdującą się pod tym adresem (ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz)

Niektóre części iproute wymagają byś włączył pewne opcje w kernelu. Warto również zauważyć, że wszystkie wydania RedHat’a do i łącznie z wersją 6.2 dostarczane są bez większością opcji odpowiedzialnych za kontrolę ruchu.

RedHat 7.2 ma domyślnie włączone wszystko.

Upewnij się również, że w konfiguracji jądra zaznaczyłeś obsługę netlink, jeśli będziesz musiał przekompilować swój kernel. Jest on wymagany przez iproute2.

Badanie obecnej konfiguracji

Może to być pewnym zaskoczeniem, ale iproute2 jest już skonfigurowane! Obecne komendy **ifconfig** i **route** używają zaawansowanych wywołań systemowych, ale w większości z domyślnymi i bardzo bezpiecznymi (tzn. nudnymi) ustawieniami.

Głównym narzędziem jest program **ip**, poprosimy go o wyświetlenie interfejsów.

ip pokazuje nam połączenia

```
[ahu@home ahu]$ ip link list
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
```

Twój ekran wynikowy może się trochę różnić, ale powyższy wydruk pochodzi z mojego domowego routera NAT. Wytłumaczę tylko część, jako że nie wszystko jest bezpośrednio związane.

Na początek widzimy interfejs loopback. O ile twój komputer może i powinien pracować bez niego, radziłbym tego nie próbować. Rozmiar MTU (ang. Maximum Transfer Unit - Maksymalnej Jednostki Transmisji) wynosi 3924 oktety i nie spodziewamy się go kolejkować. Ma to sens, ponieważ interfejs ten jest tylko fantazją twojego kernela.

Pominę interfejs dummy, może on nie być obecny na twoim komputerze. Następnie mamy dwa fizyczne interfejsy, jeden po stronie modemu kablowego i drugi podłączony do mojego domowego segmentu ethernetowego. Dalej widzimy interfejs ppp0.

Zauważ brak adresów IP. iproute nie używa koncepcji łączenia ’połączeń’ z ’adresami IP’. Po wprowadzeniem aliasów IP, koncepcja ’tego konkretnego’ adresu IP stała się jakby mniej ważna.

Pokazano natomiast adresy MAC, identyfikatory sprzętowe naszych interfejsów ethernetowych.

ip pokazuje nam nasze adresy IP

```
[ahu@home ahu]$ ip address show
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
2: dummy: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1400 qdisc pfifo_fast qlen 100
    link/ether 48:54:e8:2a:47:16 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.1/8 brd 10.255.255.255 scope global eth0
4: eth1: <BROADCAST,MULTICAST,PROMISC,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/ether 00:e0:4c:39:24:78 brd ff:ff:ff:ff:ff:ff
3764: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 10
    link/ppp
    inet 212.64.94.251 peer 212.64.94.1/32 scope global ppp0
```

Tutaj mamy więcej informacji. Pokazano wszystkie nasze adresy, łącznie ze wskazaniem do których interfejsów należą. ’inet’ oznacza Internet (IPv4). Istnieje wiele innych rodzin adresów, ale w tym momencie zupełnie nas nie obchodzą.

Przyjrzyjmy się bliżej eth0. Twierdzi, że związany jest z internetowym adresem ’10.0.0.1/8’. Co to oznacza? Człon /8 określa ilość bitów, które należą do Adresu Sieci. Są 32 bity, mamy więc 24 bity na określenie części swojej sieci. Pierwsze 8 bitów z 10.0.0.1 odpowiada 10.0.0.0, naszemu Adresowi Sieciowemu, naszą maską jest więc 255.0.0.0.

Pozostałe bity należą do przestrzeni adresowej tego interfejsu, więc 10.250.3.13 jest bezpośrednio osiągalny przez eth0, tak samo jak na przykład 10.0.0.1.

Z ppp0 jest dokładnie tak samo, mimo że różnią się numerki. Jego adres to 212.64.94.251 bez maski podsieci. Oznacza to, że jest to połączenie punkt-punkt i każdy adres, z wyjątkiem 212.64.94.251 jest zdalny. Jest jednak trochę więcej informacji. Wiemy, że po drugiej stronie połączenia też jest jeden adres - 212.64.94.1. Dodatek /32 oznacza po prostu, że nie ma ’bitów sieci’.

Bardzo ważne żebyś zrozumiał opisywane wyżej zagadnienia. Zajrzyj do wspomnianej wcześniej dokumentacji, jeśli masz problemy.

Możesz również zauważyć ’qdisc’, co oznacza Dyscyplinę Kolejkowania (ang. Queueing Discipline). Znaczenie tej funkcji stanie się jasne póŁniej.

ip pokazuje nam trasy routingu

Cóż, wiemy jak znaleŁć adresy 10.x.y.z i jesteśmy w stanie dotrzeć do 212.64.94.1. To jednak nie wystarcza, musimy mieć jeszcze instrukcje jak dotrzeć do pozostałej części świata. Internet dostępny jest przez nasze połączenie PPP i wygląda na to, że to 212.64.94.1 będzie rozsyłał nasze pakiety po świecie, oraz zbierał i dostarczał odpowiedzi.

```
[ahu@home ahu]$ ip route show
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Wydruk jest samoopisujący się. Pierwsze 4 linie wprost informują o tym, co już zostało wydrukowane po użyciu komendy **ip address show**, ostatnia linia określa, że reszta adresów będzie osiągalna przez adres 212.64.94.1 - domyślną bramkę. Wiemy, że to bramka po słowie via, które oznacza, że wysyłamy pod ten adres pakiety a on bierze na siebie resztę.

Dla porównania, poniżej wydruk ze standardowej komendy **route**:

```
[ahu@home ahu]$ route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
212.64.94.1     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
10.0.0.0        0.0.0.0         255.0.0.0       U     0      0        0 eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         212.64.94.1     0.0.0.0         UG    0      0        0 ppp0
```

ARP

ARP, to Protokół Rozwiązywania Adresów (ang. Address Resolution Protocol), opisany w RFC 826 (http://www.faqs.org/rfcs/rfc826.html). ARP używany jest w komputerach podłączonych do sieci do rozwiązywania adresów sieciowych/lokalizacji innych komputerów również podłączonych do tej sieci. Komputery w Internecie identyfikuje się generalnie po ich nazwach, które rozwiązywane są na adresy IP. W ten właśnie sposób komputer w sieci foo.com może komunikować się z inną maszyną z sieci bar.net. Adresy IP nie mówią jednak nic o fizycznej lokalizacji komputera. Wtedy właśnie pojawia się ARP.

Spójrzmy na bardzo prosty przykład. Załóżmy, że mam sieć złożoną z kilku komputerów. Dwa z komputerów w tej sieci, to foo z adresem 10.0.0.1 i bar> z adresem 10.0.0.2. foo chce wykonać ping do bar, by sprawdzić czy bar pracuje, ale nie wie przecież gdzie znajduje się bar. Chcąc wykonać ping, musi zatem najpierw wysłać zapytanie ARP.

Zapytanie to podobne jest do wykrzyczenia ’bar (10.0.0.2)! Gdzie jesteś?’. W rezultacie wszystkie komputery w sieci usłyszą zapytanie, ale tylko bar (10.0.0.2) odpowie. OdpowiedL’ ARP zostanie skierowana bezpośrednio do foo i będzie czymś w rodzaju ’foo (10.0.0.1), jestem pod 00:60:94:E9:08:12’. Po tej prostej wymianie, której użyto by znaleL’ć kolegę w sieci, foo będzie w stanie komunikować się z bar dopóki zapomni (lub pamięć podręczna arp wygaśnie) gdzie jest bar> (na maszynach Uniksowych standardowo po 15 minutach).

A teraz spójrzmy jak to działa. Możesz obejrzeć tabelę sąsiedztwa arp w ten sposób:

```
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

Jak widać, mój komputer espa041 (9.3.76.41) wie gdzie znaleL’ć espa042 (9.3.76.42) i espagate (9.3.76.1). Spróbujmy teraz dodać inny komputer do pamięci podręcznej arp.

```
[root@espa041 /home/paulsch/.gnome-desktop]# ping -c 1 espa043
PING espa043.austin.ibm.com (9.3.76.43) from 9.3.76.41 : 56(84) bytes of data.
64 bytes from 9.3.76.43: icmp_seq=0 ttl=255 time=0.9 ms

--- espa043.austin.ibm.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.9/0.9/0.9 ms

[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0 lladdr 00:06:29:21:80:20 nud reachable
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud reachable
```

W wyniku działań espa041, próbującego znaleL’ć espa043, adres tego ostatniego został dodany do pamięci podręcznej arp. Dopóki ten wpis nie wygaśnie (w rezultacie braku komunikacji pomiędzy obydwoma komputerami), espa041 wie gdzie znaleL’ć espa043 i nie ma potrzeby rozsyłania zapytań ARP.

Teraz wykasujmy espa043 z pamięci podręcznej arp:

```
[root@espa041 /home/src/iputils]# ip neigh delete 9.3.76.43 dev eth0
[root@espa041 /home/src/iputils]# ip neigh show
9.3.76.43 dev eth0  nud failed
9.3.76.42 dev eth0 lladdr 00:60:08:3f:e9:f9 nud reachable
9.3.76.1 dev eth0 lladdr 00:06:29:21:73:c8 nud stale
```

espa041 zapomniał gdzie znaleL’ć espa043 i będzie musiał wysłać kolejne zapytanie ARP jeśli będzie potrzebował skontaktować się z espa043. Możesz również zauważyć, że stan wpisu przy espagate (9.3.76.1) został zmieniony na ’stale’ (niepewny, stary). Oznacza to tylko tyle, że pokazywana lokalizacja jest prawdziwa, ale będzie musiała być potwierdzona przed nawiązaniem jakiegokolwiek połączenia do tego komputera.

Rozdział 4. Reguły - baza danych polityki routingu

Jeśli posiadasz duży ruter, możesz zaspokajać potrzeby różnych ludzi, którzy chcą być obsługiwani w różny sposób. Pozwala na to właśnie baza danych polityki routingu, dostarczając możliwości utrzymywania wielu zestawów tabel routingu.

Jeśli chcesz użyć tej opcji, upewnij się że twój kernel skompilowano z użyciem opcji "IP: advanced router" oraz "IP: policy routing".

Kiedy kernel musi podjąć decyzję o routingu, sprawdza z której tabeli ma skorzystać. Domyślnie, są trzy takie tabele. Stare narzędzie ’route’ modyfikuje główną i lokalną, dokładnie tak, jak domyślnie zachowuje się nowe narzędzie ip.

Domyślne reguły:

```
[ahu@home ahu]$ ip rule list
0:       from all lookup local
32766:   from all lookup main
32767:   from all lookup default
```

Polecenie listuje priorytety wszystkich reguł. Widać, że wszystkie wpisy dotyczą wszystkich pakietów (’from all’). Widzieliśmy już ’główną’ (ang. ’main’) tabelę wcześniej, można ją wylistować przez użycie polecenia **`ip route ls`**, ale ’lokalna’ (ang. ’local’) i ’domyślna’ (ang. ’default’) są nowe.

Jeśli chcemy robić wymyślne rzeczy, musimy wygenerować reguły, które wskazywać będą do różnych tabel routingu - innych niż podstawowe, wspólne dla całego systemu.

Po dokładny opis tego co kernel robi gdy jest więcej pasujących reguł, odsyłam do dokumentacji ip-cref Aleksieja.

Prosty routing na podstawie L’ródła (ang. source routing)

Wróćmy do prawdziwego przykładu. Mam 2 (dokładniej 3, w momencie gdy je zwracałem) modemy kablowe połączone do linuksowego routera NAT (wykonującego translację adresów sieciowych). Ludzie którzy wokół żyją, płacą mi za używanie Internetu. Powiedzmy, że jeden z nich odwiedza tylko hotmail i chce płacić mniej. Jeśli chodzi o mnie jest to w porządku, ale będą używać starszego modemu.

’Szybki’ modem kablowy znany jest jako 212.64.94.251 i jest łączem PPP do 212.64.94.1. ’Wolny’ modem ma różne adresy - z uwagi na ten przykład skupimy się na 212.64.78.148 i jest podłączony do 195.96.98.253.

Tabela lokalna:

```
[ahu@home ahu]$ ip route list table local
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
local 10.0.0.1 dev eth0  proto kernel  scope host  src 10.0.0.1
broadcast 10.0.0.0 dev eth0  proto kernel  scope link  src 10.0.0.1
local 212.64.94.251 dev ppp0  proto kernel  scope host  src 212.64.94.251
broadcast 10.255.255.255 dev eth0  proto kernel  scope link  src 10.0.0.1
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 212.64.78.148 dev ppp2  proto kernel  scope host  src 212.64.78.148
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
```

Widać wiele oczywistych rzeczy, ale gdzieś należy je podać. Cóż, znalazły się tutaj. Tabela domyślna jest pusta.

Zajrzyjmy do tabeli głównej:

```
[ahu@home ahu]$ ip route list table main
195.96.98.253 dev ppp2  proto kernel  scope link  src 212.64.78.148
```

```
212.64.94.1 dev ppp0  proto kernel  scope link  src 212.64.94.251
10.0.0.0/8 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 212.64.94.1 dev ppp0
```

Generujemy teraz nową regułę, którą nazwiemy ‘John’, tak jak nasz hipotetyczny kolega. Możemy oczywiście operować czystymi numerkami, ale łatwiej jest dodać nasze tabele do pliku `/etc/iproute2/rt_tables`.

```
# echo 200 John >> /etc/iproute2/rt_tables
# ip rule add from 10.0.0.10 table John
# ip rule ls
0:      from all lookup local
32765:      from 10.0.0.10 lookup John
32766:      from all lookup main
32767:      from all lookup default
```

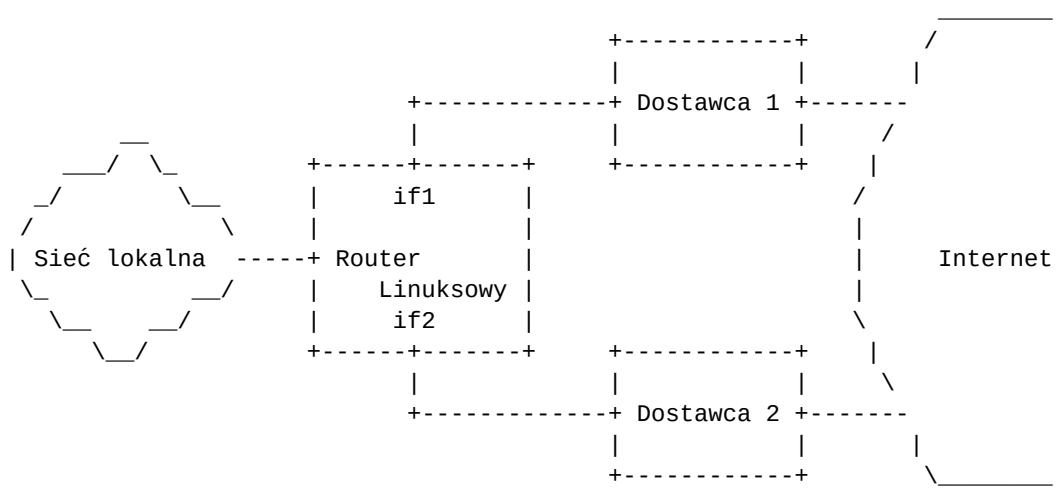
Teraz pozostało tylko wygenerować tabelę Johna i wyczyścić pamięć podręczną trasowania:

```
# ip route add default via 195.96.98.253 dev ppp2 table John
# ip route flush cache
```

I to wszystko. Pozostawiamy jako ćwiczenie dla czytelnika, zaimplementowanie tego w skrypcie `ip-up`.

Routing dla wielu łącz/do wielu dostawców usług

Często spotyka się konfiugrację taką jak poniższa - w której mamy dwóch dostawców Internetu dla sieci lokalnej (lub czasami nawet pojedyńczej maszyny).



Zwykle w takiej konfiguracji pojawiają się dwa pytania.

Dostęp rozłączny (ang. split access)

Pierwsze to jak routować odpowiedzi na pakiety przychodzące od powiedzmy dostawcy pierwszego, z powrotem tą samą drogą (nie wykorzystując drugiego łącza).

Zdefiniujmy parę nazw symbolicznych. Niech **$IF1** oznacza pierwszy interfejs (if1 na rysunku powyżej) a **$IF2** drugi interfejs. Następnie niech **$IP1** będzie adresem IP interfejsu **$IF1** a **$IP2** adresem IP skojarzonym z

interfejsem **$IF2**. Następnie, niech **$P1** oznacza adres IP pierwszego dostawcy a **$P2** drugiego. Na koniec niech **$P1_NET** będzie siecią IP w której znajduje się **$P1**, a **$P2_NET** odpowiednio siecią IP w której jest **$P2**.

Należy stworzyć dwa dodatkowe wpisy w tabeli routingu, powiedzmy **T1** i **T2**. Dodaje się je do `/etc/iproute2/rt_tables`. Następnie należy dodać odpowiednio routing w sposób jak poniżej:

```
ip route add $P1_NET dev $IF1 src $IP1 table T1
ip route add default via $P1 table T1
ip route add $P2_NET dev $IF2 src $IP2 table T2
ip route add default via $P2 table T2
```

Nic spektakularnego, tworzymy trasy do bramy i dodajemy trasę domyślną przez tą bramę - tak jakbyśmy mieli do czynienia z pojedyńczym dostawcą, z tym wyjątkiem, że mamy osobne tablice dla każdego dostawcy. Zauważ, że wystarczy samo wpisanie trasy - ponieważ mówi jak znaleL’ć dowolny host w danej sieci, obejmując również bramę tak jak w przykładzie powyżej.

Teraz należy skonfigurować główną tabelę routingu. Dobrym pomysłem jest kierować ruch do bezpośredniego sąsiada przez interfejs, którym ten sąsiad jest podłączony. Zwróć uwagę na argumenty ’src’ - zapewniają że wybrany zostanie poprawny wychodzący adres IP.

```
ip route add $P1_NET dev $IF1 src $IP1
ip route add $P2_NET dev $IF2 src $IP2
```

Następnie należy dodać trasę domyślną:

```
ip route add default via $P1
```

A na koniec skonfigurować tabele routingu. To one wybierają która tablica zostanie wybrana. Chcemy się upewnić, że wykonujemy routing interfejsem, który ma odpowiedni L’ródłowy adres IP:

```
ip rule add from $IP1 table T1
ip rule add from $IP2 table T2
```

Powyższy zestaw poleceń powoduje, że wszystkie odpowiedzi na ruch przechodzący przez dany interfejs, zostaną wysłane tym właśnie interfejsem.

Ostrzeżenie

Czytelnik Rod Roark zauważył: Jeśli $P0_NET to lokalna sieć a $IF0 to jej interfejs, warto dodać dodatkowe wpisy:

```
ip route add $P0_NET     dev $IF0 table T1
ip route add $P2_NET     dev $IF2 table T1
ip route add 127.0.0.0/8 dev lo   table T1
ip route add $P0_NET     dev $IF0 table T2
ip route add $P1_NET     dev $IF1 table T2
ip route add 127.0.0.0/8 dev lo   table T2
```

,

Jest to oczywiście bardzo prosta konfiguracja. Będzie działała prawidłowo dla wszystkich procesów działających na routerze i dla wszystkich połączeń z sieci lokalnej (jeśli włączono translację adresów). Jeśli translacji nie

włączono, to albo dostałeś osobne publiczne przestrzenie adresowe od obu dostawców, albo będziesz chciał robić translację adresów dla jednego z dostawców. W obu przypadkach musisz dodać reguły wybierające przez którego dostawcę routować ruch na podstawie adresów IP maszyn w sieci lokalnej (inicjujących ten ruch).

Rozkładanie obciążenia (ang. load balancing)

Drugie pytanie brzmi: jak rozłożyć obciążenie wychodzące na obu dostawców. Odpowiedzią jest: nie jest to specjalnie trudne, jeśli i tak masz już sytuację opisaną w punkcie powyżej.

Zamiast wybierać jednego z dostawców jako trasę domyślną, musisz wskazać obu. Domyślnie jądro będzie rozkładało ruch wychodzący pomiędzy dwie trasy domyślne. Konfiguruje się to w ten sposób:

```
ip route add default scope global nexthop via $P1 dev $IF1 weight 1 \
nexthop via $P2 dev $IF2 weight 1
```

Zapewni to opisaną funkcjonalność. Argumentem **weight** można dokładnie dostosować metrykę danego dostawcy, efektywnie zwiększając lub zmniejszając ilość ruchu przesyłanego przez niego na rzecz drugiego dostawcy.

Zwróć jednak uwagę, że równoważenie obciążenia nie będzie doskonałe, a co więcej bazuje na trasach - które przechowywane są w pamięci podręcznej. Innymi słowy, trasy do najczęściej odwiedzanych stron będą prowadziły zawsze przez tego samego dostawcę.

I jeszcze jedno - jeśli naprawdę chcesz coś takiego zastosować, prawdopodobnie powinieneś rzucić okiem na łatki Juliana Anastazowa, znajdujące się pod adresem jego strony (http://www.ssi.bg/~ja/#routes), Powinny one ułatwić ci trochę konfigurację takiego podziału ruchu.

Rozdział 5. Tunele - GRE i inne

W Linuksie dostępne są trzy rodzaje tuneli. Mamy tunelowanie IP w IP, tunelowanie GRE i tunele żyjące poza jądrem (na przykład PPTP).

Parę ogólnych uwag o tunelach:

Tunele można użyć do bardzo niezwykłych i fajnych rzeczy. Mogą również spowodować koszmarne kłopoty jeśli nie skonfiguruje się ich poprawnie. Nie kieruj swojej domyślnej trasy na urządzenie tunelujące chyba, że *dokładnie* wiesz co robisz :-). Co więcej, tunelowanie zwiększa narzut na pakiety, ponieważ wymaga dodatkowego zestawu nagłówków IP. Oznacza to typowo około 20 bajtów na każdy pakiet, więc jeśli normalny rozmiar pakietu (MTU) w sieci to 1500 bajtów, pakiet wysyłany przez tunel może mieć tylko 1480 bajtów. Nie musi to okazać się od razu wielkim problemem, ale pomyśl o fragmentacji i składaniu pakietów jeśli chcesz łączyć duże sieci tunelami.

I jeszcze jedno - najszybszym sposobem wykopania tunelu jest kopanie po obu stronach.

Tunelowanie IP w IP

Ten rodzaj tunelowania jest dostępny w Linuksie już od bardzo dawna. Wymaga dwóch modułów kernela: `ipip.o` oraz `new_tunnel.o`.

Powiedzmy że masz 3 sieci: Wewnętrzne A i B i pośrednią C (na przykład Internet). Mamy więc sieć A:

```
network 10.0.1.0
netmask 255.255.255.0
router  10.0.1.1
```

W sieci C router ma adres 172.16.17.18.

Oraz sieć B:

```
network 10.0.2.0
netmask 255.255.255.0
router  10.0.2.1
```

W sieci C router ma adres 172.19.20.21.

Jeśli chodzi o sieć C, zakładamy że przekazuje pakiety od A do B i odwrotnie. Można do tego używać nawet Internetu.

Oto co trzeba zrobić:

Po pierwsze, upewnij się że zainstalowano moduły:

```
insmod ipip.o
insmod new_tunnel.o
```

Teraz, na routerze sieci A wykonaj:

```
ifconfig tunl0 10.0.1.1 pointopoint 172.19.20.21
route add -net 10.0.2.0 netmask 255.255.255.0 dev tunl0
```

A na routerze sieci B:

```
ifconfig tunl0 10.0.2.1 pointopoint 172.16.17.18
```

```
route add -net 10.0.1.0 netmask 255.255.255.0 dev tunl0
```

A gdy skończyłeś używać tunelu, wykonaj:

```
ifconfig tunl0 down
```

Presto, koniec. Nie możesz jednak przekazywać pakietów rozgłoszeniowych (ang. broadcasts) i ruchu IPv6 przez tunel IP-w-IP. Można tylko łączyć dwie sieci IPv4, które normalnie nie mogłyby ze sobą rozmawiać - to wszystko. Jeśli chodzi o zgodność w dół, kod znalazł się w jądrze w okolicach wersji 1.3 - jest już obecny zatem jakiś czas. Natomiast jeśli chodzi o inne systemy - ten rodzaj tunelowania nie działa z innymi systemami czy routerami - przynajmniej na tyle na ile wiem. Jest bardzo prosty, ale działa. Używaj jeśli musisz, w przeciwnym razie sięgnij po tunele GRE.

Tunelowanie GRE

Protokół tunelujący GRE został pierwotnie stworzony przez Cisco i jest w stanie zrealizować trochę więcej niż tunelowanie IP-w-IP. Na przykład, możesz również transportować ruch multicastowy i IPv6 przez tunel GRE.

W Linuksie, będziesz potrzebował modułu `ip_gre.o`.

Tunelowanie IPv4

Najpierw zajmijmy się tunelowaniem IPv4.

Powiedzmy że masz 3 sieci: Wewnętrzne A i B oraz pośrednią C (lub, powiedzmy Internet).

Mamy więc sieć A:

```
network 10.0.1.0
netmask 255.255.255.0
router  10.0.1.1
```

Router ma adres 172.16.17.18 w sieci C. Nazwijmy ją *neta* (dobra, strasznie oryginalne).

...sieć B:

```
network 10.0.2.0
netmask 255.255.255.0
router  10.0.2.1
```

Router ma adres 172.19.20.21 w sieci C. Nazwijmy ją *netb* (nadal mało oryginalne).

Natomiast jeśli chodzi o sieć C, zakładamy że przekazuje pakiety od A do B i odwrotnie. Jak i dlaczego nas nie interesuje.

Na routerze w sieci A wykonamy:

```
ip tunnel add netb mode gre remote 172.19.20.21 local 172.16.17.18 ttl 255
ip link set netb up
ip addr add 10.0.1.1 dev netb
ip route add 10.0.2.0/24 dev netb
```

Omówmy to w paru zdaniach. W linii 1 dodaliśmy urządzenie tunelujące i nazwaliśmy je *netb* (co jest raczej oczywiste, ponieważ to tam ma prowadzić). Co więcej, określiliśmy, że używamy protokołu GRE (mode gre), zdalny adres to 172.19.20.21 (router po drugiej stronie), nasze tunelowane pakiety powinny pochodzić z 172.16.17.18 (dzięki temu możesz mieć wiele interfejsów w sieci C i określić wprost z którego mają być wysyłane pakiety), oraz, że pole TTL pakietów powinno być ustawiane na 255 (ttl 255).

Druga linia uaktywnia urządzenie.

W trzeciej linii ustaliliśmy adres nowego urządzenia *netb* na 10.0.1.1. Dla małych sieci jest to w zasadzie w porządku, ale jeśli rozpoczynasz poważne kopanie (WIELE tuneli), powinieneś rozważyć użycie innego zakresu adresów IP dla interfejsów tunelujących (w naszym przykładzie, mógłbyś użyć 10.0.3.0).

W czwartej linii ustawiamy trasę dla sieci B. Zauważ inną notację w masce sieciowej. Jeśli nie jest ci ona znana, oto jak działa: zapisujesz maskę sieciową w formie binarnej i liczysz wszystkie jedynki. Jeśli nie wiesz jak to zrobić, zapamiętaj że 255.0.0.0 to /8, 255.255.0.0 to /16 a 255.255.255.0 to /24. A 255.255.254.0 to /23, gdybyś się zastanawiał.

Ale koniec tego, zacznijmy z routerem w sieci B.

```
ip tunnel add neta mode gre remote 172.16.17.18 local 172.19.20.21 ttl 255
ip link set neta up
ip addr add 10.0.2.1 dev neta
ip route add 10.0.1.0/24 dev neta
```

A jeśli zechciałbyś usunąć tunel na routerze A:

```
ip link set netb down
ip tunnel del netb
```

Oczywiście, możesz zastąpić netb przez neta dla routera B.

Tunelowanie IPv6

Zajrzyj do sekcji 6, by zapoznać się z krótkim wprowadzeniem do protokołu IPv6 i formacie zapisu adresów.

Do roboty z tunelami.

Zakładamy, że masz następującą sieć IPv6 i chcesz podłączyć się do 6bone lub kolegi.

```
Network 3ffe:406:5:1:5:a:2:1/96
```

Twój adres IPv4 to 172.16.17.18 a router IPv4 6bone ma adres 172.22.23.24.

```
ip tunnel add sixbone mode sit remote 172.22.23.24 local 172.16.17.18 ttl 255
ip link set sixbone up
ip addr add 3ffe:406:5:1:5:a:2:1/96 dev sixbone
ip route add 3ffe::/15 dev sixbone
```

Omówmy to. W pierwszej linii stworzyliśmy tunel nazwany *sixbone*. Nadaliśmy mu tryb pracy sit (co oznacza tunelowanie IPv6 w IPv4) oraz wskazaliśmy gdzie ma być skierowany (remote) i gdzie się zaczynać (local). TTL ustawione jest na maksymalną wartość 255. Następnie, podnieśliśmy interfejs (up). Na koniec dodaliśmy nasz własny adres sieciowy i ustawiliśmy trasę na 3ffe::/15 (co jest obecnie adresem całej 6bone) w tunelu.

Tunele GRE są obecnie preferowanym rodzajem tunelowania. Są standardowe i przyjęte również poza społecznością linuksową, a w związku z tym są Dobrą Rzeczą.

Tunele w przestrzeni użytkownika

Istnieją dziesiątki implementacji tunelowania poza kernelem. Najbardziej znane są oczywiście PPP i PPTP, ale jest ich dużo więcej (niektóre firmowe, niektóre bezpieczne, inne nie używają nawet IP) i są zdecydowanie poza tematem tego HOWTO.

Rozdział 6. Tunelowanie IPv6 z Cisco i/lub 6bone

Autorstwa Marco Davids <marco%sara.nl>

UWAGA do koordynatora:

Na tyle ile się orientuje tunelowanie IPv6-IPv4 nie jest tak naprawdę tunelowaniem GRE. Można tunelować IPv6 przez IPv4 w urządzeniach które tworzą tunel GRE (GRE tuneluje WSZYSTKO w IPv4), ale urządzenie używane w tym rozwiązaniu tuneluje tylko IPv6 przez IPv4 i w związku z tym to co innego.

Tunelowanie IPv6

To kolejne zastosowanie dla możliwości tunelowania Linuksa. Jest bardzo popularne u pierwszych użytkowników IPv6, lub pionierów - jak wolicie. Ten praktyczny przykład z pewnością nie opisuje jedynego możliwego rozwiązania tunelowania IPv6. Jest to jednak metoda używana często przy tunelowaniu Linuksów i Cisco używających IPv6, a z doświadczenia wynika, że w ten właśnie sposób realizuje to większość ludzi. Dziesięć do jednego, że dotyczy to również ciebie ;-).

Trochę o adresach IPv6:

Adresy IPv6 są, w porównaniu do IPv4 bardzo duże: 128 bitów zamiast 32. Daje nam to czego potrzebujemy: bardzo, bardzo dużo adresów IP: 340,282,266,920,938,463,463,374,607,431,768,211,465 jeśli chodzi o ścisłość. Oprócz tego, IPv6 (lub IPng, co oznacza IP Next Generation) ma zapewnić mniejsze tablice routingu na ruterach szkieletowych, prostszą konfigurację sprzętu, lepsze bezpieczeństwo na poziomie IP i lepsze wsparcie dla QoS.

Przykład: 2002:836b:9820:0000:0000:0000:836b:9886

Zapisanie adresu IPv6 może być dosyć kłopotliwe. By życie było prostsze, stworzono pewne reguły:

- Nie używaj wiodących zer. Tak jak w IPv4.
- Używaj dwukropków do oddzielania każdych 16 bitów, lub dwóch bajtów.
- Jeśli masz dużo kolejnych zer, możesz zapisać je jako :: - ale tylko raz i tylko dla wielokrotności 16 bitów.

Adres 2002:836b:9820:0000:0000:0000:836b:9886 może być więc zapisany jako 2002:836b:9820::836b:9886, co jest zdecydowanie przyjaL'niejsze.

Inny przykład: 3ffe:0000:0000:0000:0000:0020:34A1:F32C może być zapisane jako 3ffe::20:34A1:F32C, co jest dużo krótsze.

IPv6 ma zastąpić obecnie używany IPv4. Ponieważ jest to relatywnie nowa technologia, nie ma jeszcze światowej natywnej sieci IPv6. By można było przenosić się płynnie, wprowadzono 6bone.

Sieci używające IPv6 połączone są pomiędzy sobą przez hermetyzację protokołu IPv6 w ramki IPv4 i wysyłanie tak spreparowanych pakietów przez istniejącą infrastrukturę IPv4 z jednej sieci IPv6 do innej.

Miejsce styku obu sieci jest dokładnie miejscem, w którym używamy tuneli.

By móc używać IPv6 musimy mieć kernel, który obsługuje ten protokół. Jest wiele bardzo dobrych dokumentów jak dojść do takiego stanu. Wszystko sprowadza się jednak do paru kroków:

- ZdobądL' którąś z nowszych dystrybucji Linuksa z odpowiednią wersją biblioteki glibc.
- ZdobądL' nowe L'ródła kernela.

Jeśli wszystko jest gotowe, to możesz skompilować nowy kernel z obsługą IPv6:

- PrzejdL' do `/usr/src/linux` i napisz:
- `make menuconfig`
- Wybierz "Networking Options"
- Wybierz "The IPv6 protocol", "IPv6: enable EUI-64 token format" i "IPv6: disable provider based addresses"

PODPOWIEDŹ: Nie konfiguruj tych opcji jako znajdujących się w modułach. Bardzo często takie rozwiązanie po prostu nie działa.

Innymi słowy, wkompiluj IPv6 w kernel. Możesz następnie zapisać swoją konfigurację tak jak zwykle i skompilować kernel.

PODPOWIEDŹ: Zanim zaczniesz to robić, zastanów się nad modyfikacją pliku `Makefile`: EXTRAVERSION = -x ; --> ; EXTRAVERSION = -x-IPv6

Napisano bardzo dużo dobrej dokumentacji dotyczącej kompilowania i instalowania jądra, ale ten dokument jest o czym innym. Jeśli w tym momencie wpadłeś w jakieś problemy, poszukaj czegoś na ten temat w innych dokumentach.

Plik `/usr/src/linux/README` może być dobrym początkiem. Gdy uda ci się kompilacja i uruchomiłeś ponownie swój komputer z nowym kernelem, możesz sprawdzić przez '/sbin/ifconfig -a' czy pojawiło się nowe urządzenie 'sit0-device'. SIT oznacza Proste Przejście do Internetu (ang. Simple Internet Transition). Możesz sobie pogratulować - jesteś o jeden krok bliżej do IP Nowej Generacji ;-).

Teraz drugi krok. Chcesz połączyć twoj komputer lub być może całą sieć LAN do innej sieci IPv6. Może to być właśnie "6bone", którą stworzono właśnie w tym celu.

Załóżmy, że masz następującą sieć IPv6: 3ffe:604:6:8::/64 i że chcesz połączyć się do 6bone lub kolegi. Zauważ notację /64, która działa dokładnie tak samo jak w zwykłym adresie IP.

Masz adres IPv4 145.100.24.181 a router 6bone ma adres 145.100.1.5.

```
# ip tunnel add sixbone mode sit remote 145.100.1.5 [local 145.100.24.181 ttl 255]
# ip link set sixbone up
# ip addr add 3FFE:604:6:7::2/126 dev sixbone
# ip route add 3ffe::0/16 dev sixbone
```

Omówmy to. W pierwszej linii tworzymy tunel nazwany *sixbone*. PóL'niej ustawiamy go w tryb sit, mówimy gdzie ma być skierowany (remote) i gdzie się zaczynać (local). TTL ustawione jest na wartość maksymalną - 255.

Następnie podnosimy nasze urządzenie (up). Potem dodajemy adres naszej sieci i ustawiamy trasę dla 3ffe::/15 (które stanowi obecnie całe 6bone) przez tunel. Jeśli ta konkretna maszyna na której wpisujesz te komendy jest twoją bramką do IPv6, rozważ dodanie następujących linii:

```
# echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
# /usr/local/sbin/radvd
```

Drugi program, radvd, jest podobnie jak zebra demonem rozgłaszającym routera i zapewnia wsparcie dla opcji autokonfiguracyjnych IPv6. Poszukaj go ulubioną wyszukiwarką, jeśli chciałbyś go używać. Możesz sprawdzić rzeczy takie jak np.:

```
# /sbin/ip -f inet6 addr
```

Jeśli radvd pracuje na bramce IPv6 a twój Linux jest w lokalnym LANie, będziesz mógł cieszyć się autokonfiguracją IPv6:

```
# /sbin/ip -f inet6 addr
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue inet6 ::1/128 scope host

3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 100
```

```
inet6 3ffe:604:6:8:5054:4cff:fe01:e3d6/64 scope global dynamic
valid_lft forever preferred_lft 604646sec inet6 fe80::5054:4cff:fe01:e3d6/10
scope link
```

Powinieneś teraz skonfigurować swojego binda na obsługę adresów IPv6. Rekordowi typu A w IPv4 odpowiada w IPv6 AAAA. Natomiast wpisowi in-addr.arpa odpowiada teraz ip6.int. Jest bardzo dużo informacji dostępnej na ten temat.

Dostępne są stale rosnące ilości aplikacji obsługujących IPv6, łącznie z bezpieczną powłoką (ssh), telnetem, inetd, przeglądarką Mozilla, serwerem WWW Apache i innymi. Ale ponownie - zakres aplikacji wspierających IPv6 wykracza daleko poza ten dokument ;-).

W konfiguracji routera Cisco należy wpisać coś takiego:

```
!
interface Tunnel1
 description tunel IPv6
 no ip address
 no ip directed-broadcast
 ipv6 address 3FFE:604:6:7::1/126
 tunnel source Serial0
 tunnel destination 145.100.24.181
 tunnel mode ipv6ip
!
ipv6 route 3FFE:604:6:8::/64 Tunnel1
```

Jeśli nie masz do swojej dyspozycji Cisco, popytaj innych użytkowników IPv6 w Internecie. Być może będą chcieli skonfigurować swoje Cisco z twoim tunelem, zwykle w ramach przyjaznego interfejsu WWW. Szukaj ciągu znaków "ipv6 tunnel broker" w ulubionej wyszukiwarce.

Rozdział 7. IPsec: bezpieczne IP przez Internet

W Linuksie dostępne są dwie implementacje IPsec. Dla kerneli 2.2 i 2.4 istnieje FreeS/WAN, który był pierwszym kompletnym rozwiązaniem. Oficjalna strona znajduje się pod tym adresem (http://www.freeswan.org/), natomiast nieoficjalna pod tym (http://www.freeswan.ca). FreeS/WAN tradycyjnie nie był włączony do L'ródeł jądra z paru powodów. Zwykle wymienia się 'polityczne' - związane z Amerykanami pracującymi nad kodem odpowiedzialnym za szyfrowanie. Co więcej, niezbyt dobrze integrował się z jądrem i w związku z tym nie był dobrym kandydatem.

Dodatkowo wiele (http://www.edlug.ed.ac.uk/archive/Sep2002/msg00244.html) niezależnych osób (http://lists.freeswan.org/pipermail/design/2002-November/003901.html) wyrażało się niezbyt pochlebnie o jakości kodu. Do konfiguracja FreeS/WAN dostępne jest natomiast dużo (http://www.freeswan.ca/docs/freeswan-1.99/doc/index.html) rozmaitej dokumentacji (http://www.freeswan.org/doc.html).

Od wersji 2.5.47 dostępna jest natywna implementacja IPsec. Została napisana przez Aleksieja Kuzniecowa i Dave Millera, a jej konstrukcja inspirowana była pracą grupy USAGI IPv6. Jednocześnie w tej wersji dołączono kod Jamesa Morrisa - CryptoAPI.

Ten dokument opisuje tylko wersje 2.5+ IPsec. Dla jąder serii 2.4 zalecany jest obecnie FreeS/WAN, ale zwróć uwagę że jego konfiguracja różni się od natywnej implementacji IPsec. Dostępne są obecnie łatki (http://gondor.apana.org.au/~herbert/freeswan/) umożliwiające współpracę kodu FreeS/WAN z natywną implementacją IPsec.

Od wersji 2.5.49, implementacja IPsec działa bez żadnych dodatkowych łatek.

> **Notatka:** Narzędzia działające w przestrzeni użytkownika dostępne są tutaj (http://sourceforge.net/projects/ipsec-tools) - i są aktywnie rozwijane.
>
> Podczas kompilacji jądra, upewnij się że włączyłeś 'PF_KEY', 'AH', 'ESP' oraz wszystko w CryptoAPI! Natomiast cel 'TCP_MSS' z netfilter aktualnie nie działa i powinieneś go wyłączyć.

Ostrzeżenie

Autor tego rozdziału jest zupełnym początkującym jeśli chodzi o IPsec. Jeśli znalazłeś błędy, proszę napisz do Berta Huberta <ahu%ds9a.nl>.

Na początek, pokażemy jak ręcznie skonfigurować bezpieczny kanał komunikacyjny pomiędzy dwoma hostami. Większość tego procesu można zautomatyzować, ale zrobimy to ręcznie by zapoznać się z tym, co dzieje się 'pod maską'.

Możesz pominąć tą sekcję, jeśli interesujesz się tylko automatyczną wymianą kluczy, ale weL' pod uwagę że lepiej wiedzieć o tym procesie coś więcej.

Wprowadzenie do Ręcznej Wymiany Kluczy

IPsec to skomplikowany zestaw protokołów. Bardzo dużo materiału dostępnego jest online, natomiast to HOWTO skupi się tylko na uruchomieniu podstawowej funkcjonalności.

Notatka: Wiele konfiguracji używających iptables odrzuca pakiety IPsec! By je przepuszczać dodaj: 'iptables -A xxx -p 50 -j ACCEPT' and 'iptables -A xxx -p 51 -j ACCEPT'

IPsec oferuje bezpieczną wersję IP. Bezpieczeństwo rozumiane jest tutaj jako dwie różne rzeczy: szyfrowanie i uwierzytelnianie. Naiwna wersja bezpieczeństwa oferuje tylko szyfrowanie, ale bardzo łatwo można pokazać, że to nie wystarczy - możesz komunikować się w sposób szyfrowany, ale nie masz gwarancji że to co oferujesz drugiej stronie jest tym, czym spodziewasz się że będzie.

IPsec wspiera 'Encapsulated Security Payload' (ESP) który służy do szyfrowania oraz 'Authentication Header' (AH) dla uwierzytelniania zdalnego partnera. Możesz użyć obu z nich, lub zdecydować się na zastosowanie tylko jednego.

Zarówno ESP i AH opierają się o tzw. związki bezpieczeństwa (ang. „security association", SA). SA składają się ze wskazania L'ródła ruchu, miejsca docelowego dla ruchu oraz instrukcji jak przesyłany ruch potraktować. Przykładowa SA uwierzytelniająca pokazana jest poniżej:

```
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
```

Wpis ten oznacza 'ruch z 10.0.0.11 do 10.0.0.216 wymagający zastosowania AH, należy podpisać używając algorytmu HMAC-MD5 z kluczem 1234567890123456'. Instrukcja ta posiada przydzielony indeks parametrów bezpieczeństwa (ang. „Security Parameter Index", SPI) 15700. Interesującą cechą SA jest to, że są symetryczne. Obie strony muszą posiadać dokładnie te same SA - nie są one automatycznie kopiowane do partnera. Co więcej, pojedyńczy wpis SA opisuje ruch tylko w jednym kierunku. Aby obsłużyć ruch powrotny, należy stworzyć dwie SA.

Przykładowe SA ESP:

```
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
```

Ten z kolei wpis oznacza 'ruch z 10.0.0.11 do 10.0.0.216 wymagający szyfrowania należy szyfrować algorytmem 3DES-CBC z kluczem 123456789012123456789012'. SPI dla wpisu to 15701.

Udało nam się stworzyć SA opisujące różne możliwe instrukcje, ale nie wskazaliśmy w żadnym momencie w którym momencie należy tych instrukcji użyć. Tak naprawdę, można stworzyć wiele dokładnie takich samych SA różniących się tylko numerami SPI. Aby wykonać faktycznie szyfrowanie, należy dodatkowo zdefiniować regułę. Może ona na przykład mówić 'użyj IPsec jeśli jest dostępny' lub 'odrzucaj ruch chyba że otrzymamy IPsec'.

Typowa, przykładowa Reguła Bezpieczeństwa (ang. „Security Policy", SP) wyglądać może tak:

```
spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
 esp/transport//require
 ah/transport//require;
```

Jeśli została wpisana na hoście 10.0.0.216, oznacza dokładnie tyle, że cały ruch wychodzący do 10.0.0.11 ma być szyfrowany i opakowany w nagłówek AH. Zauważ że nie wskazaliśmy które SA mają być zastosowane, ma to sprawdzić kernel.

Innymi słowy, SP opisuje CO chcemy uzyskać, a SA JAK chcemy to zrobić.

Wychodzące pakiety otrzymują etykiety z SA SPI, tak by host otrzymujący ruch mógł sprawdzić swoje wpisy i wykonać odpowiednie operacje odszyfrowując ruch.

Poniżej bardzo prosty przykład konfiguracji, dla połączenia 10.0.0.216 z 10.0.0.11 z użyciem uwierzytelniania i szyfrowania. WeL' pod uwagę że ruch z powrotem odbywać się będzie bez szyfrowania i taka konfiguracja nie powinna być stosowana.

Na hoście 10.0.0.216:

```
#!/sbin/setkey -f
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
   esp/transport//require
   ah/transport//require;
```

Na hoście 10.0.0.11 te same SA, bez SP:

```
#!/sbin/setkey -f
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";
```

W powyższej konfiguracji (pliki można wykonać jeśli 'setkey' znajduje się w katalogu /sbin) polecenie 'ping 10.0.0.11' wykonane na hoście 10.0.0.216 wyglądać powinno tak, jeśli podsłuchamy je programem tcpdump:

```
22:37:52 10.0.0.216 > 10.0.0.11: AH(spi=0x00005fb4,seq=0xa): ESP(spi=0x00005fb5,seq=0xa) (DF)
22:37:52 10.0.0.11 > 10.0.0.216: icmp: echo reply
```

Zauważ, że odpowiedL' na ping z 10.0.0.11 jest faktycznie widoczna nieszyfrowana. Natomiast sam ping widziany jest przez tcpdump jako transmisja IPsec (używająca AH i ESP).

Należy zwrócić uwagę na parę rzeczy. Konfiguracja powyżej pokazywana jest w wielu przykładach dotyczących IPsec i jest bardzo niebezpieczna. Problem polega na tym, że 10.0.0.11 nie szyfruje ruchu powrotnego oraz na tym, że 10.0.0.11 nie odrzuca ruchu niezaszyfrowanego lub nieuwierzytelnionego.

Dowolny użytkownik sieci może wysłać sfałszowany ruch do 10.0.0.11 a ten go przyjmie. Aby zapobiec temu problemowi, potrzebujemy SP na hoście 10.0.0.11 która wygląda tak:

```
#!/sbin/setkey -f
spdadd 10.0.0.216 10.0.0.11 any -P IN ipsec
   esp/transport//require
   ah/transport//require;
```

To polecenie powoduje, że cały ruch z 10.0.0.216 musi posiadać prawidłowe ESP i AH.

Pierwszy problem (nieszyfrowanie ruchu powrotnego) rozwiążemy uzupełniając konfigurację. Na 10.0.0.216:

```
#!/sbin/setkey -f
flush;
spdflush;

# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";
```

```
# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 10.0.0.11 10.0.0.216 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

I na 10.0.0.11:

```
#!/sbin/setkey -f
flush;
spdflush;

# AH
add 10.0.0.11 10.0.0.216 ah 15700 -A hmac-md5 "1234567890123456";
add 10.0.0.216 10.0.0.11 ah 24500 -A hmac-md5 "1234567890123456";

# ESP
add 10.0.0.11 10.0.0.216 esp 15701 -E 3des-cbc "123456789012123456789012";
add 10.0.0.216 10.0.0.11 esp 24501 -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.11 10.0.0.216 any -P out ipsec
           esp/transport//require
           ah/transport//require;

spdadd 10.0.0.216 10.0.0.11 any -P in ipsec
           esp/transport//require
           ah/transport//require;
```

Zauważ, że użyliśmy w przykładzie identycznych kluczy do obu kierunków ruchu - nie jest to oczywiście wymagane.

Aby obejrzeć konfigurację, którą przed chwilą stworzyłeś wykonaj polecenie **setkey -D**, która pokazuje SA lub polecenie **setkey -DP** pokazująca skonfigurowane reguły.

Automatyczna wymiana kluczy

W poprzedniej sekcji, szyfrowanie odbywało się za pomocą prostych kluczy. Innymi słowy, aby pozostać bezpiecznym, musiałbyś wymieniać konfigurację przez zestawiony uprzednio bezpieczny kanał. Jeśli musielibyśmy skonfigurować zdalnego hosta przez telnet, dowolny użytkownik mógłby podsłuchać klucze i konfiguracja nie byłaby bezpieczna.

Co więcej, ponieważ klucz jest współdzielony - nie jest już wcale tajny. Zdalny partner nie jest w stanie zrobić wiele znając go, ale mając wielu partnerów musimy zadbać by z każdym rozmawiać za pomocą różnych kluczy. Wymaga to dużej ilości kluczy - przy 10 partnerach musimy znać przynajmniej 50 różnych kluczy.

Poza problemami związanymi z kluczem symetrycznym, mamy jeszcze problem zmiany klucza. Jeśli osoba postronna podsłucha odpowiednio dużo ruchu, może być w stanie odzyskać klucz i odczytać nasze transmisje. Zapobiega się temu zmieniając regularnie klucz, ale należy to oczywiście zautomatyzować.

Inny problem związany z manualnym zarządzaniem kluczami jest dokładne ustalenie z partnerem parametrów połączenia - algorytmów szyfrowania i/lub uwierzytelniania. Wygodniej jest móc określić regułę w postaci ’Możemy używać 3DESa i Blowfish z kluczami o przynajmniej takiej a takiej długości’.

Aby rozwiązać te problemy, IPsec wprowadził standard wymiany kluczy w Internecie (ang. „Internet Key Exchange”, IKE). Umożliwia on automatyczną wymianę losowo generowanych kluczy, wykorzystującą technologię szyfrowania asymetrycznego zgodnie z wynegocjowanymi z partnerem parametrami.

Implementacja IPsec Linuksa 2.5 działa z demonem ’racoon’ pochodzącym z projektu KAME. Od 9 listopada wersja oparta o racoon znajduje się w dystrybucji iptools Aleksieja, aczkolwiek żeby ją skompilować będziesz musiał usunąć #include <net/route.h> w dwóch plikach. Możesz również ściągnąć wersję prekompilowaną (http://ds9a.nl/ipsec/racoon.bz2).

> **Notatka:** IKE wymaga dostępu z Internetu do portu 500/udp, upewnij się że iptables nie blokują go.

Teoria

Tak jak wyjaśniałem wcześniej, automatyczna zmiana kluczy może wykonać za nas masę pracy. Dokładniej rzecz biorąc, tworzy SA "w locie". Nie powoduje jednak pojawienia się reguł bezpieczeństwa i w zasadzie tak powinno być.

Żeby skorzystać z IKE, skonfiguruj regułę ale nie wskazuj SA. Kernel w momencie odkrycia reguły IPsec bez SA powiadomi demon IKE, który z kolei postara się odpowiednie SA wynegocjować.

Powtórzmy to jeszcze raz: SP opisuje CO chcemy uzyskać, a SA JAK chcemy to zrobić. Zastosowanie demona IKE automatycznie wymieniającego klucze umożliwia nam ograniczenie się tylko do wskazania, CO chcemy uzyskać.

Przykład

Racoon w wersji KAME dostarczany jest z całą masą opcji, z których większość ma bardzo dobre ustawienia domyślne więc nie będziemy ich modyfikować. Tak jak opisałem powyżej, musimy zdefiniować reguły bezpieczeństwa (SP) a nie SA. Pozostawimy ich negocjację demonowi IKE.

W tym przykładzie, 10.0.0.11 i 10.0.0.216 ponownie chcą zestawić kanał komunikacyjny, ale tym razem z pomocą demona racoon. Dla zachowania przejrzystości konfiguracji użyjemy współdzielonych kluczy (ang. „pre-shared keys”). Zastosowanie do uwierzytelnienia partnerów certyfikatów X.509 opisano w osobnym rozdziale, zajrzyj do sekcja *Automatyczna wymiana kluczy przy użyciu certyfikatów X.509*.

Użyjemy prawie domyślnej konfiguracji, identycznej na obu hostach:

```
path pre_shared_key "/usr/local/etc/racoon/psk.txt";
```

```
remote anonymous
{
        exchange_mode aggressive,main;
        doi ipsec_doi;
        situation identity_only;

       my_identifier address;

       lifetime time 2 min;   # sec,min,hour
       initial_contact on;
       proposal_check obey;       # obey, strict or claim

       proposal {
               encryption_algorithm 3des;
               hash_algorithm sha1;
               authentication_method pre_shared_key;
               dh_group 2 ;
       }
}

sainfo anonymous
{
        pfs_group 1;
        lifetime time 2 min;
        encryption_algorithm 3des ;
        authentication_algorithm hmac_sha1;
        compression_algorithm deflate ;
}
```

Wiele ustawień - myślę że można usunąć więcej by zbliżyć się do domyślnej konfiguracji. Parę rzeczy wartych zwrócenia uwagi - stworzyliśmy dwie sekcje anonimowe pasujące dla wszystkich partnerów zdalnych co ułatwia znakomicie konfigurację. Nie musimy definiować wszystkiego per-host, chyba że bardzo chcemy.

Co więcej, konfiguracja została wykonana w ten sposób, że identyfikowani jesteśmy na podstawie adresu IP ('my_identifier address') i deklarujemy, że jesteśmy w stanie używać algorytmów 3DES, SHA-1 oraz, że będziemy używać współdzielonego klucza zlokalizowanego w pliku `psk.txt`.

W rzeczonym pliku `psk.txt` wpisujemy po jednej linicje, która różnić się będzie na obu hostach. Na 10.0.0.11 będzie to:

```
10.0.0.216       password2
```

a na 10.0.0.216:

```
10.0.0.11       password2
```

Upewnij się że właścicielem tych plików jest root i uprawnienia do nich ustawiono na 0600 - w innym przypadku racoon nie będzie chciał ich użyć. Jeśli jeszcze tego nie zauważyłeś, zwróć uwagę że te dwa pliki są swoimi lustrzanymi odbiciami.

Teraz jesteśmy w stanie skonfigurować żądaną SP, która jest bardzo prosta. Na hoście 10.0.0.216:

```
#!/sbin/setkey -f
flush;
spdflush;

spdadd 10.0.0.216 10.0.0.11 any -P out ipsec
```

```
        esp/transport//require;

spdadd 10.0.0.11 10.0.0.216 any -P in ipsec
        esp/transport//require;
```

I na 10.0.0.11:

```
#!/sbin/setkey -f
flush;
spdflush;

spdadd 10.0.0.11 10.0.0.216 any -P out ipsec
        esp/transport//require;

spdadd 10.0.0.216 10.0.0.11 any -P in ipsec
        esp/transport//require;
```

Tu również widać, że SP są swoimi lustrzanymi odbiciami.

W tym momencie jesteśmy gotowi do uruchomienia racoon! Jeśli już działa, zainicjowanie ruchu z 10.0.0.11 do 10.0.0.216 lub odwrotnie (np. za pomocą telnetu) spowoduje rozpoczęcie przez racoon negocjacji:

```
12:18:44: INFO: isakmp.c:1689:isakmp_post_acquire(): IPsec-SA
  request for 10.0.0.11 queued due to no phase1 found.
12:18:44: INFO: isakmp.c:794:isakmp_ph1begin_i(): initiate new
  phase 1 negotiation: 10.0.0.216[500]<=>10.0.0.11[500]
12:18:44: INFO: isakmp.c:799:isakmp_ph1begin_i(): begin Aggressive mode.
12:18:44: INFO: vendorid.c:128:check_vendorid(): received Vendor ID:
  KAME/racoon
12:18:44: NOTIFY: oakley.c:2037:oakley_skeyid(): couldn't find
  the proper pskey, try to get one by the peer's address.
12:18:44: INFO: isakmp.c:2417:log_ph1established(): ISAKMP-SA
  established 10.0.0.216[500]-10.0.0.11[500] spi:044d25dede78a4d1:ff01e5b4804f0680
12:18:45: INFO: isakmp.c:938:isakmp_ph2begin_i(): initiate new phase 2
  negotiation: 10.0.0.216[0]<=>10.0.0.11[0]
12:18:45: INFO: pfkey.c:1106:pk_recvupdate(): IPsec-SA established:
  ESP/Transport 10.0.0.11->10.0.0.216 spi=44556347(0x2a7e03b)
12:18:45: INFO: pfkey.c:1318:pk_recvadd(): IPsec-SA established:
  ESP/Transport 10.0.0.216->10.0.0.11 spi=15863890(0xf21052)
```

Jeśli w tym momencie wykonamy polecenie 'setkey -D', które pokazuje nam nasze SA, powinny wyglądać mniej więcej tak:

```
10.0.0.216 10.0.0.11
        esp mode=transport spi=224162611(0x0d5c7333) reqid=0(0x00000000)
        E: 3des-cbc  5d421c1b d33b2a9f 4e9055e3 857db9fc 211d9c95 ebaead04
        A: hmac-sha1  c5537d66 f3c5d869 bd736ae2 08d22133 27f7aa99
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Nov 11 12:28:45 2002       current: Nov 11 12:29:16 2002
        diff: 31(s)       hard: 600(s)       soft: 480(s)
        last: Nov 11 12:29:12 2002       hard: 0(s)       soft: 0(s)
        current: 304(bytes)       hard: 0(bytes)       soft: 0(bytes)
        allocated: 3       hard: 0       soft: 0
        sadb_seq=1 pid=17112 refcnt=0
10.0.0.11 10.0.0.216
        esp mode=transport spi=165123736(0x09d79698) reqid=0(0x00000000)
```

```
        E: 3des-cbc  d7af8466 acd4f14c 872c5443 ec45a719 d4b3fde1 8d239d6a
        A: hmac-sha1  41ccc388 4568ac49 19e4e024 628e240c 141ffe2f
        seq=0x00000000 replay=4 flags=0x00000000 state=mature
        created: Nov 11 12:28:45 2002       current: Nov 11 12:29:16 2002
        diff: 31(s)       hard: 600(s)       soft: 480(s)
        last:                            hard: 0(s)       soft: 0(s)
        current: 231(bytes)       hard: 0(bytes)       soft: 0(bytes)
        allocated: 2       hard: 0       soft: 0
        sadb_seq=0 pid=17112 refcnt=0
```

Powinny być również widoczne reguły bezpieczeństwa, które skonfigurowaliśmy sami:

```
10.0.0.11[any] 10.0.0.216[any] tcp
        in ipsec
        esp/transport//require
        created:Nov 11 12:28:28 2002 lastused:Nov 11 12:29:12 2002
        lifetime:0(s) validtime:0(s)
        spid=3616 seq=5 pid=17134
        refcnt=3
10.0.0.216[any] 10.0.0.11[any] tcp
        out ipsec
        esp/transport//require
        created:Nov 11 12:28:28 2002 lastused:Nov 11 12:28:44 2002
        lifetime:0(s) validtime:0(s)
        spid=3609 seq=4 pid=17134
        refcnt=3
```

Problemy i znane usterki

Jeśli coś poszło nie tak, sprawdL' czy wszystkie pliki konfiguracyjne należą do roota i tylko przez niego mogą zostać przeczytane. Aby uruchomić demon racoon na pierwszym planie użyj opcji '-F'. By zmusić go do przeczytania innego pliku konfiguracyjnego niż domyślnego, użyj opcji '-f'. Aby obejrzeć dokładne informacje o wszystkim co demon robi, dodaj do pliku `racoon.conf` linijkę zawierającą 'log debug;'.

Automatyczna wymiana kluczy przy użyciu certyfikatów X.509

Wspominałem wcześniej, że zastosowanie współdzielonych kluczy nie jest najwygodniejsze, ponieważ nie są zbyt łatwe do bezpiecznego współdzielenia - a gdy już są współdzielone, przestają być sekretem. Na szczęście w tym momencie technologie wykorzystujące kryptografię asymetryczną wchodzą do gry i rozwiązują problem.

Jeśli każdy z partnerów IPsec stworzy klucz prywatny i publiczny, możemy zestawić bezpieczny kanał komunikacyjny pozwalając obu stronom opublikować swoje klucze publiczne i odpowiednio skonfigurować reguły bezpieczeństwa.

Stworzenie klucza jest relatywnie łatwe, aczkolwiek wymaga trochę pracy. Poniższy opis opiera się o zastosowanie do tego celu narzędzia `openssl`.

Tworzenie certyfikatu X.509 dla twojego hosta

OpenSSL dysponuje bogatą infrastrukturą związaną z operacjami na kluczach, podpisanych lub niepodpisanych przez autorytety/instytucje certyfikujące (ang. „certificate authorities"). Na razie zajmiemy się domyślnie generowanymi kluczami bez użycia instytucji certyfikującej.

Najpierw stworzymy ‘prośbę o certyfikat’ dla naszego hosta, nazwanego ‘laptop’:

```
$ openssl req -new -nodes -newkey rsa:1024 -sha1 -keyform PEM -keyout \
  laptop.private -outform PEM -out request.pem
```

Polecenie spowoduje konieczność odpowiedzi na parę pytań:

```
Country Name (2 letter code) [AU]:NL
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:Delft
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Linux Advanced
Routing & Traffic Control
Organizational Unit Name (eg, section) []:laptop
Common Name (eg, YOUR name) []:bert hubert
Email Address []:ahu@ds9a.nl

Please enter the following ’extra’ attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Od ciebie zależy ile z informacji podasz i w jaki sposób je podasz. Może to zależeć od twoich oczekiwań związanych z bezpieczeństwem. W powyższym przykładzie podaliśmy większość informacji.

Teraz sami ‘podpiszemy’ swoją prośbę:

```
$ openssl x509 -req -in request.pem -signkey laptop.private -out \
  laptop.public
Signature ok
subject=/C=NL/L=Delft/O=Linux Advanced Routing & Traffic \
  Control/OU=laptop/CN=bert hubert/Email=ahu@ds9a.nl
Getting Private key
```

Plik `request.pem` można teraz usunąć.

Powtórz tą operację dla wszystkich hostów które potrzebować będą klucza. Możesz oczywiście dystrybuować plik `.public`, ale plik `.private` trzymaj z daleka od wszystkich w tajemnicy.

Konfiguracja i uruchomienie

Po wygenerowaniu kluczy należy poinformować racoon by ich użył.

Wracamy w tym momencie do naszej poprzedniej konfiguracji pomiędzy dwoma hostami - 10.0.0.11 (’upstairs’) i 10.0.0.216 (’laptop’).

Do pliku `racoon.conf` na 10.0.0.11 dodaj:

```
path certificate "/usr/local/etc/racoon/certs";

remote 10.0.0.216
{
       exchange_mode aggressive,main;
       my_identifier asn1dn;
       peers_identifier asn1dn;

       certificate_type x509 "upstairs.public" "upstairs.private";

       peers_certfile "laptop.public";
```

```
        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2 ;
        }
}
```

Taka konfiguracja spowoduje, że racoon poszuka kluczy w katalogu `/usr/local/etc/racoon/certs/`. Konfiguracja zawiera również elementy specyficzne dla zdalnego hosta 10.0.0.216.

Linijka **asn1dn** informuje racoon, że identyfikator obu stron należy pobrać z kluczy publicznych. Będzie to ciąg znaków ‘subject=/C=NL/L=Delft/O=Linux Advanced Routing & Traffic Control/OU=laptop/CN=bert hubert/Email=ahu@ds9a.nl’.

Linijka **certificate_type** wskazuje lokalny klucz prywatny i publiczny. Natomiast linijka **peers_certfile** wskazuje demonowi racoon plik z kluczem publicznym zdalnego partnera - i jest to plik `laptop.public`.

Sekcja **proposal** pozostaje w niezmienionej formie w stosunku do tego co widzieliśmy wcześniej za wyjątkiem opcji **authentication_method** którą jest obecnie **rsasig** - wskazując, że używamy pary prywatnej/publicznej RSA do wykonania uwierzytelnienia.

Zmiany w konfiguracji 10.0.0.216 są analogiczne do tych, wykonanych na 10.0.0.11:

```
path certificate "/usr/local/etc/racoon/certs";

remote 10.0.0.11
{
        exchange_mode aggressive,main;
        my_identifier asn1dn;
        peers_identifier asn1dn;

        certificate_type x509 "laptop.public" "laptop.private";

        peers_certfile "upstairs.public";

        proposal {
                encryption_algorithm 3des;
                hash_algorithm sha1;
                authentication_method rsasig;
                dh_group 2 ;
        }
}
```

Teraz, gdy już dodaliśmy odpowiednie opcje na obu hostach, musimy tylko umieścić klucze we właściwych miejscach. Maszyna ‘upstairs’ powinna mieć pliki `upstairs.private`, `upstairs.public`, i `laptop.public` w katalogu `/usr/local/etc/racoon/certs`. Upewnij się że właścicielem tego katalogu jest root a sam katalog ma prawa dostępu ustawione na 0700 - w przeciwnym wypadku racoon odmówi ich odczytania.

Maszyna ‘laptop’ powinna mieć pliki `laptop.private`, `laptop.public` i `upstairs.public` w katalogu `/usr/local/etc/racoon/certs`. Innymi słowy, każdy host potrzebuje dostęp do swojego klucza prywatnego, publicznego oraz do klucza publicznego hosta z którym chce nawiązać łączność.

SprawdL’, czy reguły bezpieczeństwa zostały prawidłowo zdefiniowane (wykonaj polecenia **spdadd** z sekcja *Przykład*). Teraz uruchom racoon i wszystko powinno działać.

Jak bezpiecznie budować tunele

Aby ustanowić bezpieczną trasę komunikacji ze zdalnym partnerem, musimy wymienić klucze publiczne. O ile same klucze publiczne nie muszą być sekretem, ważne jest by zachować je w niezmienionej formie. Innymi słowy, trzeba mieć pewność że nie dojdzie do ataku typu ‘człowiek-pośrodku’ (ang. „man in the middle”).

Aby to ułatwić, w OpenSSL zaimplementowano polecenie **digest**:

```
$ openssl dgst upstairs.public
MD5(upstairs.public)= 78a3bddafb4d681c1ca8ed4d23da4ff1
```

Teraz wystarczy zweryfikować że nasz zdalny partner ma taką samą sumę kontrolną dla tego pliku. Można to zrobić po prostu spotykając się, lub przez telefon - generalnie chodzi o to, by potwierdzić to inną drogą komunikacji. Niedopuszczalne jest, by zdalna strona wysłała swoją sumę kontrolną również np. mailem!

Innym sposobem osiągnięcia tego samego, jest zaufana trzecia osoba/instytucja (ang. „Trusted Third Party”) która utrzymuje instytucję certyfikującą (ang. „Certificate Authority”, CA). CA podpisze klucz, który w naszym przypadku sami podpisaliśmy i partner będzie mógł w CA zweryfikować że otrzymany od nas klucz jest faktycznie tym, który my pierwotnie zgłosiliśmy do podpisania w CA.

Tunele IPsec

Jak dotąd zajmowaliśmy się połączeniami IPsec w tak zwanym trybie ‘transportowym’, w którym oba punkty transmisji rozumieją IPsec. Ponieważ często nie można doprowadzić do takiej sytuacji, można skonfigurować IPsec tylko na routerach i kazać im zestawić bezpieczny tunel dla całej komunikacji z podsieci "za" routerami. Jak można się łatwo domyślić, nazywane jest to właśnie ‘tunelowaniem’ lub po prostu trybem tunelu.

Konfiguracja jest prościutka. Aby tunelować cały ruch do podsieci 130.161.0.0/16 z 10.0.0.216 przez 10.0.0.11, wydajemy następujące polecenie na 10.0.0.216:

```
#!/sbin/setkey -f
flush;
spdflush;

add 10.0.0.216 10.0.0.11 esp 34501
       -m tunnel
       -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.0/24 130.161.0.0/16 any -P out ipsec
           esp/tunnel/10.0.0.216-10.0.0.11/require;
```

Zauważ że opcja **-m tunnel** jest bardzo istotna! Linijka konfiguruje SA szyfrujące za pomocą ESP pomiędzy dwoma końcami tuneli - 10.0.0.216 i 10.0.0.11.

Następnie konfigurujemy sam tunel. Polecenie instruuje kernel, by szyfrował cały ruch z 10.0.0.0/24 do 130.161.0.0/16. Zaszyfrowany ruch ma być wysyłany do routera 10.0.0.11.

Host 10.0.0.11 również wymaga analogicznej konfiguracji:

```
#!/sbin/setkey -f
flush;
spdflush;

add 10.0.0.216 10.0.0.11 esp 34501
```

```
        -m tunnel
        -E 3des-cbc "123456789012123456789012";

spdadd 10.0.0.0/24 130.161.0.0/16 any -P in ipsec
           esp/tunnel/10.0.0.216-10.0.0.11/require;
```

Zauważ że wpisy są prawie identyczne, poza zmianą **-P out** na **-P in**. Tak jak w poprzednich przykładach, skonfigurowaliśmy reguły dla ruchu przechodzącego w jedną stronę. Poprawną konfigurację szyfrowania ruchu w drugą stronę pozostawiamy jako ćwiczenie dla czytelnika.

Inną nazwą dla takiej konfiguracji jest ‘proxy ESP’, która może choć nie musi być jaśniejsza.

Notatka: Tunel IPsec wymaga włączonego przekazywania pakietów IP (ang. „IP Forwarding”) w kernelu!

Inne oprogramowanie IPsec

Tomasz Walpuski donosi, że napisał łatkę dla isakmpd w OpenBSD, pozwalającą na połączenia z implementacją IPsec w Linuksie 2.5. Co więcej, repozytorium CVS isakmpd zawiera obecnie ten kod! Trochę informacji znajdziecie na tej stronie (http://bender.thinknerd.de/~thomas/IPsec/isakmpd-linux.html).

isakmpd jest trochę inny niż racoon wspominamy powyżej, ale wiele ludzi lubi go. Można go znaleL’ć tutaj (http://www.openbsd.org/cgi-bin/cvsweb/src/sbin/isakmpd/). Więcej o CVS OpenBSD można przeczytać tutaj (http://www.openbsd.org/anoncvs.html). Tomasz udostępnił również gotowe archiwum (http://bender.thinknerd.de/~thomas/IPsec/isakmpd.tgz) dla osób niezaznajomionych z CVS lub ideą nakładania łatek.

Współpraca IPsec z innymi systemami

FIXME: Napisać to

Windows

FIXME: Napisać to

Rozdział 8. Routing multicastowy

FIXME: nie ma autora!

HOWTO poświęcone temu tematowi jest praktycznie starożytne i może być niedokładne lub mylące.

Zanim zajmiesz się routingiem multicastowym, musisz skonfigurować swój kernel. Oznacza to zdecydowanie się na konkretny typ tego routingu. Istnieją cztery główne "wspólne" rodzaje - DVMRP (multicastowa wersja unicastowego protokołu RIP), MOSPF (to samo dla OSPF), PIM-SM ("Protocol Independent Multicasting - Sparse Mode", który zakłada że użytkownicy są mocno rozproszeni) i PIM-DM (to samo, ale w trybie zagęszczonym, zakładającym że grupy użytkowników będą blisko siebie).

Przy konfiguracji jądra Linuksa zauważysz zapewne, że nie ma tych opcji. Dzieje się tak, ponieważ sam protokół obsługiwany jest przez aplikację routującą, np. Zebra, mrouted czy pimd. Z drugiej strony musisz wiedzieć, którego protokołu będziesz używać by wybrać w konfiguracji kernela odpowiednie opcje.

Dla całego routingu multicastowego, będziesz zdecydowanie musiał zaznaczyć opcję ‘multicasting’ i ‘multicast routing’. Dla DVMRP i MOSPF to wystarczy. Jeśli będziesz używał PIM, musisz również włączyć PIMv1 lub PIMv2 w zależności od tego, której wersji używa sieć do której będziesz się podłączał.

Gdy skonfigurujesz już kernel i skompilujesz go, zauważysz, że lista obsługiwanych protokołów IP wyświetlana podczas startu zawiera również IGMP. Jest to właśnie protokół do zarządzania grupami multicastowymi. W momencie pisania tego tekstu, Linuks wspiera wersję 1 i 2, mimo że wersja 3 jest udokumentowana i istnieje. Nie ma to jednak dużego znaczenia, ponieważ IGMPv3 jest nowe i nowa funkcjonalność nie jest aż tak niezbędna. Ponieważ IGMP zajmuje się grupami, tylko funkcje obsługiwane przez wszystkie implementacje w grupie będą używane. W większości przypadków będzie to funkcjonalność IGMPv2, choć można czasami spotkać gdzieniegdzie również IGMPv1.

Jak na razie szło nieL’le. Włączyliśmy multicasting. Teraz poinstruujemy kernel Linuksa by coś z tym zrobił tak, byśmy mogli zacząć routowanie. Oznacza to dodanie wirtualnej sieci multicastowej do tabeli rutownania:

```
ip route add 224.0.0.0/4 dev eth0
```

(zakładając oczywiście, że zamierzasz obsługiwać ruch multicastowy przez interfejs ‘eth0’! Zamień go w poleceniu na twoje urządzenie)

Teraz poinstruujemy Linuksa, by przekazywał pakiety...

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

W tym momencie możesz zastanawiać się czy to w ogóle do czegoś będzie służyło. By przetestować połączenie, pingnijmy domyślną grupę, 224.0.0.1, by sprawdzić czy ktoś odpowie. Wszystkie komputery w twojej sieci LAN z włączoną obsługą multicastów *powinny* odpowiedzieć - i nic innego. Zauważysz, że żadna maszyna która odpowiedziała, nie miała adresu 224.0.0.1. Co za niespodzianka! :) Jest to po prostu adres grupowy (rodzaj rozgłoszenia tylko dla prenumeratorów) i wszyscy członkowie grupy odpowiadają swoimi adresami.

```
ping -c 2 224.0.0.1
```

W tym momencie jesteś gotowy do prowadzenia routingu multicastowego. Zakładając, że masz dwie sieci między którymi możesz to robić.

(Będzie kontynuowane!)

Rozdział 9. Dyscypliny kolejkowania dla Zarządzania Pasmem

Gdy je odkryłem, *naprawdę* mnie to rozwaliło. Linuks 2.2/2.4 posiada wszystko potrzebne do zarządzania pasmem i to z funkcjonalnością porównywalną do dedykowanych systemów zarządzania pasmem z górnej półki.

Linuks idzie nawet dalej, wykraczając poza to co dają sieci Frame Relay i ATM.

By zapobiec nieporozumieniom, **tc** używa następujących reguł do specyfikacji pasma:

```
mbps = 1024 kbps = 1024 * 1024 bps => bajtów/sekundę
mbit = 1024 kbit => kilo bitów/sekundę.
mb = 1024 kb = 1024 * 1024 b => bajtów
mbit = 1024 kbit => kilo bitów.
```

Wewnętrznie, liczby przechowywane są w bps i b.

Ale gdy **tc** drukuje wartości, używa następujących wartości:

```
1Mbit = 1024 Kbit = 1024 * 1024 bps => bajtów/sekundę
```

Kolejki i Dyscypliny Kolejkowania wyjaśnione

Dzięki kolejkowaniu, określamy które dane są *WYSYŁANE*. Ważne jest, byś zrozumiał że możemy jedynie kontrolować w ten sposób dane, które *wysyłamy*.

Z uwagi na taką a nie inną budowę Internetu, nie mamy bezpośredniej kontroli nad tym, co ludzie wysyłają do nas. To trochę jak z twoją fizyczną skrzynką pocztową w domu. Nie ma sposobu by zmusić świat, aby wysyłał ci tylko określoną liczbę poczty bez skontaktowania najpierw ze wszystkimi ludL'mi.

Internet na szczęście oparty jest głównie o TCP/IP, które ma pewne cechy mogące nam pomóc. TCP/IP nie zna przepustowości sieci pomiędzy dwoma komputerami, więc zaczyna od wysyłania danych coraz szybciej („wolny start") i kiedy zaczyna gubić pakiety ponieważ nie ma już dla nich miejsca, zwalnia. Tak naprawdę jest to trochę bardziej skomplikowane, ale więcej napiszemy póL'niej.

Wracając do naszej analogii z fizyczną skrzynką pocztową - to tak jakbyś przestał czytać np. połowę poczty w nadziei, że ludzie których poczty nie czytasz, przestaną w końcu do ciebie pisać. Jedyna różnica to fakt, że działa to dla Internetu :-)

Jeśli masz router i chciałbyś zapobiec sytuacji, w której określone komputery ściągają dane za szybko, musisz wprowadzić ograniczenia na *wewnętrznym* interfejsie routera - tym, który wysyła dane do twoich komputerów.

Musisz być również pewien, że kontrolujesz połączenie w najwęższym miejscu. Jeśli masz 100Mbitową kartę sieciową a router ma łącze o przepustowości 256kbitów, musisz upewnić się że nie wysyłasz więcej danych niż router jest w stanie obsłużyć. Jeśli o to nie zadbasz, to router będzie kontrolował połączenie i ograniczał pasmo w mniej lub bardziej przydatny dla nas sposób. Musimy 'zawładnąć kolejką' mówiąc po prostu i być najwolniejszym połączeniem w łańcuchu. Jest to na szczęście bardzo łatwe.

Proste, bezklasowe Dyscypliny Kolejkowania

Tak jak to już powiedziano, dyscyplinami kolejkowania zmieniamy sposób w jaki dane są wysyłane. Bezklasowe dyscypliny kolejkowania to te, które zajmują się jedynie odbieraniem danych, przesuwaniem ich transmisji w czasie lub ewentualnie odrzucaniem.

Mogą zostać użyte do kontroli pasma dla całego interfejsu, bez żadnych dodatkowych podziałów. Bardzo ważne jest, byś zrozumiał tą część kolejkowania zanim zajmiemy się zagadnieniem zagnieżdżonych dyscyplin kolejkowania z klasami ruchu.

Najczęściej używaną dyscypliną jest ‘pfifo_fast’ - i jest domyślna. Jej popularność wyjaśnia dlaczego zaawansowane opcje są takie wydajne. Nie zawierają po prostu nic oprócz ‘kolejnej kolejki’.

Każda z tych kolejek ma swoje mocne i słabe strony. Nie wszystkie są też dokładnie przetestowane.

pfifo_fast

Kolejka ta, dokładnie tak jak wskazuje na to jej nazwa, to tradycyjne ‘Pierwszy Wszedł, Pierwszy Wyjdzie’ (ang. „First In First Out”, FIFO). Oznacza to, że żaden pakiet nie będzie specjalnie traktowany (przynajmniej nie wprost). Kolejka ta ma 3 ‘pasma’ (ang. „band”). W każdym z pasm reguły FIFO działają niezależnie. Jednak jeśli w paśmie 0 są jeszcze pakiety, pasmo 1 nie zostanie obsłużone. Tak samo dzieje się w przypadku pasm 1 i 2.

Kernel honoruje tak zwaną flagę ‘Typu Usługi’ (ang. „Type of Service”) i zajmuje się ustawianiem opcji ‘minimalna zwłoka’ w pakietach z pasma 0.

Nie pomylcie tej bezklasowej kolejki z klasową kolejką PRIO! Mimo, że zachowują się podobnie, pfifo_fast jest bezklasowa i nie można dodawać innych kolejek do niej przy użyciu polecenia **tc**.

Parametry i ich użycie

Nie można konfigurować kolejki pfifo_fast, ponieważ jest ustawiona na sztywno w domyślnej konfiguracji. Poniżej jak to jest zrobione:

priomap

Określa w jaki sposób priorytety dla pakietów, przydzielane przez kernel, odwzorowywane są na pasma. Odwzorowywanie zachodzi na podstawie oktetu ToS pakietu, który wygląda tak:

```
   0     1     2     3     4     5     6     7
+-----+-----+-----+-----+-----+-----+-----+-----+
|                 |                       |     |
|   KOLEJNOŚĆ     |          TOS          | MBZ |
|                 |                       |     |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

Czterobitowe pole ToS definiowane jest w następujący sposób:

```
Binarnie Decymalnie  Znaczenie
-----------------------------------------
1000     8           Zminimalizuj zwłokę (md)
0100     4           Zmaksymalizuj przepustowość (mt)
0010     2           Zmaksymalizuj niezawodność (mr)
0001     1           Zminimalizuj koszt (mmc)
0000     0           Normalna usługa
```

Ponieważ na prawo od pola ToS znajduje się jeszcze jeden bit, pole ToS jest równe podwojonej wartości bitów ToS. **tcpdump -v -v** pokazuje wartość całego pola ToS, nie tylko tych 4 bitów. Jest to wartość, którą możesz znaleL’ć w pierwszej kolumnie tej tabeli:

```
TOS     Bity  Znaczenie                     Priorytet Linuksa    Pasmo
------------------------------------------------------------------------
```

```
0x0     0     Normalna Usługa              0 Najlepszy efekt    1
0x2     1     Zminimalizuj koszt           1 Wypełniacz         2
0x4     2     Zmaksymalizuj niezawodność   0 Najlepszy efekt    1
0x6     3     mmc+mr                       0 Najlepszy efekt    1
0x8     4     Zmaksymalizuj przepustowość  2 Większość          2
0xa     5     mmc+mt                       2 Większość          2
0xc     6     mr+mt                        2 Większość          2
0xe     7     mmc+mr+mt                    2 Większość          2
0x10    8     Zminimalizuj zwłokę          6 Interaktywnie      0
0x12    9     mmc+md                       6 Interaktywni       0
0x14    10    mr+md                        6 Interaktywni       0
0x16    11    mmc+mr+md                    6 Interaktywni       0
0x18    12    mt+md                        4 Większość interakt.1
0x1a    13    mmc+mt+md                    4 Większość interakt.1
0x1c    14    mr+mt+md                     4 Większość interakt.1
0x1e    15    mmc+mr+mt+md                 4 Większość interakt.1
```

Dużo numerków. Druga kolumna zawiera wartość czterech bitów pola ToS, a następnie ich znaczenie. Na przykład wartość 15 oznacza pakiet wymagający Minimalnego Kosztu, Maksymalnej Niezawodności, Maksymalnej Przepustowości i Minimalnej Zwłoki. Nazwałbym go ‘Pakietem Holenderskim’.

Czwarta kolumna to sposób w jaki kernel interpretuje bity ToS, określony priorytetem, który jest im przyznawany.

Ostatnia kolumna to wynik domyślnej mapy priorytetów (priomap). W linii poleceń, wygląda ona tak:

```
1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1
```

Oznacza to na przykład dla priorytetu 4, że mapowany jest na pasmo 1. Mapa priorytetów pozwala również na wylistowanie wyższych priorytetów (> 7), które nie odpowiadają mapowaniom ToS, ale są rozsyłane dla innych zastosowań.

Poniższy wydruk pochodzi z RFC 1349 (przeczytaj ten dokument jeśli chcesz zapoznać się ze szczegółami) i omawia jak aplikacje mogłyby ustawiać pole TOS swoich pakietów:

```
TELNET                  1000           (zminimalizuj zwłokę)
FTP
    kanał kontroli      1000           (zminimalizuj zwłokę)
    Dane                0100           (zmaksymalizuj przepustowość)

TFTP                    1000           (zminimalizuj zwłokę)

SMTP
    Faza komend         1000           (zminimalizuj zwłokę)
    Faza danych         0100           (zmaksymalizuj przepustowość)

Domain Name Service
    Zapytanie UDP       1000           (zminimalizuj zwłokę)
    Zapytanie TCP       0000
    Transfer Strefy     0100           (zmaksymalizuj przepustowość)

NNTP                    0001           (zminimalizuj koszt)

ICMP
    Błędy               0000
    Zapytania           0000 (zwykle)
    Odpowiedzi          <takie jak zapytania> (zwykle)
```

txqueuelen

Długość tej kolejki odpowiada konfiguracji interfejsu. Można ją sprawdzić i ustawić posługując się poleceniami **ifconfig** i **ip**. By ustawić długość kolejki na 10, wykonaj polecenie **ifconfig eth0 txqueuelen 10**.

Nie można ustawić wartości tego parametru za pomocą **tc**!

Token Bucket Filter

Token Bucket Filter (TBF, w wolnym tłumaczeniu „Filtr Wiadra Żetonów") to prosta kolejka z dyscypliną, która przepuszcza tylko dane przychodzące z pewną częstotliwością nie przekraczającą nałożonych ograniczeń, ale z możliwością przyjęcia krótkich serii danych które przekraczają tych ograniczeń.

TBF jest bardzo precyzyjna, przyjazna zarówno dla sieci jak i procesora. Powinna być twoim pierwszym wyborem jeśli chcesz po prostu zwolnić interfejs!

Implementacja TBF składa się z bufora ('wiadra', ang. „bucket"), wypełnianego pewnymi wirtualnymi porcjami danych ('żetonami', ang. „token") z określoną częstotliwością. Najważniejszym parametrem wiadra jest jego rozmiar, określający ilość żetonów które może ono przechować.

Każdy nadchodzący żeton wyjmuje jeden przychodzący pakiet z kolejki danych i jest kasowany z wiadra. Skojarzenie tego algorytmu z dwoma przepływami - żetonów i danych, daje trzy możliwe scenariusze:

- Dane docierają do TBF z częstotliwością *równą* częstotliwości napływania żetonów. W tym przypadku każdy przychodzący pakiet ma swój żeton i przechodzi przez kolejkę bez zwłoki.
- Dane docierają do TBF z częstotliwością *mniejszą* niż częstotliwość napływania żetonów. Tylko ich część jest używana, więc te puste zbierane są do momentu osiągnięcia rozmiaru 'wiadra'. Mogą być potem użyte do krótkiej serii pakietów przekraczających normalny limit przepustowości.
- Dane docierają do TBF z częstotliwością *większą* niż częstotliwość napływania żetonów. Oznacza to, że 'wiadro' zostanie w końcu opróżnione z pustych żetonów, powodując przez moment zwiększenie przepustowości. Nazywa się to 'przekroczeniem limitu' i jeśli pakiety nadal będą napływały z niezmienną częstotliwością, niektóre będą odrzucane.

Ostatni scenariusz jest bardzo ważny, ponieważ umożliwia kształtować przepustowość dostępną dla danych które przechodzą przez filtr.

Akumulacja żetonów zapewnia pewnego rodzaju zapas bezpieczeństwa, który wykorzystany zostanie gdy pakiety zaczną nagle docierać z większą niż dotychczas częstotliwością. W końcu jednak, ciągłe obciążenie większe od maksymalnego zdefiniowanego spowoduje przeładowanie i pakiety zaczną być najpierw opóźniane, a w końcu odrzucane.

Proszę zauważyć, że w implementacji programowej, żetony odpowiadają bajtom a nie pakietom.

Parametry i ich użycie

Mimo, że zapewne nie będziesz potrzebował ich zmieniać, tbf udostępnia parę zmiennych kontrolnych. Po pierwsze, parametry dostępne zawsze:

limit lub latency (limit lub opóźnienie)

> Limit to numer bajtów, które mogą zostać skolejkowane w oczekiwaniu na wolne żetony. Możesz również określić parametr wskazując opóźnienie, określające maksymalny czas jaki pakiet może spędzić w TBF. To drugie bierze pod uwagę rozmiar ‘wiadra’, częstotliwość i ewentualnie (jeśli zostanie ustawiona) częstotliwość szczytową.

burst/buffer/maxburst

> Rozmar ‘wiadra’, w bajtach. Jest to maksymalna ilość bajtów dla których jednocześnie mogą być dostępne żetony. Generalnie, kształtowanie ruchu większych przepustowości wymaga większego bufora. Dla ruchu 10mbit/s na karcie Intel, potrzebujesz przynajmniej 10 kilobajtowego bufora, jeśli chcesz osiągnąć żądaną przepustowość!
>
> Jeśli bufor będzie za mały, pakiety mogą być odrzucane dlatego, że przepływa więcej żetonów na jednostkę zegara, niż mieści się w ‘wiadrze’.

mpu

> Pakiet długości zero bajtów nie zużywa zero przepustowości. Dla ethernetu, pusty pakiet zużywa mniej niż 64 bajty. ‘Minimalna Jednostka Pakietu’ (ang. „Minimum Packet Unit”) określa minimalną zajętość żetona przez pakiet.

rate

> Ustawienie prędkości. Zajrzyj do notatek powyżej o limitach!

Jeśli ‘wiadro’ zawiera żetony i nie może być puste, domyślnie pracuje z nieograniczoną prędkością. Jeśli jest to nieakceptowalne, użyj następujących parametrów:

peakrate

> Domyślnie, jeśli dostępne są żetony a przybywają pakiety, są one natychmiast wysyłane. Może to nie być konkretnie to, czego chcesz, szczególnie jeśli masz duże ‘wiadro’.
>
> Parametr ‘peakrate’ może zostać użyty do określenia jak szybko ‘wiadro’ może być opróżniane. Jeśli go określisz, po wysłaniu pakietu odczekamy chwilę i dopiero po jej upływie wyślemy następny - dzięki przeliczeniu przestrzeni czasowej między pakietami tak, aby wysyłać dokładnie tyle ile wynosi wartość tego parametru.
>
> Jednak ponieważ zegar Unixowy ma rozdzielczość tylko 10ms, otrzymując 10.000 bitów średniego ruchu ograniczeni jesteśmy do ‘peakrate’ równej około 1mbit/s!

mtu/minburst

> Oczywiście ‘peakrate’ równe 1mbit/s nie jest zbyt użyteczne, jeśli twój normalny ruch wynosi więcej. Większa wartość szczytowa możliwa jest do osiągnięcia przez wysyłanie większej ilości pakietów na jedną jednostkę zegara, co oznacza, że tworzymy drugie wiadro!
>
> To drugie wiadro domyślnie ma wielkość jednego pakietu, więc tak naprawdę nie jest wcale wiadrem.

By wyliczyć maksymalną wartość szczytową, pomnóż skonfigurowane MTU przez 100 (lub, bardziej poprawnie, HZ, co oznacza 100 na platformie intelowskiej i 1024 na Alfie).

Przykładowa konfiguracja

Prosta, ale *bardzo* przydatna konfiguracja to:

```
# tc qdisc add dev ppp0 root tbf rate 220kbit latency 50ms burst 1540
```

No dobra, dlaczego jest taka przydatna? Jeśli masz urządzenie sieciowe z całkiem dużą kolejką, tak jak na przykład modem kablowy lub DSL i łączysz je do szybkiego urządzenia takiego jak na przykład interfejs ethernetowy, zauważysz od razu, że wrzucanie czegoś od razu niszczy interaktywność.

Dzieje się tak dlatego, ponieważ wysyłanie pakietów zapełni kolejkę w modemie, która jest prawdopodobnie *bardzo duża* - taka konfiguracja pomaga uzyskać dużą przepustowość. Ale to nie jest to co chcesz osiągnąć, chcesz mieć pełną interaktywność a nie zapchać modem jednym strumieniem i nie móc zrobić już nic innego.

Linia powyżej spowalnia wysyłanie do częstotliwości, która nie prowadzi zapełniania kolejki w modemie - kolejka będzie już na Linuksie, gdzie mamy swobodę konfiguracji jej do określonego rozmiaru.

Zmień wartość 220kbit na swoją *faktyczną* prędkość wysyłania danych minus parę procent. Jeśli masz naprawdę bardzo szybki modem, możesz troszkę zwiększyć wartość ‘burst’.

Sprawiedliwe Kolejkowanie Stochastyczne (ang. „Stochastic Fairness Queueing”)

Stochastic Fairness Queueing (SFQ) to prosta implementacja rodziny algorytmów ze sprawiedliwym podziałem pasma. Jest mniej dokładna niż inne, ale wymaga również mniejszej ilości wyliczeń.

Kluczowym pojęciem w SFQ jest konwersacja (lub potok), która odpowiada zwykle sesji TCP lub strumieniowi UDP. Ruch dzielony jest w dużą ilość kolejek FIFO, po jednej na każdą konwersację. Następnie każda z kolejek opróżniana jest na zasadzie ‘round-robin’ tak aby każda z sesji mogła wysłać swoje dane.

Zapewnia to sprawiedliwą pracę i uniemożliwia jednej konwersacji zajęcie całego pasma. SFQ nazywana jest ‘stochastyczną’, ponieważ tak naprawdę nie przydziela kolejki dla każdej sesji, a używa procedury dzielącej ruch na ograniczoną liczbę kolejek przy pomocy algorytmu ‘mieszającego’ (ang. „hashing”).

Ponieważ używana jest wartość mieszająca (wspominany hash), może dojść do sytuacji w której więcej niż jedna sesja może skończyć w tym samym wiadrze. Oznaczałoby to w praktyce zmniejszenie szansy wysłania pakietu o połowę i podzielenie tym samym na pół dostępnej efektywnej prędkości. By temu zapobiec, SFQ zmienia często algorytm mieszający i nawet jeśli dojdzie do kolizji sesji będzie to działo się tylko przez parę sekund.

Warto zauważyć, że SFQ jest użyteczne w przypadku gdy twój interfejs wychodzący jest naprawdę pełen! Jeśli nie będzie, nie stworzona zostanie kolejka i tak naprawdę nie będzie miało to żadnego efektu. Później omówimy jak połączyć SFQ z innymi dyscyplinami kolejkowania i uzyskać to co najlepsze z obu sytuacji.

W szczególności, zastosowanie SFQ wobec interfejsu ethernetowego podłączonego do modemu kablowego lub rutera DSL jest bezcelowe, bez dalszego nałożenia ograniczeń!

Parametry i użycie

SFQ jest praktycznie samokonfigurowalna:

perturb

> Wartość określajaca co ile sekund zmieniany będzie algorytm mieszający. Jeśli nie zostanie podana, wyliczona wartość mieszająca nie będzie rekonfigurowana. Generalnie pomijanie tego parametru nie jest zalecane - wartość 10 sekund wydaje się dobrym wyborem.

quantum

> Ilość bajtów, którą strumień może zdjąć z kolejki zanim szansę wysłania otrzyma kolejna kolejka. Domyślnie ustawione jest na równowartość maksymalnej jednostki transmisji 1 pakietu (MTU). Nie ustawiaj tego parametru poniżej MTU!

Przykładowa konfiguracja

Jeśli masz urządzenie połączone łączem o identycznej przepustowości jaka jest ogólnie dostępna, tak jak na przykład w sytuacji połączenia modemowego, to polecenie pomoże ci uwydatnić sprawiedliwy podział ruchu:

```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc -s -d qdisc ls
qdisc sfq 800c: dev ppp0 quantum 1514b limit 128p flows 128/1024 perturb 10sec
 Sent 4812 bytes 62 pkts (dropped 0, overlimits 0)
```

Numer ‘800c:’ to automatycznie nadawana etykietą, limit oznacza, że w kolejce oczekiwać może 128 pakietów. Dostępne są 1024 wiadra kontrolowane algorytmem mieszającym, z których 128 może być jednocześnie aktywnych (więcej pakietów nie wejdzie do kolejki!). Co 10 sekund, wartości mieszające są rekonfigurowane.

Kiedy używać której kolejki

Podsumowując, powyższe algorytmy tworzą proste kolejki zarządzające ruchem przez rekolejkowanie, zwalnianie lub odrzucanie pakietów.

Poniższe podpowiedzi mogą pomóc w wyborze odpowiedniej kolejki. Wspominają one niektóre dyscypliny kolejkowania omówione w rozdziale *Rozdział 14*.

- Jeśli chcesz zwolnić ruch wychodzący, użyj TBF. Działa nawet dla dużych przepustowości, jeśli odpowiednio wyskalujesz wiadro.
- Jeśli masz naprawdę zapchane łącze i chcesz mieć pewność, że żadna sesja nie zdominuje całego pasma wychodzącego, użyj SFQ.
- Jeśli masz naprawdę dużą sieć szkieletową i wiesz co robisz, rozważ algorytm RED (opisany w rozdziale zaawansowanym).
- By kształtować (ang. „shape”) ruch przychodzący, którego nie przekazujesz gdzie indziej, użyj ‘Polityki dla ruchu przychodzącego’ (ang. „Ingress Policer”). Nawiasem mówiąc, kontrolowanie ruchu przychodzącego to ‘określanie polityki’ (ang. „policing”) a nie ‘kształtowanie’.

- Jeśli go jednak przekazujesz, używaj TBF na interfejsie wysyłającym te dane, chyba, że może on wychodzić przez wiele interfejsów. W tym wypadku jedyną wspólną cechą będzie interfejs którym ruch dotarł i wtedy należy użyć polityki dla ruchu przychodzącego.
- Jeśli nie chcesz kształtować ruchu, a jedynie sprawdzić czy twój interfejs jest tak obciążony, że musi używać swojej kolejki, użyj kolejkowania ‘pfifo’ (nie ‘pfifo_fast’). Brakuje jej ograniczeń, ale zlicza parametry pracy.
- Na koniec - możesz użyć „kształtowania osobistego”. Nie zawsze będziesz w stanie użyć dostępnej technologii do osiągnięcia dokładnie założonego celu, a użytkownicy zwykle odczuwają nakładane ograniczenia jako złośliwość. Przyjazne słowo może pomoć w osiągnięciu odpowiednich parametrów pracy w sieci!

Terminologia

By prawidłowo zrozumieć bardziej zaawansowane konfiguracje, niezbędne jest prawidłowe wytłumaczenie pewnych koncepcji. Z uwagi na poziom skomplikowania i świeżość tematu, wiele ludzi używa różnych słów na oznaczanie tych samych rzeczy.

Poniższa lista bazuje na zawartości dokumentu `draft-ietf-diffserv-model-06.txt` zatytułowanego *An Informal Management Model for Diffserv Routers*. Można go znaleźć pod adresem
http://www.ietf.org/internet-drafts/draft-ietf-diffserv-model-06.txt
(http://www.ietf.org/internet-drafts/draft-ietf-diffserv-model-06.txt).

Zapoznaj się z dokumentem by poznać dokładne definicje terminów, które będziemy używali.

Dyscyplina kolejkowania (ang. „Queueing Discipline”, qdisc)

: Algorytm zarządzający kolejką urządzenia, dla ruchu ‘przychodzącego’ (ang. „ingrees”) lub ‘wychodzącego’ (ang. „egrees”).

root qdisc

: Główna qdisc to dyscyplina dołączona bezpośrednio do urządzenia.

Bezklasowa dyscyplina kolejkowania (ang. „Classless qdisc”)

: Qdisc w której nie ma konfigurowalnych wewnętrznych podziałów.

Dyscyplina kolejkowania z klasami (ang. „Classful qdisc”)

: Dyscyplina kolejkowania może zawierać wiele klas. Każda z nich może zawierać następne dyscypliny kolejkowania i tak dalej i tak dalej - ale nie musi. Zgodnie z definicją, ‘pfifo_fast’ *jest* dyscypliną kolejkowania z klasami - zawiera trzy pasma będące klasami. Ponieważ jednak żadne jej parametry nie mogą być modyfikowane narzędziem **tc**, z punktu widzenia użytkownika jest bezklasowa.

Klasy (ang. „Classes”)

: Dyscyplina kolejkowania z klasami, tak jak wskazuje jej nazwa, zawiera klasy. Mogą one zawierać kolejne dyscypliny i tak dalej i tak dalej. Innymi słowy, możesz skonfigurować qdisc która będzie rodzicem dla innej qdisc. Liść to qdisc, która nie posiada dzieci. Klasa ma przydzieloną jedną dyscyplinę kolejkowania i jest ona odpowiedzialna za wysyłanie danych z tej klasy. Gdy tworzysz klasę, przydzielana jest do niej domyślnie dyscyplina FIFO. Gdy dodasz dziecko do dyscypliny, FIFO jest usuwana. Dla klasy będącej liściem, algorytm FIFO można zastąpić bardziej odpowiednią dyscypliną kolejkowania. Możesz nawet ją zastąpić dyscypliną kolejkowania z klasami, co pozwoli ci na dalszą rozbudowę klas.

Klasyfikator (ang. „Classifier”)

Każda dyscyplina kolejkowania z klasami musi wiedzieć, do której klasy ma wysłać pakiet. Wykonuje to właśnie na podstawie klasyfikatora.

Filtr (ang. „Filter”)

Klasyfikacja może zostać wykonana na podstawie filtrów. Filtr zawiera pewną liczbę warunków, które jeśli pasują, sprawiają że filtr również pasuje.

Planowanie (ang. „Scheduling”)

Dyscyplina kolejkowania z klasami, z pomocą klasyfikatora może zdecydować, że niektóre pakiety powinny zostać wysłane wcześniej niż inne. Proces ten nazywamy planowaniem i jest wykonywany na przykład przez qdisc ‘pfifo_fast’, o której wspomniano wcześniej. Planowanie nazywa się również ‘porządkowaniem’, ale jest to raczej mylące.

Kształtowanie (ang. „Shaping”)

Proces odwlekania momentu wysłania pakietu tak, by pasował do zdefiniowanych charakterystyk ruchowych i jego maksymalnych wartości. Kształtowanie ruchu wykonywane jest przy opuszczaniu przez pakiet systemu. Czasami kształtowanie oznacza również odrzucanie pakietów zamiast opóźniania ich wysłania.

Narzucanie polityki (ang. „Policing”)

Proces odwrotny do kształtowania, wykonywany na ruchu przychodzącym. Może on tylko odrzucać pakiety a nie odwlekać ich obsługę - nie istnieje ‘kolejka dla ruchu przychodzącego’.

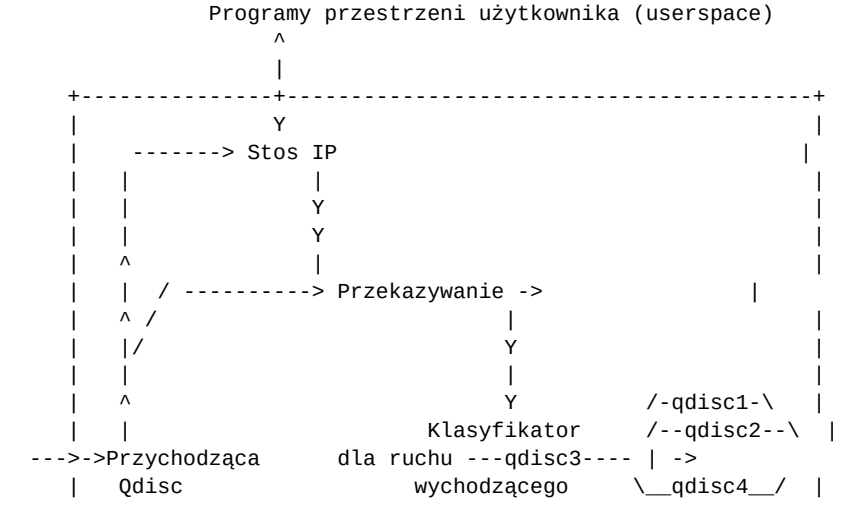
Kolejka bezstratna (ang. „Work-Conserving”)

Ta qdisc zawsze dostarcza pakiet, jeśli tylko jest dostępny. Innymi słowy, nigdy nie odwleka wysłania pakietu, jeśli urządzenie sieciowe jest gotowe do przyjęcia go (w przypadku ruchu wychodzącego).

Kolejka stratna (ang. „non-Work-Conserving”)

Niektóre kolejki, takie jak na przykład TBF, mogą być zmuszone przytrzymać pakiet przez pewien czas, by dopasować się do ustalonego limitu na przepustowość. Oznacza to, że czasami nie wyślą pakietu, mimo że są już do tego gotowe.

Dobrze, przebrnęliśmy przez terminologię, zobaczmy teraz gdzie te wszystkie rzeczy się znajdują.



```
|                                          \-qdiscN_/   |
|                                                       |
+-------------------------------------------------------+
```

Dziękuje Jamal Hadi Salim za ten diagram ASCII.

Duży kwadrat reprezentuje kernel. Lewa strzałka reprezentuje ruch docierający do twojego komputera z sieci. Zostaje on następnie wrzucony do klasyfikatora ruchu przychodzącego, który może zastować filtry i być może odrzucić ten ruch. Nazywamy to ‘narzucaniem polityki’.

Dzieje się to we wczesnym stadium, zanim pakiet obejrzał sobie większość kernela. Jest to więc dobre miejsce by odrzucać ruch bez zbędnego angażowania cykli CPU.

Jeśli pakiet zostanie przepuszczony, może być skierowany do aplikacji działającej lokalnie - trafia wtedy na stos IP by zostać przetworzony a potem do tej konkretnej aplikacji. Pakiet może jednak być przekazywany dalej, trafia wtedy do klasyfikatora dla ruchu wychodzącego. Programy przestrzeni użytkownika wysyłające pakiety również dostarczają danych dla klasyfikatora ruchu wychodzącego.

Ruch wychodzący z takiego czy innego źródła trafia następnie do przetworzenia w określonej liczbie skonfigurowanych qdisc. W domyślnym, nieskonfigurowanym stanie, istnieje jedynie jedna kolejka dla ruchu wychodzącego - jest to ‘pfifo_fast’, która zawsze odbiera pakiet. Nazywamy to ‘kolejkowaniem’.

Pakiet czeka zatem w qdisc na decyzję kernela powodującą wysłanie go przez któryś interfejs sieciowy. Nazywamy to z kolei ‘opróżnianiem kolejki’.

Rysunek ten ma zastosowanie również dla sytuacji z jednym urządzeniem sieciowym - strzałki skierowane do i od kernela nie powinny być brane zbyt dosłownie. W każdym urządzeniu sieciowym obsługiwane jest zarówno odebranie pakietu jak i jego wysłanie.

Dyscypliny kolejkowania z klasami

qdisc z klasami przydatne są jeśli masz ruch różnego rodzaju, który powinien być w różny sposób traktowany. Jedną z qdisc z klasami jest ‘CBQ’ - ‘Kolejkowanie oparte o klasy’ (ang. „Class Based Queueing”) i jest tak popularna, że ludzie identyfikują całe zagadnienie kolejkowania jako właśnie CBQ, ale nie jest to do końca poprawne.

CBQ jest jedynie najstarszą zabawką - a jednocześnie najbardziej skomplikowaną. Nie zawsze może robić to co chciałeś. Może to być szok szczególnie dla ludzi podatnych na ‘efekt sendmaila’ - który uczy nas, że skomplikowane technologie dostarczane bez dokumentacji muszą być najlepszymi dostępnymi.

Więcej o CBQ i jego alternatywach już niedługo.

Przepływ w qdisc z klasami

Ruch docierający do qdisc z klasami, musi zostać rozesłany do klas podrzędnych, czyli ‘sklasyfikowany’. By określić co zrobić z konkretnym pakietem używa się filtrów. Ważne jest, by zauważyć że to filtry wywoływane są w qdisc a nie odwrotnie!

Filtry dołączone do określonych qdisc odpowiadają decyzją i kolejka używa jej do skolejkowania pakietu w jedną z klas. Każda podklasa może sprawdzić swoje filtry by ewentualnie dopasować się do jeszcze innych założeń czy instrukcji. Jeśli tego nie robi, klasa kolejkuje pakiet do qdisc, ją zawierającą.

Poza zawieraniem w sobie innych kolejek, większość qdisc z klasami wykonuje również kształtowanie. Dzięki temu można zarówno planować pakiety (np. przy użyciu SFQ) i ograniczać częstotliwość z jaką będą wysyłane. Potrzebujesz takiej możliwości gdy masz szybki interfejs (na przykład, ethernet) i wolne urządzenie sieciowe (np. modem kablowy).

Gdybyśmy uruchomili tylko SFQ nic się nie stanie, ponieważ pakiety docierają i opuszczają twój router bez żadnej zwłoki: interfejs wyjściowy jest dużo szybszy niż faktyczna przepustowość łącza. Nie ma więc kolejki, którą możnaby kontrolować.

Rodzina kolejek z klasami: korzenie, uchwyty, rodzeństwo i rodzice

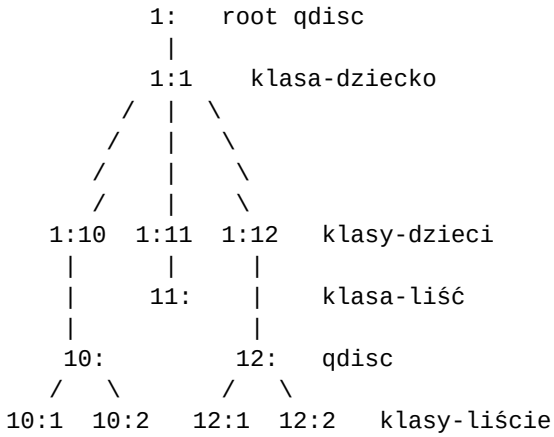
Każdy interfejs ma jedną wychodzącą kolejkę-korzeń, domyślnie jest to wspomniana wcześniej bezklasowa ‘pfifo_fast’. Każdej z qdisc można przydzielić uchwyt, który będzie potem używany przez polecenia konfigurujące by odwołać się do niej. Poza kolejkami dla ruchu wychodzącego, interfejs może mieć również przypisaną politykę dla ruchu przychodzącego.

Uchwyt kolejki składa się z dwóch części: starszego i młodszego numeru. Charakterystyczną nazwą dla kolejki-korzenia jest ‘1:’, które równe jest ‘1:0’. Młodszy numer kolejki z dyscypliną zawsze równy jest 0.

Klasy muszą mieć ten sam starszy numer, określający ich rodzica. Musi być on unikalny w danej konfiguracji qdisc dla ruchu wychodzącego lub przychodzącego. Młodszy numer musi być unikalny w obrębie qdisc i jej dzieci.

Jak filtry używane są do klasyfikowania ruchu

Reasumując, typowa hierarchia może wyglądać tak:



Ale niech to drzewko cię nie zmyli! Nie powinieneś wyobrażać sobie kernela w wierzchołku tego drzewa z siecią poniżej, to po prostu nie tak. Pakiety są kolejkowane i usuwane z kolejki w korzeniu, który jest jedyną kolejką z którą rozmawia kernel.

Pakiet może być sklasyfikowany w następującym łańcuchu:

1: -> 1:1 -> 1:12 -> 12: -> 12:2

Pakiet znajduje się w kolejkce należącej do qdisc dołączonej do klasy 12:2. W tym przykładzie, filtr został dołączony do każdego ‘węzła’ w drzewie, gdzie decyduje do której gałęzi pakiet zostanie skierowany dalej. Może to mieć sens. Jednak możliwe jest również:

1: -> 12:2

W tym przypadku, filtr dołączony do korzenia zdecydował, by wysłać pakiet od razu do 12:2.

Jak pakiety są zdejmowane z kolejki przez sprzęt

Kiedy kernel zdecyduje, że musi zdjąć pakiety z kolejki i wysłać je interfejsem, kolejka-korzeń 1: odbiera żądanie rozkolejkowania, które przesyłane jest do ‘1:1’, następnie do ‘10:’, ‘11:’ i ‘12:’ - z których każde odpytuje swoje rodzeństwo i próbuje wykonać na nich funkcję dequeue(). W tym przypadku, kernel musi przejść całe drzewo, ponieważ tylko 12:2 zawiera pakiet.

Krótko mówiąc, zagnieżdżone klasy rozmawiają TYLKO ze swoimi rodzicami, nigdy z interfejsami. Tylko kolejka-korzeń jest rozkolejkowywana przez kernel!

Wynikiem tego jest fakt, że klasy nigdy nie są rozkolejkowywane szybciej niż na to pozwalają ich rodzice. I to jest dokładnie to czego chcemy - możemy dzięki temu mieć kolejkowanie SFQ w klasie przychodzącej, która nie wykonuje kształtowania a jedynie planowanie; możemy również kształtować ruch w kolejkach wychodzących, które wykonują kształtowanie.

qdisc PRIO

Kolejka PRIO nie zajmuje się tak naprawdę kształtowaniem ruchu, dzieli jedynie ruch na podstawie tego, jak skonfigurowałeś filtry. Możesz traktować ją jak ‘pfifo_fast’ na sterydach - zamiast pasma jest osobna klasa zamiast prostej kolejki FIFO.

Kiedy pakiet zostaje skolejkowany w qdisc PRIO, na podstawie komend filtrujących które podałeś wcześniej, wybierana jest klasa. Domyślnie, stworzone są trzy i zawierają czyste kolejki FIFO bez żadnej struktury wewnętrznej, ale możesz zastąpić je dowolnymi dostępnymi qdisc.

Zawsze gdy pakiet musi zostać wyjęty z kolejki, sprawdza się klasę :1. Wyższe klasy są sprawdzane tylko wtedy, gdy poprzednie nie zwróciły pakietu.

Kolejka PRIO jest bardzo użyteczna, gdy chcesz prioretyzować określone rodzaje ruchu nie tylko na podstawie flag ToS, ale całej potęgi, którą zapewniają filtry **tc**. Może również zawierać inne kolejki do predefiniowanych 3, podczas gdy ‘pfifo_fast’ jest ograniczona do prostych kolejek FIFO.

Ponieważ kolejka ta nie zajmuje się kształtowaniem ruchu, obowiązuje to samo ostrzeżenie co do SFQ: używaj jej tylko jeśli fizyczne łącze jest naprawdę zapchane, lub wsadź ją do qdisc z klasami, która nie zajmuje się kształtowaniem ruchu. To ostatnie oznacza praktycznie wszystkie modemy kablowe i urządzenia DSL.

Formalnie rzecz biorąc, kolejka PRIO jest planującą kolejką bezstratną.

Parametry PRIO i ich użycie

Następujące parametry rozpoznawane są przez **tc**:

bands - pasma

> Ilość pasm do utworzenia. Każde pasmo to tak naprawdę klasa. Jeśli zmienisz ten numer, musisz zmienić również:

priomap - mapa priorytetów

> Jeśli nie dostarczysz filtrów **tc** do klasyfikowania ruchu, kolejka PRIO będzie sprawdzać priorytety TC_PRIO by stwierdzić, jak kolejkować ruch.
>
> Działa to tak jak kolejka ‘pfifo_fast’ wspomniana wcześniej.

Pasma są klasami i domyślnie nazywane są od ‘numer_starszy:1’ do ‘numer_starszy:3’, więc jeśli twoja kolejka PRIO nazywa się 12:, użyj **tc** by przefiltrować ruch na 12:1 dla podniesienia jej priorytetu.

Powtarzam, że pasmo 0 trafia na młodszy numer 1! A pasmo 1 na młodszy numer 2 i tak dalej!

Przykładowa konfiguracja

Stworzymy takie drzewko:

```
           1:   root qdisc
          / | \
         /  |  \
        /   |   \
      1:1  1:2  1:3    klasy
       |    |    |
      10:  20:  30:    qdiscs    qdiscs
      sfq  tbf  sfq
pasmo  0    1    2
```

‘Ciężki’ ruch pójdzie do 30:, ruch interaktywny do 20: lub 10:.

Polecenia do wpisania:

```
# tc qdisc add dev eth0 root handle 1: prio
## To *od razu* tworzy klasy 1:1, 1:2, 1:3

# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

Teraz sprawdźmy co stworzyliśmy:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 132 bytes 2 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 174 bytes 3 pkts (dropped 0, overlimits 0)
```

Jak widzisz, pasmo 0 dostało już trochę ruchu i jeden pakiet został wysłany w czasie wykonywania tego polecenia!

Użyjemy teraz trochę transferu danych narzędziem, które poprawnie ustawia flagi ToS i spojrzymy jeszcze raz:

```
# scp tc ahu@10.0.0.11:./
ahu@10.0.0.11’s password:
tc                    100% |*****************************|   353 KB    00:00
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
```

```
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 2230 bytes 31 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 389140 bytes 326 pkts (dropped 0, overlimits 0)
```

Widać, że cały ruch poszedł do uchwytu 30:, który identyfikuje pasmo z najmniejszym priorytetem, tak jak chcieliśmy. Żeby sprawdzić, czy ruch interaktywny faktycznie trafia do wyższych pasm, wygenerujemy trochę takiego ruchu:

```
# tc -s qdisc ls dev eth0
qdisc sfq 30: quantum 1514b
 Sent 384228 bytes 274 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 20Kbit burst 1599b lat 667.6ms
 Sent 2640 bytes 20 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 14926 bytes 193 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap  1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 401836 bytes 488 pkts (dropped 0, overlimits 0)
```

Zadziałało - cały dodatkowy ruch trafił do 10:, która jest naszą kolejką o najwyższym priorytecie. Żaden ruch nie trafił do niższych priorytetów, które otrzymały przed chwilą cały ruch wygenerowany przez ‘scp’.

Sławna qdisc CBQ

Tak jak wspomniano wcześniej, kolejka CBQ jest najbardziej skomplikowaną, tą o której jest najwięcej szumu a jednocześnie najmniej ludzi rozumie co robi i jak skonfigurować ją dokładnie tak jak chcemy. Nie dzieje się tak dlatego, że autorzy byli złośliwi lub niekompetentni - przeciwnie, po prostu algorytm CBQ nie jest zbyt precyzyjny i niezbyt pasuje do sposobu w jaki działa Linuks.

Poza byciem kolejką z klasami, CBQ zajmuje się również kształtowaniem ruchu i to właśnie tego nie robi zbyt dobrze. Powinna pracować w ten sposób: jeśli chcesz ograniczyć ruch 10mbit/s do 1mbit/s, łącze powinno być przez 90% bezczynne. Jeśli nie jest, musimy je dławić by BYŁO w 90% czasu faktycznie bezczynne.

Jest to raczej trudne do zmierzenia, więc CBQ pobiera czas bezczynności z ilości mikrosekund pomiędzy wywołaniami sprzętowymi o więcej danych. Po połączeniu tych informacji, CBQ aproksymuje jak bardzo łącze jest zajęte.

Nie jest to raczej ostrożne i nie zawsze prowadzi do poprawnych rezultatów. Na przykład, jaka jest prędkość łącza na interfejsie, który nie potrafi wysłać pełnych 100mbit/s danych - być może z powodu źle zaimplementowanego sterownika? Karty sieciowe PCMCIA również nigdy nie osiągają 100mbit/s ponieważ nie projektowano do tego tej szyny i ponownie - jak skalkulować czas bezczynności?

Robi się jeszcze gorzej, jeśli rozważymy trochę nietypowe urządzenia sieciowe, takie jak PPP ponad Ethernetem czy PPTP ponad TCP/IP. Efektywna przepustowość w tym przypadku to tak naprawdę miara tego jak dobrze działają ‘rurki’ (ang. „pipe”) do przestrzeni użytkownika - a działają bardzo wydajnie.

Ludzie, którzy prowadzili pomiary stwierdzili, że CBQ nie zawsze jest bardzo dokładne a czasami w ogóle podaje nieadekwatne rezultaty.

W większości wypadków, działa jednak dobrze. Z dokumentacją, którą masz w ręku, powinieneś porawdzić sobie ze skonfigurowaniem jej tak, by działała dobrze.

Szczegóły kształtowania ruchu przez CBQ

Tak jak wcześniej napisano, CBQ dba cały czas o to, by łącze było przez odpowiedni czas bezczynne i dzięki temu prawdziwa przepustowość spadła do skonfigurowanej wielkości. Żeby to robić kalkuluje czas, który powinien upłynąć pomiędzy kolejnymi pakietami.

W czasie pracy, efektywny czas bezczynności mierzony jest za pomocą ‘wykładnika ważonego średniego przesyłu’ (ang. „exponential weighted moving average”, EWMA), który traktuje świeże pakiety jako wykładniczo ważniejsze niż starsze. Tak samo kalkulowany jest poziom obciążenia stacji uniksowych.

Wyliczony czas bezczyności odejmowany jest od wartości wyliczonej przez EWMA i wynik nazywa się średnim czasem bezczynności - ‘avgidle’. Dokładnie załadowane łącze ma ten czas równy zero: pakiety docierają dokładnie co jeden wyliczony interwał czasowy.

Przeładowane łącze ma ujemny średni czas bezczynności, a jeśli wartość ta jest duża, CBQ zamyka je na chwilę i oznacza to jako ‘przekroczenie limitu’ (ang. „overlimit”).

Odpowiednio, łącze puste ma duży średni czas bezczynności, który po paru godzinach ciszy umożliwiłby zajęcie nieskończonej przepustowości. By temu zapobiec, wartość średniego czasu bezczynności ogranicza się odgórnie parametrem ‘maxidle’.

Jeśli dojdzie do przekroczenia limitu, CBQ zdławi się na dokładnie tyle czasu ile zostało wyliczone pomiędzy pakietami a następnie przepuści jeden pakiet i zdławi się znowu. Sprawdź znaczenie parametru ‘minburst’.

Poniższe parametry można podać by skonfigurować kształtowanie ruchu:

avpkt

> Średnia wielkość pakietu, mierzona w bajtach. Potrzebna do wyliczania ‘maxidle’, który jest wyliczany z ‘maxburst’, który z kolei podawany jest w pakietach.

bandwidth - pasmo

> Fizyczne pasmo twojego urządzenia, potrzebne do wyliczania czasu bezczynności.

cell - komórka

> Czas który zajmuje pakietowi przesłanie przez urządzenie może rosnąć krokowo, na podstawie rozmiaru pakietu. Pakiety o wielkościach 800 i 806 bajtów mogą być wysyłane dokładnie tyle samo czasu - parametr ten kontroluje ziarnistość. Zwykle ustawiany na ‘8’. Musi być całkowitą potęgą dwójki.

maxburst

> Ta ilość pakietów służy do wyliczania ‘maxidle’ tak, by gdy ‘avgidle’ jest równe ‘maxidle’, ta ilość pakietów mogła zostać wysłana dodatkowo zanim ‘avgidle’ spadnie do 0. Ustaw tą wartość wyżej by być bardziej tolerancyjnym dla dodatkowych serii (ang. „bursts”). Nie możesz ustawić bezpośrednio ‘maxidle’ - możesz to zrobić tylko przez ten parametr.

minburst

> Jak wspomniano wcześniej, CBQ musi dławić ruch w przypadku przekroczenia limitu. Idealnym rozwiązaniem byłoby robienie tego na dokładnie wyliczony czas a następnie wysłać 1 pakiet. Kernele uniksa generalnie mają problemy z planowaniem zadań krótszych niż 10ms, więc lepiej jest dławić ruch na dłuższy okres, następnie przepuszczać ilość pakietów określoną przez parametr ‘minburst’ w jednym ruchu a potem zasypiać na ‘minburst’-razy dłużej.

Czas czekania nazywa się ‘czasem wolnym’ (ang. „offtime”). Wyższe wartości ‘minburst’ prowadzą do dokładniejszego kształtowania ruchu na dłuższą metę, ale jednocześnie do większych serii pakietów w skali milisekundowej.

minidle

Jeśli ‘avgidle’ ma wartość poniżej 0, oznacza to stan przekroczenia limitu i trzeba czekać dopóki ‘avgidle’ będzie na tyle duże, by wysłać jeden pakiet. By zapobiec nagłemu wypuszczeniu serii przy zamykaniu połączenia na określony okres czasu, ‘avgidle’ ustawiany jest na ‘minidle’ jeśli wartość ‘avgidle’ spadnie za nisko.

Wartość ‘minidle’ podaje się w ujemnych mikrosekundach, więc 10 oznacza że ‘avgidle’ wynosi -10 mikrosekund.

mpu

‘Minimalny rozmiar pakietu’ (ang. „Minimum packet size”) - wymagany, ponieważ nawet pakiety zawierające zero danych wyrównywane są do 64 bajtów w ethernecie i w związku z tym zajmują określony czas i pasmo podczas transmisji. CBQ musi wiedzieć ile wynosi rozmiar pakietu by poprawnie wyliczać wartość czasu bezczynności.

rate - częstotliwość

Wymagana częstotliwość ruchu opuszczającego qdisc - to właśnie jest regulator szybkości!

Wewnętrznie, CBQ ma masę parametrów konfigurujących. Na przykład, klasy o tórych wiadomo, że nie zbierają danych do kolejkowania nie są o takie dane odpytywane. Klasy zajmujące się ruchem nadmiarowym (ponad ustalonym limitem) ograniczane są dodatkowo przez zmniejszenie ich efektywnego priorytetu. Wszystko bardzo mądre i skomplikowane.

Zachowanie CBQ z klasami

Poza kształowaniem ruchu, za pomocą wspomnianych wyliczeń czasu bezczynności, CBQ zachowuje się również jak kolejka PRIO w tym sensie, że klasy mogą mieć różne priorytety i te o niższych priorytetach będą odpytywane przed tymi o wysokich priorytetach.

Za każdym razem gdy warstwa sprzętowa zażąda pakietu do wysłania w sieć, zaczyna się ‘ważony proces round-robin’ (ang. „weighted round robin process”, WRR) rozpoczynający się od klas z najniższymi priorytetami.

Są one grupowane i odpytywane czy mają jakieś dane do wysłania. Jeśli tak, dane zwracane są do pytającego. Po tym, jak klasa otrzyma zgodę na zdjęcie z kolejki określonej liczby bajtów, sprawdzana jest następna klasa z tym samym priorytetem.

Poniższe parametry kontrolują proces WRR:

allot

Kiedy CBQ proszone jest o pakiet do wysłania przez interfejs, sprawdzi wszystkie kolejki wewnętrzne (w klasie) w kolejności parametru ‘priorytet’. Za każdym razem gdy kolejna klasa dostaje swoją kolej, może wysłać tylko ograniczoną ilość danych. Parametr ‘allot’ jest podstawową jednostką tej ilości. Zajrzyj od opisu parametru ‘weight’ po więcej informacji.

prio

Kolejka CBQ może zachowywać się jak qdisc PRIO. Wewnętrzne klasy z niższymi priorytetami sprawdzane są pierwsze (jeśli mają jakiś ruch), inne klasy nie są odpytywane o ruch.

weight

'Waga' wspomaga proces WRR. Każda klasa otrzymuje szansę wysłania danych gdy przyjdzie jego kolej. Jeśli posiadasz klasy z wyraźnie większą przepustowością niż inne istnieje powód, by pozwolić im wysłać więcej danych w jednym ruchu niż innym klasom.

CBQ dodaje wszystkie wagi w klasach podrzędnych i normalizuje je, więc widzisz wartości arbitralne: tylko 'współczynniki' (ang. „ratio") są ważne. Generalnie używa się wzoru 'częstotliwość/10' i zdaje się to sprawdzać dobrze. Obliczona waga jest mnożona przez parametr 'allot' by sprawdzić, jak dużo danych można wysłać w jednym ruchu.

Zwróć uwagę na to, że wszystkie klasy w obrębie hierarchii CBQ współdzielą ten sam starszy numer!

Parametry CBQ określające możliwości pożyczania i współdzielenia łącza

Poza czystym ograniczaniem określonych rodzajów ruchu możliwe jest również określanie, które klasy mogą pożyczać pasma z innych klas lub współdzielić.

Isolated/sharing

Klasa skonfigurowana z parametrem 'wyizolowana' (ang. „isolated") nie pożycza przepustowości rodzeństwu. Użyj go jeśli masz konkurujące lub wzajemnie nieprzyjazne działy (lub użytkowników, aplikacje itp.) na jednym łączu.

Program **tc** umożliwa również ustawienie parametru 'współdzieląca' (ang. „sharing"), co jest odwrotnością wyizolowanej.

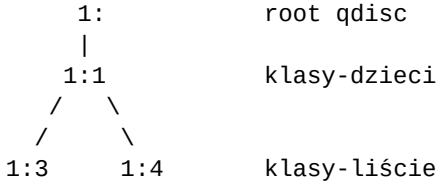
bounded/borrow

Klasa może być 'ograniczona' (ang. „bounded"), co oznacza, że nie będzie próbowała pożyczać od rodzeństwa. Możliwy jest również parametr 'pożyczająca' (ang. „borrow"), który jest dokładnie odwrotnością tego zachowania.

Typową sytuacją jest, gdy obie jednostki organizacyjne są zarówno 'wyizolowane' jak i 'ograniczone', co oznacza, że są ograniczone do przydzielonej częstotliwości i nie będą próbowały nic pożyczać.

W obrębie takiej klasy pojedyńczej jednostki, mogą znaleźć się klasy, którym zezwolimy na wymianę przepustowości.

Przykładowa konfiguracja



```
          |       |
         30:     40:       qdiscs
        (sfq)   (sfq)
```

Konfiguracja ogranicza ruch serwera WWW do 5mbit, ruch SMTP do 3mbit. Razem, nie mogą zająć więcej niż 6mbit. Mamy 100mbit'ową kartę sieciową oraz klasy, które mogą pożyczać od siebie pasmo.

```
# tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit         \
  avpkt 1000 cell 8
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit  \
  rate 6Mbit weight 0.6Mbit prio 8 allot 1514 cell 8 maxburst 20      \
  avpkt 1000 bounded
```

Ta część konfiguruje klasę-korzeń i odpowiednią klasę 1:0. Klasa 1:1 jest ograniczona, więc ogólnie przepustowość nie może przekroczyć progu 6mbit.

Jak już wcześniej wspomniano, CBQ wymaga wielu regulacji. Wszystkie parametry wyjaśniono wyżej. Odpowiadająca konfiguracja HTB jest o wiele prostsza.

```
# tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit  \
  rate 5Mbit weight 0.5Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
# tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit  \
  rate 3Mbit weight 0.3Mbit prio 5 allot 1514 cell 8 maxburst 20      \
  avpkt 1000
```

Widać nasze dwie klasy. Zauważ, że skalujemy wagę ze skonfigurowaną częstotliwością. Obie klasy nie są ograniczone, ale połączone z klasą 1:1, która jest ograniczona. Oznacza to, że suma obu klas nigdy nie wykroczy poza 6mbitów. Numery identyfikacyjne muszą znajdować się w obrębie jednego numeru starszego, równego nadrzędnej kolejce CBQ!

```
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
# tc qdisc add dev eth0 parent 1:4 handle 40: sfq
```

Obie klasy mają domyślnie kolejki FIFO. Zamieniliśmy je na SFQ, więc każdy przepływ danych jest traktowany równo.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 80 0xffff flowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip \
  sport 25 0xffff flowid 1:4
```

Powyższe komendy, dołączone bezpośrednio do korzenia rozsyłają ruch do odpowiednich kolejek.

Zauważ, że użyliśmy komendy **tc class add** by STWORZYĆ klasy w obrębie qdisc, ale by je dodać użyliśmy komendy **tc qdisc add**.

Możesz zastanawiać się co dzieje się z ruchem, który nie podlega klasyfikacji przez żadną z dwóch reguł. W tym wypadku, dane będą przetwarzane w obrębie klasy 1:0 i w związku z tym nie będą podlegać ograniczeniom.

Jeśli łącznie ruch smtp i WWW spróbują przekroczyć ustalony limit 6mbit/s, przepustowość zostanie podzielona zgodnie z wagą, to jest 5/8 dla serwera WWW i 3/8 dla serwera poczty.

Przy okazji można powiedzieć, że przy tych ustawieniach, serwer WWW zawsze otrzyma przynajmniej 5/8 * 6mbit = 3.75mbita.

Inne parametry CBQ: split & defmap

Jak już napisałem, kolejka z dyscypliną potrzebuje wywoływać filtry by określić do której klasy zostanie skolejkowany pakiet.

Poza wywołaniem filtra, CBQ oferuje inne opcje - defmap i split. Jest to trochę trudne do zrozumienia, ale nie konieczne. Ale ponieważ jest to jedyne miejsce, w którym w ogóle wyjaśnia się takie terminy, postaram się zrobić co w mojej mocy.

Ponieważ zwykle będziesz filtrował tylko na podstawie pola ToS, dostępna jest specjalna składnia. Kiedy CBQ potrzebuje określić gdzie skolejkować pakiet, sprawdza czy ten węzeł to ‘węzeł rozdzielczy’ (ang. „split node”). Jeśli tak, jedna z pod-kolejek określiła, że chciałaby otrzymywać wszystkie pakiety z określonym, skonfigurowanym priorytetem - który może pochodzić z pola ToS, lub opcji gniazd ustawionych przez aplikację.

Bity priorytetów pakietów są poddawane logicznej operacji AND z polem defmap by sprawdzić czy istnieje pasujący odpowiednik. Innymi słowy, jest to krótszy sposób na stworzenie bardzo szybkiego filtru, który pasuje tylko do określonych priorytetów. Mapa defmap równa 0xFF (heksdecymalnie) będzie pasowała do wszystkiego, a 0 do niczego. Przykładowa konfiguracja, by rozjaśnić to tłumaczenie:

```
# tc qdisc add dev eth1 root handle 1: cbq bandwidth 10Mbit allot 1514 \
  cell 8 avpkt 1000 mpu 64

# tc class add dev eth1 parent 1:0 classid 1:1 cbq bandwidth 10Mbit    \
  rate 10Mbit allot 1514 cell 8 weight 1Mbit prio 8 maxburst 20        \
  avpkt 1000
```

Standardowa preambuła CBQ. Nigdy nie przywyknę do ilości numerków, które należy podać!

defmap odwołuje się do bitów TC_PRIO, które zdefiniowane są jak następuje:

```
TC_PRIO..          Numer  Odpowiada w ToS
-----------------------------------------------
BESTEFFORT         0      Maksymalna niezawodność
FILLER             1      Minimalny koszt
BULK               2      Maksymalna przepustowość (0x8)
INTERACTIVE_BULK   4
INTERACTIVE        6      Minimalna zwłoka (0x10)
CONTROL            7
```

Numer TC_PRIO odpowiada bitom, liczonym od prawej. Zajrzyj do sekcji opisującej ‘pfifo_fast’ po więcej informacji w jaki sposób bity ToS mapowane są na priorytety.

A teraz klasy interaktywna i dla reszty ruchu:

```
# tc class add dev eth1 parent 1:1 classid 1:2 cbq bandwidth 10Mbit     \
  rate 1Mbit allot 1514 cell 8 weight 100Kbit prio 3 maxburst 20        \
  avpkt 1000 split 1:0 defmap c0

# tc class add dev eth1 parent 1:1 classid 1:3 cbq bandwidth 10Mbit     \
  rate 8Mbit allot 1514 cell 8 weight 800Kbit prio 7 maxburst 20        \
  avpkt 1000 split 1:0 defmap 3f
```

‘Kolejką rozdzielającą’ jest 1:0 i w niej dokonywany jest wybór. c0 to binarnie 11000000, 3F to 00111111, więc obie łącznie będą obejmować cały ruch. Pierwsza klasa pasuje jeśli ustawione są bity 6 i 7 odpowiadając tym samym ruchowi ‘interaktywnemu’ i ‘kontrolnemu’. Druga klasa odpowiada reszcie.

Węzeł 1:0 ma teraz tabelę taką jak ta:

```
priorytet  wyślij do
0            1:3
1            1:3
2            1:3
3            1:3
4            1:3
5            1:3
6            1:2
7            1:2
```

Co więcej, możesz również wydać polecenie ‘zmiany maski’, które określa dokładnie które priorytety chciałbyś zmienić. Potrzebujesz tego tylko gdy używasz **tc class change**. Na przykład, by dodać ruch typu ‘best effort’ do kolejki 1:2, możemy napisać tak:

```
# tc class change dev eth1 classid 1:2 cbq defmap 01/01
```

Mapa priorytetów nad 1:0 wygląda teraz tak:

```
priorytet  wyślij do
0            1:2
1            1:3
2            1:3
3            1:3
4            1:3
5            1:3
6            1:2
7            1:2
```

FIXME: nie testowałem **tc class change**, przeglądałem tylko źródła.

Hierarchiczne Wiadro Żetonów (ang. „Hierarchical Token Bucket”)

Martin Devera (<devik>) słusznie zauważył, że CBQ jest skomplikowane i nie wydaje się zoptymalizowane do zastosowania w wielu typowych sytuacjach. Jego hierarchiczne podejście jest bardzo dobrze dostosowane do konfiguracji, w których masz określone pasmo sieciowe do podzielenia wśród różnych zastosowań. Przy okazji wiesz ile każde powinno dostać i mniej więcej ile mogą od siebie pożyczać.

HTB działa tak jak CBQ, ale nie wykonuje przeliczania czasu bezczynności. Zamiast tego, funkcjonuje jako TBF z klasami - stąd jej nazwa. Ma tylko parę parametrów, opisanych dokładniej pod tym adresem (http://luxik.cdi.cz/~devik/qos/htb/).

Rozbudowa konfiguracji HTB doskonale się skaluje przy wzroście skomplikowania. Jeśli chodzi o CBQ, to wiadomo, że jest już skomplikowana na początku! HTB3 (zajrzyj pod ten adres

(http://luxik.cdi.cz/~devik/qos/htb/) aby sprawdzić informacje o wersjach HTB) jest obecnie częścią oficjalnych źródeł jądra (od 2.4.20-pre1 i 2.5.31). Prawdopodobnie jednak będziesz musiał nadal pobrać załataną wersję **tc** do obsługi HTB3: wersje oprogramowania w jądrze i w przestrzeni użytkownika muszą być w tej samej wersji lub **tc** po prostu nie zadziała.

Jeśli akurat masz nowy kernel, lub jesteś w stanie go odpowiednio załatać, powinieneś poważnie rozważyć użycie HTB.

Przykładowa konfiguracja

Funkcjonalnie, praktycznie identyczny przykład z tym dla CBQ z sekcji powyżej:

```
# tc qdisc add dev eth0 root handle 1: htb default 30

# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k

# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

Autor rekomenduje następnie użycie SFQ poniżej tych klas:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Dodajmy filtry, które skierują ruch do odpowiednich klas:

```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 25 0xffff flowid 1:20
```

I to wszystko - bez niewyjaśnionych numerków i nieudokumentowanych parametrów.

HTB z pewnością wygląda pięknie - jeśli 10: i 20: obie mają swoją gwarantowaną przepustowość i zostało coś do podzielenia, pożyczają w stosunku 5:3, tak jakbyś mógł się spodziewać.

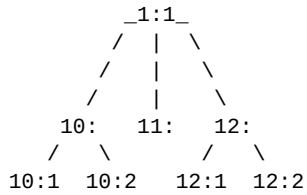
Ruch niesklasyfikowany do żadnej klasy kierowany jest do 30:, która ma tylko trochę własnej przepustowości ale może pożyczać wszystko co aktualnie jest niewykorzystane. Ponieważ używamy w środku SFQ, mamy zapewniony sprawiedliwy dostęp za darmo!

Klasyfikowanie pakietów filtrami

By określić, która klasa powinna zająć się pakietem wywoływany jest tzw. ‘łańcuch klasyfikacyjny’ (ang. „classifier chain”) za każdym razem gdy wymagane jest dokonanie wyboru. Łańcuch ten składa się ze wszystkich filtrów dołączonych do kolejek z klasami, które muszą dokonać wyboru.

Przypomnijmy drzewo, które drzewem nie jest:

```
                root 1:
                  |
```



W momencie skolejkowania pakietu, każda gałąź łańcucha filtrów konsultowana jest pod kątem określonych instrukcji. Typowa konfiguracja mogłaby przefiltrować pakiet z 1:1 do 12:, a następnie z 12: do 12:2.

Możesz również dołączyć tą drugą regułę do 1:1, ale uzyskasz większy zysk wykonując bardziej specyficzne testy na dole łańcucha.

Nie możesz również filtrować pakietów ‘w górę’. Przy użyciu HTB powinieneś dołączać wszystkie filtry do korzenia głównego!

I znowu - pakiety są kolejkowane tylko w dół! W momencie gdy są zdejmowane z kolejki kierowane są do góry, do interfejsu. Nie spadają w dół drzewa do karty sieciowej!

Trochę prostych przykładów filtrowania

Jak wyjaśniono w rozdziale o Klasyfikatorze, możesz sprawdzić praktycznie wszystko, używając skomplikowanej składni. Na początek, pokażemy jak wskazać podstawowe rzeczy co na szczęście robi się raczej prosto.

Powiedzmy, że mamy kolejkę PRIO nazwaną 10:, zawierającą trzy klasy; chcemy przypisać cały ruch z i do portu 22 do pasma o najwyższym priorytecie

```
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip dport 22 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 1 u32 match \
  ip sport 80 0xffff flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2 flowid 10:2
```

Co to znaczy? Dołącz do eth0, węzeł 10: filtr u32 z priorytetem 1, który pasuje dokładnie do portu przeznaczenia 22 i wyślij pakiety pasujące do pasma 10:1. W następnej linijce powtarzamy to dla portu 80. Ostatnia komenda określa, że wszystko to co nie zostało wskazane wprost powinno trafić do pasma 10:2, następnego w kolejności jeśli chodzi o priorytet.

Musisz dodać ‘eth0’, lub jakkolwiek nazywa się twój interfejs, ponieważ każdy ma swoją własną, unikalną przestrzeń uchwytów.

By wybrać adres IP, użyj:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip dst 4.3.2.1/32 flowid 10:1
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 \
  match ip src 1.2.3.4/32 flowid 10:1
# tc filter add dev eth0 protocol ip parent 10: prio 2      \
  flowid 10:2
```

Polecenia te przydzielają ruch do 4.3.2.1 i z 1.2.3.4 do kolejki z najwyższym priorytetem a resztę ruchu do kolejki z następnym w kolejności priorytetem.

Możesz łączyć warunki - by pasowały do ruchu z 1.2.3.4 z portu 80, wpisz:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 1 u32 match ip src 4.3.2.1/32
  match ip sport 80 0xffff flowid 10:1
```

Wszystkie komendy filtrujące, których będziesz normalnie potrzebował

Większość komend kształtujących ruch zaczyna się od następującego ciągu:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 ..
```

Są one nazywane tzw. ‘testami u32’, które mogą pasować do KAŻDEJ części pakietu.

testy na adresach źródłowym/docelowym

Maska źródłowa ‘match ip src 1.2.3.0/24’, maska docelowa ‘match ip dst 4.3.2.0/24’. By wskazać pojedyńczy host, użyj /32 lub po prostu pomiń maskę.

testy na portach źródłowym/docelowym, wszystkie protokoły IP

Port źródłowy: ‘match ip sport 80 0xffff’, port docelowy: ‘match ip dport 0xffff’

testy na protokół (tcp, udp, icmp, gre, ipsec)

Użyj numerków z `/etc/protocols`, na przykład ICMP ma przydzielony numer 1, więc: ‘match ip protocol 1 0xff’.

testy na znacznik ‘fwmark’

Możesz oznaczać pakiety przy użyciu np. ipchains; taki znacznik ‘przeżywa’ routing pomiędzy interfejsami. Jest to użyteczne dla np. kształtowania ruchu na interfejsie eth1, przychodzacego z eth0. Składnia:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 1 handle 6 fw flowid 1:1
```

Zwróć uwagę, że to nie jest test u32!

Pakiet możesz oznaczyć w ten sposób:

```
# iptables -A PREROUTING -t mangle -i eth0 -j MARK --set-mark 6
```

Numer 6 w powyższym przykładzie jest zupełnie przypadkowy.

Jeśli nie chcesz zrozumieć całej składni filtrów, użyj iptables i naucz się korzystać tylko ze znaczników.

testy na polu ToS

By wybrać ruch interaktywny z minimalną zwłoką:

```
# tc filter add dev ppp0 parent 1:0 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff \
     flowid 1:4
```

Dla ruchu typowego użyj 0x08 0xff.

Po więcej komend filtrujących, zajrzyj do rozdziału o Zaawansowanych Filtrach.

Pośrednie urządzenie kolejkujące (ang. „The Intermediate queueing device”, IMQ

Pośrednie urządzenie kolejkujące nie jest qdisc, ale jego użycie jest ściśle związane z dyscyplinami kolejkowania. W Linuksie, qdisc doczepiane są do urządzeń sieciowych i wszystko co ma zostać skolejkowane do urządzenia jest najpierw kolejkowane do qdisc. Po zapoznaniu się z tą koncepcją, widać dwa ograniczenia:

1. Możliwe jest kształtowanie tylko ruchu wychodzącego (istnieje qdisc dla ruchu przychodzącego ale jej możliwości są bardzo ograniczone w porównaniu do zwykłych dyscyplin kolejkowania z klasami).

2. Dana qdisc widzi ruch tylko na pojedyńczym interfejsie, nie można nakładać globalnych ograniczeń.

IMQ umożliwia rozwiązanie tych dwóch problemów. Krótko mówiąc, możesz włożyć cokolwiek tylko chcesz do qdisc. Specjalnie oznaczone pakiety są przechwytywane w wywołaniach netfiltera NF_IP_PRE_ROUTING oraz NF_IP_POST_ROUTING i przechodzą przez qdisc doczepioną do urządzenia IMQ. Do oznaczania pakietów używa się celu (ang. „target”) w iptables.

Umożliwia to kształtowanie wchodzącego ruchu, ponieważ możesz kolejno oznaczać nadchodzące pakiety i/lub traktować interfejsy jak klasy by ustalać limity globalne. Można również robić wiele innych rzeczy, takich jak wkładanie tylko ruchu HTTP do qdisc, tylko nowych połączeń itp.

Przykładowa konfiguracja

Pierwszą rzeczą która może przyjść do głowy jest użycie kształtowania ruchu wchodzącego by dać sobie gwarantowane pasmo ;) Konfiguracja odbywa się dokładnie tak jak dowolnego innego interfejsu:

```
tc qdisc add dev imq0 root handle 1: htb default 20

tc class add dev imq0 parent 1: classid 1:1 htb rate 2mbit burst 15k

tc class add dev imq0 parent 1:1 classid 1:10 htb rate 1mbit
tc class add dev imq0 parent 1:1 classid 1:20 htb rate 1mbit

tc qdisc add dev imq0 parent 1:10 handle 10: pfifo
tc qdisc add dev imq0 parent 1:20 handle 20: sfq

tc filter add dev imq0 parent 10:0 protocol ip prio 1 u32 match \
              ip dst 10.0.0.230/32 flowid 1:10
```

W tym przykładzie u32 został użyty do sklasyfikowania ruchu. Inne klasyfikatory powinny również działać tak jak je opisano. Następnie musimy wybrać i oznaczyć ruch, który ma trafić do urządzenia ‘imq0’.

```
iptables -t mangle -A PREROUTING -i eth0 -j IMQ --todev 0

ip link set imq0 up
```

Cel IMQ w iptables jest prawidłowy tylko w łańcuchach PREROUTING i POSTROUTING tabeli przekształceń (ang. „mangle”). Jego składnia wygląda następująco:

```
IMQ [ --todev n ]       n : numer urządzenia IMQ
```

Dostępny jest również cel dla wersji iptables dla IPv6 (ip6tables).

Zwróć uwagę na to, że ruch nie jest kolejkowany gdy cel zostaje trafiony, ale dopiero później. Dokładne miejsce, w którym ruch wchodzi do urządzenia IMQ zależy od kierunku ruchu (wejściowy/wyjściowy). Poniżej predefiniowane stałe używane przez iptables:

```
enum nf_ip_hook_priorities {
        NF_IP_PRI_FIRST = INT_MIN,
        NF_IP_PRI_CONNTRACK = -200,
        NF_IP_PRI_MANGLE = -150,
        NF_IP_PRI_NAT_DST = -100,
        NF_IP_PRI_FILTER = 0,
        NF_IP_PRI_NAT_SRC = 100,
        NF_IP_PRI_LAST = INT_MAX,
};
```

Dla ruchu przychodzącego, IMQ rejestruje się z priorytetem NF_IP_PRI_MANGLE + 1, co oznacza że pakiety trafiają do niej zaraz po przejściu łańcucha PREROUTING w tablicy przekształceń.

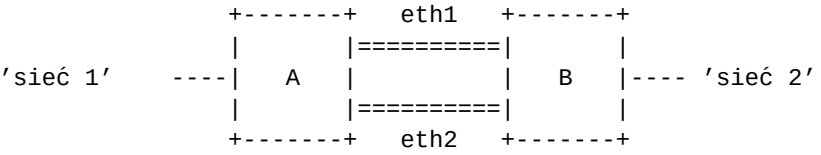
Dla ruchu wychodzącego, IMQ używa NF_IP_PRI_LAST, który bierze pod uwagę fakt, że pakiety odrzucone przez tablice filtrowania nie zajmują dostępnego pasma.

Łaty i trochę informacji znaleźć można pod tym adresem (http://luxik.cdi.cz/~patrick/imq/).

Rozdział 10. Rozkładanie obciążenia na wiele interfejsów

Istnieje kilka sposobów na zrealizowanie tego zagadnienia. Jednym z najprostszych jest ‘TEQL’ - ‘Trywialny wyrównywacz łącza’ (ang. „Trivial link equalizer”). Podobnie jak większość rzeczy, które działają z kolejkami - musi działać z obu stron. Obie strony łącza będą musiały współpracować dla pełnego efektu.

Wyobraź sobie taką sytuację:



A i B to routery i załóżmy na moment, że pracują w oparciu o Linux. Jeśli ruch przechodzi z sieci 1 do sieci 2, ruter A będzie musiał rozłożyć pakiety na oba połączenia z ruterem B. Ruter B musi wiedzieć, że taka sytuacja ma miejsce i być na to przygotowany (skonfigurowany). Dokładnie taka sama sytuacja zachodzi w drugą stronę, gdy pakiety z sieci 2 przechodzą do sieci 1 - muszą zostać wysłane zarówno przez interfejsy eth1 jak i eth2.

Część zajmująca się dystrybucją pakietów obsługiwana jest przez urządzenie ‘TEQL’, które konfiguruje się w następujący sposób:

```
# tc qdisc add dev eth1 root teql0
# tc qdisc add dev eth2 root teql0
# ip link set dev teql0 up
```

Nie zapomnij wydać polecenia **ip link set up**!

Trzeba to oczywiście wykonać na obu komputerach. Urządzenie teql0 to dystrybutor działający na zasadzie round-robin w oparciu o interfejsy eth1 i eth2. Żadne dane nigdy nie przychodzą przez urządzenie teql, istnieje ono tylko w postaci ‘surowej’ na interfejsach eth1 i eth2.

Mamy urządzenia, potrzebujemy teraz odpowiedniego routingu. Jednym ze sposobów, jest przedzielenie sieci /31 do obu łączy i do urządzenia teql0:

Na routerze A:

```
# ip addr add dev eth1 10.0.0.0/31
# ip addr add dev eth2 10.0.0.2/31
# ip addr add dev teql0 10.0.0.4/31
```

Na routerze B:

```
# ip addr add dev eth1 10.0.0.1/31
# ip addr add dev eth2 10.0.0.3/31
# ip addr add dev teql0 10.0.0.5/31
```

Router A powinien móc wykonać ping do 10.0.0.1, 10.0.0.3 i 10.0.0.5 przez dwa fizyczne łącza i jedno urządzenie wyrównujące. Router B powinien móc wykonać ping do 10.0.0.0, 10.0.0.2 i 10.0.0.4.

Jeśli to działa, router A powinien ustawić trasę 10.0.0.5 na domyślną do sieci 2, a router B trasę 10.0.0.4 do sieci 1. W tym szczególnym przypadku, w którym sieci 1 jest twoją siecią domową, a sieć 2 Internetem, router A powinien ustawić sobie 10.0.0.5 jako domyślną bramkę.

Problemy

Nic nie jest takie proste jak wygląda. eth1 i eth2 na obu routerach muszą mieć wyłączone filtrowanie dla ścieżek powrotnych, ponieważ w innym przypadku będą odrzucały pakiety przeznaczone dla innych adresów IP niż ich własne:

```
# echo 0 > /proc/sys/net/ipv4/conf/eth1/rp_filter
# echo 0 > /proc/sys/net/ipv4/conf/eth2/rp_filter
```

Następną sprawą jest zmiana kolejności pakietów. Powiedzmy, że musimy wysłać 6 pakietów z A do B. eth1 obsłuży 1, 3 i 5. eth2 obsłuży 2, 4 i 6. W idealnym świecie, router B otrzyma je w takiej kolejności: 1, 2, 3, 4, 5, 6. Ale w świecie rzeczywistym większe prawdopodobieństwo ma inna sytuacja, np. taka kolejność: 2, 1, 4, 3, 6, 5. Problem polega na tym, że myli to TCP/IP. O ile nie jest to problem dla łączy przez które przechodzi wiele różnych sesji TCP/IP, to nie będziesz w stanie połączyć obu fizycznych łączy w jedno logiczne by uzyskać np. szybsze transfery ftp chyba, że wysyłającym lub odbierającym systemem operacyjnym jest Linuks, który nie daje się takim prostym problemom.

Oczywiście, dla wielu aplikacji, równoważenie obciążenia łącza to wspaniały pomysł.

Inne możliwości

William Stearns zastosował zaawansowaną konfigurację by połączyć wiele niezależnych łącz do Internetu w jedność. Informacje o tej konfiguracji można znaleźć na jego stronie (http://www.stearns.org/tunnel/).

Być może również w tym HOWTO znajdzie się kiedyś więcej informacji o tym rozwiązaniu.

Rozdział 11. Netfilter & iproute - oznaczanie pakietów

Jak na razie, widzieliśmy jak działa iproute i wspomnieliśmy o netfilter. Jest to dobry moment by przejrzeć kolekcję poradników Rustyego: (http://netfilter.samba.org/unreliable-guides/). Sam pakiet netfilter można znaleźć pod tym (http://netfilter.filewatcher.org/).

Netfilter pozwala na filtrowanie pakietów i grzebanie w ich nagłówkach. Jedną ze specjalności jest możliwość znaczenia pakietów numerami. Można to zrobić posługując się opcją **--set-mark**.

Jako przykład, to polecenie znaczy wszystkie pakiety wysłane na port 25, czyli wychodzące połączenie pocztowe SMTP:

```
# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 25 \
 -j MARK --set-mark 1
```

Teraz załóżmy, że mamy wiele połączeń, jedno bardzo szybkie (i bardzo drogie, rozliczane za megabajty) i drugie wolniejsze, ale prawie za darmo. Chcielibyśmy zapewne, by wychodząca poczta przechodziła przez to tańsze.

Oznaczyliśmy już takie połączenia numerem ‘1’, teraz poinstruujemy bazę danych polityki routingu by odpowiednio kierowała ruchem:

```
# echo 201 mail.out >> /etc/iproute2/rt_tables
# ip rule add fwmark 1 table mail.out
# ip rule ls
0:      from all lookup local
32764:      from all fwmark        1 lookup mail.out
32766:      from all lookup main
32767:      from all lookup default
```

Teraz wygenerujemy tabelę mail.out dla routowania na wolne, ale tanie łącze:

```
# /sbin/ip route add default via 195.96.98.253 dev ppp0 table mail.out
```

I to już. Jeśli chcielibyśmy zrobić jakieś wyjątki jest wiele różnych sposobów. Możemy zmodyfikować polecenie netfilter, by wyłączyło pewne hosty, możemy wstawić regułę z niższym priorytetem, która wskazuje na główną tabelę dla wyłączonych z grupy hostów.

Możemy również używać tej cechy by honorować bity ToS, przez oznaczanie pakietów różnymi numerami w zależności od zdefiniowanych w tabeli ToS i jednocześnie stworzenie reguł filtrujących na tej podstawie. W ten sposób możemy np. przeznaczyć połączenie ISDN na dedykowane sesje interaktywne.

Oczywiście, działa to również gdy prowadzimy NAT (maskaradę).

WAŻNE: Otrzymaliśmy informacje, że MASQ i SNAT kolidują ze sobą jeśli chodzi o znaczenie pakietów. Rusty Russell wyjaśnia to w tym poście (http://lists.samba.org/pipermail/netfilter/2000-November/006089.html). Musisz wyłączyć filtr kolejki wychodzącej by działało to poprawnie.

UWAGA: By oznaczać pakiety, musisz mieć włączone pewne opcje w konfiguracji jądra:

```
IP: advanced router (CONFIG_IP_ADVANCED_ROUTER) [Y/n/?]
```

```
IP: policy routing (CONFIG_IP_MULTIPLE_TABLES) [Y/n/?]
IP: use netfilter MARK value as routing key (CONFIG_IP_ROUTE_FWMARK) [Y/n/?]
```

Zajrzyj również do sekcja *Przezroczyste cache przy użyciu netfilter, iproute2, ipchains i squida* w Rozdział 15 w *Cookbook*.

Rozdział 12. Zaawansowane filtry i (re-)klasyfikowanie pakietów

Jak już wyjaśniono powyżej, filtry potrzebne są do klasyfikowania pakietów do podkolejek. Wywoływane są z kolejek z klasami.

Poniżej niekompletna lista klasyfikatorów:

fw

Podejmuje decyzję na podstawie tego, jak firewall oznaczył pakiet. Użycie go może być prostsze, niż uczenie się całej składni **tc**. Zajrzyj do rozdziału o kolejkowaniu.

u32

Podejmuje decyzję na podstawie pól w pakiecie (np. źródłowego adresu IP itd.).

route

Podejmuje decyzję na podstawie trasy, którą pakiet będzie routowany.

rsvp, rsvp6

Kieruje pakiety na trasy określone przez standard RSVP (http://www.isi.edu/div7/rsvp/overview.html). Użyteczne tylko jeśli kontrolujesz sieć na całej trasie pakiety - Internet nie respektuje RSVP.

tcindex

Używane w kolejce DSMARK, zajrzyj do odpowiedniej sekcji.

Zauważ, że generalnie jest wiele sposobów na który możesz sklasyfikować pakiet i w zasadzie to twoje preferencje decydują o tym, który system wykorzystać.

Klasyfikatory ogólnie rzecz biorąc, akceptują parę podstawowych argumentów. Wymieniono je tutaj dla wygody:

protocol

Protokół, który ten klasyfikator zaakceptuje - w zasadzie będziesz akceptował ruch IP. Wymagane.

parent

Oznaczenie, do którego ten klasyfikator będzie podpięty. Musi być już zdefiniowane dla istniejącej klasy. Wymagane.

prio

Priorytet klasyfikatora. Niższy numer będzie sprawdzany szybciej.

handle

Uchwyt oznacza różne rzeczy dla różnych filtrów.

Wszystkie następne sekcje zakładają, że próbujesz kształtować ruch skierowany do `HostA`. Zakłada się, że klasę-korzeń skonfigurowano na 1: i że wyselekcjonowany ruch chcesz wysyłać do 1:1.

Klasyfikator u32

Klasyfikator U32 jest w obecnej implementacji najbardziej zaawansowanym z dostępnych filtrów. W całości zbudowany w oparciu o tablice ‘mieszające’, zapewniające dużą wydajność nawet w przypadku dużej liczby reguł.

W swojej najprostszej postaci, filtr U32 jest listą rekordów, z których każdy składa się z dwóch pól: selektora i akcji. Selektor, opisany poniżej, jest porównywany do aktualnie przetwarzanego pakietu IP dopóki nie dojdzie do pierwszej zgodności. Wtedy wykonywana jest skojarzona z pasującym selektorem akcja. Najprostszym typem akcji byłoby skierowanie pakietu do zdefiniowanej klasy CBQ.

Linia poleceń programu **tc filter** używanego do konfigurowania filtrowania, składa się z trzech części: określenia filtra, selektora i akcji. Określenie filtra można opisać jako:

```
tc filter add dev IF [ protocol PROTO ]
                     [ (preference|priority) PRIO ]
                     [ parent CBQ ]
```

Pole `protocol` opisuje protokół, w stosunku do którego filtr będzie sprawdzany. Rozważamy tylko przypadek protokołu `ip`. Pole `preference` (`priority` można używać zamiennie) określa priorytet aktualnie definiowanego filtra. Jest to o tyle ważne, że możesz posiadać kilka filtrów (list reguł) z różnymi priorytetami. Każda lista będzie sprawdzana w kolejności w jakiej dodawano reguły, następnie przetwarzana będzie lista z mniejszym priorytetem (wyższym numerem preferencji). W końcu pole `parent` określa górne drzewo CBQ (np. 1:0) do którego powinien dołączyć się filtr.

Opcje opisane wyżej dotyczą wszystkich filtrów, nie tylko U32.

Selektor U32

Selektor U32 zawiera definicję wzorca, który zostanie sprawdzony z przetwarzanym pakietem. Dokładniej, definiuje które bity mają pasować z nagłówka pakietu - ale jest to bardzo potężne narzędzie. Spójrzmy na następujący przykład, wzięty wprost z rzeczywistego, całkiem skomplikowanego filtra:

```
# tc filter add dev eth0 protocol ip parent 1:0 pref 10 u32 \
  match u32 00100000 00ff0000 at 0 flowid 1:10
```

Na razie zostawmy pierwszą linię w spokoju - wszystkie parametry opisują tablice mieszające filtra. Skupmy się na linii selektora, zawierającej słowo `match`. Selektor wybiera nagłówki IP, których drugi bajt jest równy 0x10 (0010). Jak łatwo zgadnąć, numer 00ff jest maską testu, mówiącą filtrowi które bajty dokładnie dopasować. Wynosi 0xff, więc bajt ma być równy dokładnie 0x10. Słowo kluczowe `at` oznacza, że test rozpoczynany jest od określonego przesunięcia (w bajtach) -- w tym przypadku w stosunku do początku pakietu. Tłumacząc to na ludzki język, pakiet będzie pasował do filtra, jeśli pole ToS będzie miało ustawione bity ‘niska zwłoka’. Zanalizujmy następną regułę:

```
# tc filter add dev eth0 protocol ip parent 1:0 pref 10 u32 \
  match u32 00000016 0000ffff at nexthdr+0 flowid 1:10
```

Opcja `nexthdr` oznacza następny nagłówek zahermetyzowany w pakiecie IP, tzn. nagłówek wyższego protokołu. Test rozpocznie się od początku następnego nagłówka i dotyczyć będzie drugiego, 32-bitowego słowa w nagłówku. W protokołach TCP i UDP pole to zawiera port przeznaczenia. Numer podawany jest w formacie

big-endian, tzn. starsze bity pierwsze, więc czytamy to jako 0x16 czyli 22 decymalnie - port usługi SSH jeśli byłoby to TCP. Jak możesz zgadnąć, test jest bez sensu jeśli nie ma kontekstu i omówimy to później.

Jeśli zrozumiałeś wszystko co było wcześniej, łatwo możesz zrozumieć następujący selektor: `match c0a80100 ffffff00 at 16`. Chodzi o trzy-bajtowy test, począwszy od 17-tego bajtu licząc od startu nagłówka IP. Test pasuje dla pakietów, których docelowym adresem jest sieć 192.168.1.0/24. Po zanalizowaniu przykładów, możemy podsumować to, czego się nauczyliśmy.

Selektory ogólne

Selektory ogóle definiują wzór, maskę i przesunięcie (offset), które stosowane będą w stosunku do zawartości pakietu. Używając ich, można sprawdzić praktycznie dowolny bit w nagłówku IP, lub nagłówku wyższej warstwy. Są jednak trochę trudniejsze do spisania i odczytania, niż selektory specyficzne opisane niżej. Ich ogólna składnia wygląda tak:

```
match [ u32 | u16 | u8 ] WZÓR MASKA [ at PRZESUNIĘCIE | nexthdr+PRZESUNIĘCIE]
```

Jedno ze słów kluczowych - `u32`, `u16` lub `u8` określa długość wzorca w bitach. Następnie powinny pojawić się ‘wzór’ (ang. „pattern”) i ‘maska’, w długości zadeklarowanej wcześniej. Przesunięcie określane jest w bajtach. Jeśli podano dodatkowo `nexthdr+`, przesunięcie jest relatywne w stosunku do początku nagłówka wyższej warstwy.

Trochę przykładów:

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match u8 64 0xff at 8 \
     flowid 1:4
```

Pakiet będzie pasował do tej reguły, jeśli jego czas życia (TTL) jest równy 64. TTL to pole zaczynające się za 8-mym bajtem nagłówka IP.

Reguła dotyczy wszystkich pakietów TCP z ustawioną flagą ACK:

```
# tc filter add dev ppp14 parent 1:0 prio 10 u32 \
     match ip protocol 6 0xff \
     match u8 0x10 0xff at nexthdr+13 \
     flowid 1:3
```

Użyj tego selektora do wyboru pakietów z ustawioną flagą ACK krótszych niż 64 bajty.

```
# tc filter add dev ppp14 parent 1:0 protocol ip prio 10 u32 \
            match ip protocol 6 0xff \
            match u8 0x05 0x0f at 0 \
            match u16 0x0000 0xffc0 at 2 \
            match u8 0x10 0xff at 33 \
            flowid 1:3
```

Ta reguła dotyczy pakietów TCP z ustawioną flagą ACK i nie zawierających żadnych danych. Możemy zobaczyć tu przykład dwóch selektorów. Wynikiem finalnym będzie logiczna operacja AND na rezultatach ich obu. Jeśli

przyjrzymy się nagłówkowi IP, widzimy, że bit ACK jest drugim w kolejności, najstarszym bitem (0x10), w 14-tym bajcie nagłówka TCP (`at nexthdr+13`). Jeśli chodzi o drugi selektor, jeśli chcielibyśmy sobie utrudnić życie, moglibyśmy napisać `match u8 0x06 0xff at 9` zamiast używać selektora specyficznego w postaci `protocol tcp`. 0x06 to numer protokołu TCP a wartość ta zapisana jest w 10-tym bajcie nagłówka IP. Z drugiej strony w tym przykładzie nie możemy użyć żadnego specyficznego selektora dla pierwszego testu, ponieważ nie ma takiego selektora dla bitu ACK nagłówka TCP.

Filtr poniżej to zmodyfikowana wersja wcześniejszego. Różnica polega na tym, że nie sprawdza on długości nagłówka. Dlaczego? Ponieważ powyższy filtr działa tylko na systemach 32-bitowych.

```
tc filter add dev ppp14 parent 1:0 protocol ip prio 10 u32 \
     match ip protocol 6 0xff \
     match u8 0x10 0xff at nexthdr+13 \
     match u16 0x0000 0xffc0 at 2 \
     flowid 1:3
```

Selektory specyficzne

Poniższa tabela zawiera listę wszystkich selektorów specyficznych, które autor znalazł w źródłach programu **tc**. Mogą ułatwić ci życie i zwiększyć czytelność twojej konfiguracji.

FIXME: Tabela znajduje się w osobnym pliku - ‘selector.html’

FIXME: I nadal tylko po Polsku :-(

FIXME: Trzeba ją zsgmlizować

Trochę przykładów:

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
     match ip tos 0x10 0xff \
     flowid 1:4
```

FIXME: test ‘tcp dport’ nie działa tak jak opisano to poniżej:

Powyższa reguła będzie pasować do pakietów z polem ToS ustawionym na wartość 0x10. Zaczyna się ono w drugim bajcie pakiety i ma wielkość jednego bajtu, więc moglibyśmy napisać to za pomocą generalnego selektora: `match u8 0x10 0xff at 1`. Tkwi tu pewna podpowiedź - selektory specyficzne są zawsze tłumaczone na ogólne i w takiej formie zapisywane w pamięci jądra. Prowadzi to do konkluzji, że selektory `tcp` i `udp` są dokładnie takie same i dlatego nie możesz użyć pojedyńczej komendy `match tcp dport 53 0xffff` by wskazać pakiety TCP wysłane do określonego portu - pasować będą również pakiety UDP wysyłane do tego portu. Musisz pamiętać, żeby podać protokół na koniec poniższej reguły:

```
# tc filter add dev ppp0 parent 1:0 prio 10 u32 \
        match tcp dport 53 0xffff \
        match ip protocol 0x6 0xff \
        flowid 1:2
```

Klasyfikator `route`

Klasyfikator filtruje na podstawie rezultatów z tabeli routingu. Kiedy pakiet przemierza przez klasy i dociera do takiej, którą oznaczono filtrem ‘route’, dzieli pakiety na podstawie informacji z tabeli routingu.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 route
```

Dodajemy klasyfikator do węzła 1:0 rodzica z priorytetem 100. Kiedy pakiet dotrze do tego węzła (który jest korzeniem więc stanie się to od razu) sprawdzi tablicę routingu i jeśli znajdzie pasujący wpis wyśle pakiet do określonej klasy z priorytetem 100. Na koniec, by to zaczęło działać, należy dodać odpowiedni wpis do tabeli routingu. Sztuczka polega na tym, by zdefiniować ‘realm’ bazując a adresie źródłowym lub docelowym. Można to zrobić w ten sposób:

```
# ip route add Host/Sieć via Bramka dev Urządzenie realm NumerŚwiata
```

Na przykład, zdefiniujmy naszą sieć docelową 192.168.10.0 z numerem ‘realm’ równym 10:

```
# ip route add 192.168.10.0/24 via 192.168.10.1 dev eth1 realm 10
```

Przy dodawaniu filtrów tras, możemy użyć tych identyfikatorów by określić sieci lub komputery i jednocześnie wyspecyfikować jak trasy będą pasować do filtrów.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route to 10 classid 1:10
```

Powyższa reguła mówi, że pakiety przeznaczone do sieci 192.168.10.0 pasują do klasy o identyfikatorze 1:10.

Filtr tras może być również użyty do testowania tras źródłowych. Na przykład, mamy podsieć dołączoną do rutera Linuksowego przez interfejs eth2.

```
# ip route add 192.168.2.0/24 dev eth2 realm 2
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 \
  route from 2 classid 1:2
```

Filtr określa, że pakiety z podsieci 192.168.2.0 (realm 2) będą pasowały do klasy o identyfikatorze 1:2.

Filtry określające politykę

By umożliwić tworzenie jeszcze bardziej skomplikowanych konfiguracji, możesz zbudoać filtry, które będą pasowały tylko do określonej przepustowości. Możesz zadeklarować, że filtr przestaje działać powyżej określonej częstotliwości, lub że działa tylko gdy przekroczona zostanie pewna częstotliwość.

Jeśli więc zamierzasz ograniczyć ruch do 4mbit/s a obecnie odbywa się ruch do 5mbit/s, możesz przestać testować albo całe 5mbit/s albo tylko nadmiarowy 1mbit/s i wysłać 4mbit/s do skonfigurowanej klasy.

Jeśli pasmo przekracza skonfigurowaną wielkość, możesz odrzucić pakiet, przeklasyfikować go lub sprawdzić czy inny filtr nie będzie do niego pasował.

Sposoby na określenie polityki

Są w zasadzie dwa. Jeśli skompilowałeś kernel z opcją ‘szacowania’ (ang. „estimators”), może on dokonywać pomiaru dla każdego filtra sprawdzając, ile ruchu przez niego przechodzi. Są bardzo łagodne dla CPU, ponieważ zliczają 25 razy na sekundę ile przepływa danych i kalkulują na tej podstawie częstotliwość przepływu bitów.

Drugi sposób działa przez TBF, co oznacza, że żyje z twoim filtrem. TBF sprawdza tylko ruch DO górnej granicy skonfigurowanego pasma, a jeśli jest ono większe, można kontrolować tylko nadwyżkę.

Sposób z estymatorem kernela

Bardzo prosta sprawa, mamy do dyspozycji tylko jeden parametr: ‘avrate’. Albo wielkość ruchu pozostaje poniżej wartości tego parametru i filtr klasyfikuje ruch na podstawie numeru classid, albo wielkość ruchu przekracza wartość parametru i wtedy możemy określić akcję, która zostanie wykonana - domyślnie jest to ‘reklasyfikacja’.

Kernel używa algorytmu EWMA (wykładnika ważonego średniego przesyłu) dla kalkulacji aktualnego pasma i w związku z tym wyliczenia te są mniej czułe na krótkotrwałe serie.

Sposób z Token Bucket Filter

Używa następujących parametrów:

- buffer/maxburst
- mtu/minburst
- mpu
- rate

Zachowują się one dokładnie tak jak te opisane w sekcji opisującej TBF. Zauważ jednak, że jeśli ustawisz ‘mtu’ za nisko nie przejdą *żadne* pakiety, podczas gdy kolejka TBF obsługująca ruch przychodzący będzie przy takiej samej wartości tego parametru przepuszczać ruch wolniej.

Inna różnica polega na tym, że tutaj tylko przepuszczamy albo odrzucamy pakiety. Nie możemy ich w żaden sposób opóźniać.

Akcje do podjęcia przy przekroczeniu pasma

Jeśli twój filtr wykryje taką sytuację, może wykonać jedną z ‘akcji’. Obecnie, dostępne są trzy:

continue

Filtr nie pasuje, ale być może inne będą.

drop

: Bardzo surowa opcja, która po prostu odrzuci ruch wykraczający poza określoną wartość. Często używa się jej przy ruchu wchodzącym i przy ograniczonej liczbie użytkowników. Na przykład, możesz mieć serwer nazw, który nie daje sobie rady z ruchem powyżej 5mbit/s, więc filtr ustawiony na ruch przychodzący może zapewnić, że maszyna nie będzie próbowała obsłużyć więcej.

Pass/OK

: Przepuszcza ruch. Może być użyteczne do wyłączenia skomplikowanego filtra, który przydaje się w innej sytuacji.

reclassify

: Zwykle ogranicza się do sklasyfikowania ruchu jako typ best-effort. Jest to domyślna akcja.

Przykłady

Jedynym prawdziwym przykładem jest wspomniany dotyczący ochrony hosta przed powodziami SYN.

FIXME: jeśli go używasz, podziel się z nami swoim doświadczeniem

Filtry mieszające i bardzo dużo szybkiego filtrowania

Jeśli potrzebujesz tysięcy reguł, na przykład w sytuacji gdy masz bardzo dużo klientów lub komputerów, dodatkowo różne specyfikacje co do ‘jakości usług’ (ang. „Quality of Service”, QoS) może się okazać, że twój kernel spędza bardzo dużo czasu sprawdzając te reguły.

Domyślnie, wszystkie reguły zapisane są w dużym łańcuchu, który sprawdzany jest w kolejności zmniejszających się priorytetów. Jeśli masz 1000 reguł, być może konieczne będzie 1000 testów by określić co zrobić z pakietem.

Sprawdzanie odbywałoby się szybciej, gdybyś miał 256 różnych łańcuchów i w każdym cztery reguły - jeśli podzieliłbyś pakiety pomiędzy te 256 łańcuchów, tak by reguły odpowiadały pakietom.

Tutaj przychodzi na pomoc mieszanie. Powiedzmy, że masz 1024 użytkowników w swojej sieci połączonych modemami kablowymi, z przydzielonymi adresami od 1.2.0.0 do 1.2.3.255. Każdy z nich należy do innej kategorii, np. ‘podstawowy’, ‘standardowy’ i ‘wyjątkowy’. Potrzebujesz 1024 reguł, takich jak te:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.1 classid 1:1
...
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.254 classid 1:3
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.255 classid 1:2
```

By przyśpieszyć przetwarzanie, możemy użyć ostatniej części adresu IP jako ‘klucza mieszającego’. Dostajemy 256 tabel, z których pierwsze wyglądają tak:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.1.0 classid 1:1
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.2.0 classid 1:3
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.3.0 classid 1:2
```

Następna zaś zaczyna się tak:

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 100 match ip src \
  1.2.0.1 classid 1:1
...
```

W ten sposób mamy tylko cztery testy w najgorszym wypadku a średnio dwa.

Konfiguracja jest trochę skomplikowana, ale warta czasu który zyskasz. Najpierw tworzymy filtr-korzeń a następnie tabelę z 256 wpisami:

```
# tc filter add dev eth1 parent 1:0 prio 5 protocol ip u32
# tc filter add dev eth1 parent 1:0 prio 5 handle 2: protocol ip u32 divisor 256
```

Teraz dodajemy trochę reguł do wpisów w tabeli:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.0.123 flowid 1:1
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.1.123 flowid 1:2
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.3.123 flowid 1:3
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 2:7b: \
        match ip src 1.2.4.123 flowid 1:2
```

Mamy tu wpisy dla 123, które zawiera testy dla `1.2.0.123`, `1.2.1.123`, `1.2.2.123` oraz dla `1.2.3.123`. Wysyła je do kolejek odpowiednio 1:1, 1:2, 1:3 i 1:2. Zauważ, że wartość mieszającą podajemy w wartościach heksdecymalnych, tutaj `0x7b` to właśnie `123`.

Teraz tworzymy ‘filtr mieszający’, który kieruje ruch do właściwego wpisu w tabeli:

```
# tc filter add dev eth1 protocol ip parent 1:0 prio 5 u32 ht 800:: \
        match ip src 1.2.0.0/16 \
        hashkey mask 0x000000ff at 12 \
        link 2:
```

Parę wartości może wymagać wyjaśnienia. Domyślna tablica mieszająca nazywa się 800:: i całe filtrowanie zaczyna się od niej. Wybieramy adres źródłowy, który znajduje się na pozycji 12, 13, 14 i 15 w nagłówku IP i wskazujemy, że jesteśmy zainteresowani tylko tą częścią ostatnią. Następnie wysyłamy go do tablicy mieszającej 2:, którą stworzyliśmy poprzednio

Trochę to skomplikowane, ale działa w praktyce a zysk wydajności będzie naprawdę uderzający. Zauważ, że ten przykład może być jeszcze polepszony, w idealnym przypadku każdy łańcuch zawierałby 1 filtr!.

Filtrowanie ruchu IPv6

Dlaczego filtry tc dla IPv6 nie działają?

Baza danych zasad routingu (ang. „Routing Policy Database”, RPDB) zastąpiła strukturę routingu i adresowania dla IPv4 w jądrze Linuksa, która odpowiadała za całą wspaniałą funkcjonalność opisywaną w tym HOWTO. Niestety, struktura odpowiedzialna za IPv6 została zaimplementowana poza tą bazą. O ile obie współdzielą część mechanizmów, RPDB nie współpracuje z procesem adresowania ani routingu IPv6.

To się na pewno zmieni, musimy tylko trochę cierpliwie poczekać.

FIXME: Czy ktoś nad tym pracuje? Albo przynajmniej ma plany?

Oznaczanie pakietów IPv6 przez ip6tables

ip6tables umożliwia oznaczanie pakietów i przydzielanie im numerów:

```
# ip6tables -A PREROUTING -i eth0 -t mangle -p tcp -j MARK --mark 1
```

Nadal jednak nic to nie zmienia - pakiet nie przejdzie przez wspomnianą wcześniej strukturę RPDB.

Zastosowanie selektora u32 do testu pakietu IPv6

Normalnie, IPv6 jest hermetyzowany w tunel SIT i transportowany ponad siecią IPv4. Zajrzyj do sekcji ‘Tunelowanie IPv6’ po więcej informacji jak zestawić taki tunel. Takie podejście do przesyłania pakietów IPv6 umożliwia nam ich filtrowanie na poziomie protokołu IPv4.

Poniższy filtr pasuje do całegu ruchu IPv6 zahermetyzowanego w pakietach IPv4:

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff flowid 42:42
```

Przyjrzymy się temu. Załóżmy, że pakiet IPv6 został wysłany w IPv4 i oba nie mają ustawionych żadnych opcji. Możemy użyć poniższego filtra by wybrać pakiety ICMPv6 w IPv6 zahermetyzowane w IPv4. Wartość 0x3a (58) to pole typu ‘next-header’ dla ICMPv6.

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
           match ip protocol 41 0xff \
           match u8 0x05 0x0f at 0 \
           match u8 0x3a 0xff at 26 \
           flowid 42:42
```

Dopasowanie adresu przeznaczenia IPv6 wymaga trochę większej ilości pracy. Poniższy filtr pasuje do następującego adresu docelowego: `3ffe:202c:ffff:32:230:4fff:fe08:358d:`

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff \
            match u8 0x05 0x0f at 0 \
            match u8 0x3f 0xff at 44 \
            match u8 0xfe 0xff at 45 \
            match u8 0x20 0xff at 46 \
            match u8 0x2c 0xff at 47 \
            match u8 0xff 0xff at 48 \
            match u8 0xff 0xff at 49 \
```

```
            match u8 0x00 0xff at 50 \
            match u8 0x32 0xff at 51 \
            match u8 0x02 0xff at 52 \
            match u8 0x30 0xff at 53 \
            match u8 0x4f 0xff at 54 \
            match u8 0xff 0xff at 55 \
            match u8 0xfe 0xff at 56 \
            match u8 0x08 0xff at 57 \
            match u8 0x35 0xff at 58 \
            match u8 0x8d 0xff at 59 \
            flowid 10:13
```

Tą samą technikę można zastosować do testów podsieci. Na przykład dla `2001::`.

```
# tc filter add dev $DEV parent 10:0 protocol ip prio 10 u32 \
            match ip protocol 41 0xff \
            match u8 0x05 0x0f at 0 \
            match u8 0x20 0xff at 28 \
            match u8 0x01 0xff at 29 \
            flowid 10:13
```

Rozdział 13. Parametry sieciowe kernela

Kernel ma wiele parametrów, które mogą być dopasowane stosowanie do potrzeb. Ponieważ domyślne wartości w 99% instalacji sprawdzają się dobrze, nie nazywamy tego HOWTO Zaawansowanym dla samej radości!

W podkatalogu `/proc/sys/net` znajduje się wiele interesujących rzeczy i warto tam zajrzeć. Nie wszystko zostanie udokumentowane od razu, ale pracujemy nad tym.

W międzyczasie, możesz rzucić okiem na źródła jądra i zapoznać się z plikiem `Documentation/filesystems/proc.txt`. Większość opcji opisano właśnie tam.

(FIXME)

Filtrowanie trasy powrotnej

Domyślnie, routery routują wszystko, nawet pakiety w oczywisty sposób nie należące do twojej sieci. Przykładem może być prywatna przestrzeń adresowa IP wyciekająca do Internetu. Jeśli masz interfejs z ustawioną trasą `195.96.96.0/24`, nie spodziewasz się na nim raczej pakietów z adresami `212.64.94.1`.

Wielu ludzi chciałoby taką opcję wyłączyć, więc hakerzy kernela sprawili by było to proste. W katalogu `/proc` znajdują się pliki, w których możesz poinstruować kernel by zrobił to dla ciebie. Metoda nazywa się właśnie ‘Filtrowaniem trasy powrotnej’ (ang. „Reverse Path Filtering”). W skrócie, jeśli pakiety odpowiedzi nie docierają interfejsem którym wyszło zapytanie, uznajemy pakiet za zagadkowy i powinniśmy go zignorować.

Poniższy fragment włącza to filtrowanie dla wszystkich obecnych i przyszłych interfejsów:

```
# for i in /proc/sys/net/ipv4/conf/*/rp_filter ; do
>  echo 2 > $i
> done
```

Kontynuując przykład z powyżej, jeśli pakiet dotarłbym do rutera Linuksowego interfejsem eth1 twierdząc, że pochodzi z podsieci ISP+Biurowej, zostałby odrzucony. Podobnie, pakiet z podsieci Biura twierdący że pochodzi zza ściany ogniowej również zostałby odrzucony.

Powyżej opisano pełen filtr trasy powrotnej. Domyślnie, filtruje się tylko na podstawie adresów IP i bezpośrednio podłączonych sieci. Dzieje się tak dlatego, że filtrowanie nie działa dobrze przy routingu asymetrycznym (w którym pakiety przychodzą innym interfejsem a wychodzą innym, jak dzieje się na przykład w połączeniach satelitarnych lub używających routingu dynamicznego - BGP, OSPF lub RIP). Dane przychodzą łączem satelitarnym a odpowiedzi wychodzą jakimś rodzajem połączenia naziemnego.

Jeśli taka sytuacja ma miejsce u ciebie (i prawdopodobnie wiesz o tym, jeśli tak jest) możesz wyłączyć `rp_filter` na interfejsie, którym odbierasz dane z łącza satelitarnego. Jeśli chcesz sprawdzić czy jakieś pakiety są odrzucane, w tym samym katalogu znajduje się plik `log_martians`, którym możesz spowodować logowanie takich pakietów za pomocą sysloga.

```
# echo 1 >/proc/sys/net/ipv4/conf/<interfacename>/log_martians
```

FIXME: czy ustawienie plików `conf/{default,all}/*` wystarcza? - martijn

Mało znane ustawienia

Jest bardzo dużo parametrów, które można modyfikować. Próbujemy je tu wymienić wszystkie. Po części, są również udokumentowane w pliku `Documentation/ip-sysctl.txt`.

Niektóre z tych ustawień mają różne wartości domyślne, w zależności od tego jak odpowiedziałeś na pytanie ‘Configure as router and not host’ podczas konfiguracji kernela.

Oskar Andreasson stworzył swoją stronę o tych ustawieniach i wydaje się ona lepsza niż nasz dokument, sprawdź więc http://ipsysctl-tutorial.frozentux.net/.

Podstawowe IPv4

Jako ogólna uwaga warto zaznaczyć, że większość właściwości ograniczania/kształtowania ruchu nie działa na interfejsie loopback, więc testowanie ich na nim nie ma sensu. Limity podaje się w jednostkach zwanych ‘jiffies’ a wymusza się je za pomocą wspomnianego wcześniej TBF’a.

Kernel ma wewnętrzny zegar pracujący w jednostkach ’HZ’ (lub ‘jiffies’) na sekundę. Na platformie intelowskiej, ‘HZ’ to prawie 100, więc ustawienie pliku `*_rate` na powiedzmy 50 spowoduje przesłanie tylko 2 pakietów na sekundę. TBF jest skonfigurowany na przepuszczanie maksymalnie serii po 6 pakietów, jeśli dostępna jest odpowiednia ilość żetonów.

Niektóre wpisy w poniższej liście skopiowano z `/usr/src/linux/Documentation/networking/ip-sysctl.txt`, autorstwa Aleksieja Kuzniecowa <kuznet@ms2.inr.ac.ru> i Andi Kleen <ak@muc.de>.

/proc/sys/net/ipv4/icmp_destunreach_rate

> Jeśli kernel zdecyduje, że nie może dostarczyć pakietu odrzuci go i wyśle źródle pakietu informacje w postaci ICMP by je o tym powiadomić.

/proc/sys/net/ipv4/icmp_echo_ignore_all

> Nie reaguj w ogóle na pakiety ICMP echo. Nie ustawiaj tego domyślnie, chyba że jesteś używany jako przekaźnik w atakach DoS.

/proc/sys/net/ipv4/icmp_echo_ignore_broadcasts [użyteczne]

> Jeśli pingujesz adres rozgłoszeniowy spodziewasz się odpowiedzi od wszystkich komputerów. Jest to wygodne narzędzie DoS, ustawienie w tym pliku wartości ‘1’ powoduje, że rozgłaszane wiadomości ICMP echo są ignorowane.

/proc/sys/net/ipv4/icmp_echoreply_rate

> Częstotliwość z jaką kernel wysyła odpowiedzi na ping pod określony adres przeznaczenia.

/proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

> Ustaw by ignorować błędy ICMP spowodowane przez komputery źle reagujące na ramki, które uważają za adresy rozgłoszeniowe.

/proc/sys/net/ipv4/icmp_paramprob_rate

> Mało znana wiadomość ICMP wysyłana w odpowiedzi na pakiet z uszkodzonym nagłówkiem IP lub TCP.

/proc/sys/net/ipv4/icmp_timeexceed_rate

> Znany powód występowania ‘solarisowej gwiazdki w środku’ w przypadku wykonywania traceroute. Ogranicza liczbę wiadomości ICMP Przekroczono Czas.

/proc/sys/net/ipv4/igmp_max_memberships

Maksymalna liczba nasłuchujących gniazd IGMP na maszynie. FIXME: Czy faktycznie?

/proc/sys/net/ipv4/inet_peer_gc_maxtime

FIXME: Dodać trochę wyjaśnień do mechanizmu przechowywania informacji?

Minimalny interwał pomiędzy ‘zbieraniem śmieci’. Interwał mierzony jest tylko gdy wolna jest mała część puli pamięci.

/proc/sys/net/ipv4/inet_peer_gc_mintime

Minimalny interwał pomiędzy ‘zbieraniem śmieci’. Ten interwał mierzony jest tylko gdy wolna jest duża część puli pamięci. Mierzony w ‘jiffies’.

/proc/sys/net/ipv4/inet_peer_maxttl

Maksymalny czas życia wpisów. Nieużywane wpisy wygasają po upływie określonego czasu lub gdy nie ma pamięci w puli (tzn. gdy liczba wpisów w puli jest już bardzo mała). Mierzony w ‘jiffies’.

/proc/sys/net/ipv4/inet_peer_minttl

Minimalny czas życia wpisów. Powinien być na tyle duży, by obejmować czas potrzebny na składanie fragmentów. Czas ten jest gwarantowany, jeśli wielkość puli jest mniejsza niż wartość ‘inet_peer_threshold’. Mierzony w ‘jiffies’.

/proc/sys/net/ipv4/inet_peer_threshold

Przybliżona wielkość archiwum INET. Poczynając od tego progu, wpisy będą wyrzucane agresywniej. Próg określa również czasy życia wpisów dla interwałów zbierania śmieci. Czym więcej wpisów, mniejszy czas TTL i interwał zbierania śmieci.

/proc/sys/net/ipv4/ip_autoconfig

Plik zawiera wartość ‘1’ jeśli konfiguracja IP odbywa się za pomocą protokołu RARP, BOOTP, DHCP lub podobnego. W innym wypadku zawiera zero.

/proc/sys/net/ipv4/ip_default_ttl

Czas życia (TTL) pakietów. Ustawiony na bezpieczną wartość ‘64’. Podnieś, jeśli masz dużą sieć i raczej nie baw się tym parameterm - pętle routingu potrafią wyrządzić wiele kłopotu. Czasami nawet warto obniżyć tą wartość.

/proc/sys/net/ipv4/ip_dynaddr

Musisz ustawić tu ‘1’ jeśli korzystasz z połączeń zestawianych na żądanie i masz przydzielany dynamiczny adres interfejsu. Po tym, jak podniesiony zostanie interfejs, gniazda TCP nie otrzymają odpowiedzi dopóki nie skojarzone zostaną z odpowiednim adresem. Rozwiązuje to problem w przypadku, gdy pierwsze nawiązanie połączenia nie podnosi interfejsu a np. drugie pomaga.

/proc/sys/net/ipv4/ip_forward

Czy kernel ma przekazywać pakiety. Domyślnie wyłączone.

/proc/sys/net/ipv4/ip_local_port_range

Zakres portów lokalnych przeznaczonych dla połączeń wychodzących. Domyślnie całkiem mały, od 1024 do 4999.

/proc/sys/net/ipv4/ip_no_pmtu_disc

Ustaw ‘1’ w tym pliku jeśli chcesz wyłączyć rozpoznawanie MTU ścieżki (ang. „Path MTU Discovery”, PMTUD) - techniki pozwalającej określić ‘Maksymalną Jednostkę Transmisji’ (ang. „Maximum Transfer Unit”) możliwą na twojej trasie. Zajrzyj również do tego rozdziału *Cookbook*.

/proc/sys/net/ipv4/ipfrag_high_thresh

Maksymalna ilość pamięci przeznaczona na fragmenty IP. Jeśli zaalokowano już ‘ipfrag_high_thresh’ bajtów do tego celu, funkcja obsługująca defragmentację będzie odrzucać nowe fragmenty aż osiągnięty zostanie poziom wartości ‘ipfrag_low_thresh’.

/proc/sys/net/ipv4/ip_nonlocal_bind

Ustaw w tym pliku ‘1’ jeśli chcesz, by aplikacje mogły bindować się do adresów nie istniejących na żadnym urządzeniu w twoim systemie. Może to być użyteczne, jeśli komputer jest podłączony do dynamicznego łącza, więc dobrze byłoby gdyby usługi mogły startować i bindować się do określonych adresów gdy łącze jest nieczynne.

/proc/sys/net/ipv4/ipfrag_low_thresh

Minimalna ilość pamięci używana do składania fragmentów IP.

/proc/sys/net/ipv4/ipfrag_time

Czas w sekundach, przez który fragmenty IP są przetrzymywane w pamięci.

/proc/sys/net/ipv4/tcp_abort_on_overflow

Wartość logiczna kontrolująca zachowanie w przypadku dużej ilości połączeń przychodzących. Jeśli ustawiona na ‘1’ powoduje, że kernel aktywnie wysyła pakiety RST gdy usługa jest przeładowana.

/proc/sys/net/ipv4/tcp_fin_timeout

Czas przez który utrzymywać gniazdo w stanie FIN-WAIT-2 jeśli zostało zamknięte przez drugą stronę. Mogło dojść do awarii lub w ogóle zniknięcia z sieci i nigdy nie doczekamy się końca sesji. Domyślnie wynosi 60 sekund. W kernelach 2.2 używano wartości 180 sekund, więc możesz chcieć tak ustawić swoje jądro, ale pamiętaj, że jeśli twój komputer jest np. przeładowanym serwerem WWW ryzykujesz przeładowanie pamięci kilotonami martwych gniazd. Co prawda gniazda w stanie FIN-WAIT-2 są mniej groźne niż w stanie FIN-WAIT-1 ponieważ zabieraja tylko po 1.5kB pamięci, ale generalnie żyją zwykle dłużej. Zajrzyj również do opisu ‘tcp_max_orphans’.

/proc/sys/net/ipv4/tcp_keepalive_time

Jak często TCP wysyła wiadomości ‘keepalive’, jeśli włączono tą funkcję - domyślnie co 2 godziny.

/proc/sys/net/ipv4/tcp_keepalive_intvl

Jak często retransmitowane są wiadomości ‘keepalive’, jeśli nie zostaną potwierdzone - domyślnie co 75 sekund.

/proc/sys/net/ipv4/tcp_keepalive_probes

Jak wiele wiadomości ‘keepalive’ zostanie wysłanych, zanim kernel zdecyduje, że połączenie zostało przerwane - domyślnie 9. Pomnożone przez wartość ‘tcp_keepalive_intvl’ pozwala obliczyć przez jak długi czas połaczenie może nie przekazywać pakietów po wysłaniu pierwszej wiadomości ‘keepalive’.

/proc/sys/net/ipv4/tcp_max_orphans

Maksymalna ilość gniazd TCP utrzymywanych przez system, nie dołączonych do żadnego uchwytu pliku użytkownika. Jeśli wartość ta zostanie przekroczona, osierocone połączenia są resetowane i wysyłane jest

ostrzeżenie. Limit istnieje tylko po to, by zapobiec prostym atakom DoS, nie możesz polegać tylko na nim, lub zbyt go obniżyć, a raczej powinieneś nawet go zwiększyć (być może po zwiększeniu pamięci), jeśli warunki sieciowe wymagają więcej niż domyślna wartość. To usługi powinny zajmować się śledzeniem i eliminowaniem takich stanów bardziej agresywniej. Pozwól sobie przypomnieć ważną informację: każde osierocone gniazdo to stracone około 64K pamięci, której nie można wyrzucić do pliku wymiany.

/proc/sys/net/ipv4/tcp_orphan_retries

Jak wiele razy próbować ponowić próbę, zanim połączenie TCP zostanie zabite po naszej stronie. Domyślna wartość '7' odpowiada około ~50 sekundom - 16 minutom w zależności od RTO. Jeśli masz przeładowaną maszynę powinieneś rozważyć obniżenie tej wartości, ponieważ gniazda mogą zająć wiele zasobów.

/proc/sys/net/ipv4/tcp_max_syn_backlog

Maksymalna ilość pamiętanych żądań połączeń, które nadal nie zostały potwierdzone przez klienta. Domyślna wartość 1024 jest właściwa dla systemów z pamięcią powyżej 128MB, a 128 jest dobre dla komputerów z mniejszą ilością pamięci. Jeśli masz przeładowany serwer spróbuj zwiększyć tą wartość. Uwaga! Jeśli ustawisz ją na więcej niż 1024, lepiej będzie zmienić również TCP_SYNQ_HSIZE w pliku `include/net/tcp.h` by utrzymać zależność TCP_SYNQ_HSIZE*16<=tcp_max_syn_backlog i przekompilować kernel.

/proc/sys/net/ipv4/tcp_max_tw_buckets

Maksymalna ilość gniazd w stanie TIMEWAIT utrzymywana w danym momencie przez system. Jeśli ta wartość zostanie przekroczona, gniazdo jest niszczone i generowane jest ostrzeżenie. Limit istnieje tylko po to, by zapobiec prostym atakom DoS, nie możesz polegać tylko na nim, lub zbyt go obniżyć (być może po zwiększeniu pamięci), jeśli warunki sieciowe wymagają więcej niż domyślna wartość.

/proc/sys/net/ipv4/tcp_retrans_collapse

Włączenie błędu, który może być konieczny do obsłużenia innego błędu w rzypadku pracy z niektórymi uszkodzonymi drukarkami. W czasie retransmisji pakietów powoduje wysyłanie większych pakietów by niektóre stosy TCP prawidłowo je rozpoznały.

/proc/sys/net/ipv4/tcp_retries1

Jak wiele razy próbować, zanim stwierdzimy że coś jest nie tak i należy to zgłosić do warstwy sieciowej. Minimalną wartością zalecaną w RFC jest '3' i jest to wartość domyślna. Odpowiada ~3 sekundom - 8 minutom w zależności od RTO.

/proc/sys/net/ipv4/tcp_retries2

Jak wiele razy próbować przed zabiciem żywej sesji TCP. RFC1122 (http://www.ietf.org/rfc/rfc1122.txt) mówi, że limit ten powinien wynosić więcej niż 100 sekund, ale jest to za mało. Domyślną wartością jest 15, co odpowiada ~13-30 minutom w zależności od RTO.

/proc/sys/net/ipv4/tcp_rfc1337

Ta wartość logiczna włącza poprawki opisane w RFC 1337 związane z niebezpieczeństwami ginięcia gniazd w stanie time-wait. Jeśli zostanie ustawiona, kernel odrzuca pakiety RST dla gniazd w stanie time-wait. Domyślnie wyłączone ('0').

/proc/sys/net/ipv4/tcp_sack

Użyj selektywnego ACK, co może pomóc stwierdzić, że brakuje pewnych pakietów - a w związku z tym wspomóc proces odzyskiwania zasobów.

/proc/sys/net/ipv4/tcp_stdurg

Użyj interpretacji wymagań w stosunku do hosta dla wskaźnika pilnych danych. Większość komputerów używa starej implementacji BSD, więc jeśli ją włączysz, Linux może nie móc się z nimi komunikować. Domyślnie wyłączone (‘0’).

/proc/sys/net/ipv4/tcp_syn_retries

Ilość pakietów SYN, które wyśle kernel zanim się podda podczas kreowania nowego połączenia.

/proc/sys/net/ipv4/tcp_synack_retries

By otworzyć zdalne połączenie, kernel wysyła pakiet z ustawioną flagą SYN i ACK wskazującą na poprzedni, tak by go potwierdzić. Jest to część druga trzy-stopniowego przywitania. Ten parametr kontroluje ile pakietów zostanie wysłanych, zanim kernel podda się i odrzuci połączenie.

/proc/sys/net/ipv4/tcp_timestamps

Znaczniki czasu używane są, między innymi, do ochrony przed ‘zawijaniem’ się numerów sekwencyjnych. Na łączu 1 gigabitowym połączenia mogłyby łatwo zacząć używać już raz użytych numerów sekwencyjnych, w związku z tym używa się dodatkowo znaczników czasu.

/proc/sys/net/ipv4/tcp_tw_recycle

Włącz szybkie odzyskiwane gniazd TIME-WAIT. Domyślnie ustawione. Nie powinno być zmieniane bez poinstruowania wprost przez ekspertów technicznych.

/proc/sys/net/ipv4/tcp_window_scaling

TCP/IP umożliwia normalnie obsługę okien do rozmiarów 65535 bajtów. Dla bardzo szybkich sieci może to nie być wystarczająca wartość. Opcje skalowania okien pozwalają rozszerzyć je aż do łączy gigabitowych, co jest pożyteczne dla połączeń z dużymi przepustowościami.

Ustawienia dotyczące urządzeń

‘DEV’ oznacza albo prawdziwy interfejs, lub ‘all’ (wszystkie) lub ‘default’ (domyślny). Ostatnie znaczenie zmienia również ustawienia dla interfejsów, które będą stworzone w przyszłości.

/proc/sys/net/ipv4/conf/DEV/accept_redirects

Jeśli router zdecyduje, że używasz go do złych celów (np. to przesyłania dalej twoich pakietów na tym samym interfejsie), wyśle pakiet ICMP Redirect (przekierowanie). Jest to trochę niebezpieczne, więc być może chciałbyś wyłączyć tą opcję, lub używać bezpiecznych przekierowań.

/proc/sys/net/ipv4/conf/DEV/accept_source_route

Rzadko używane. Można było dać pakietowi listę adresów IP, które powinien odwiedzić po drodze. Można poinstruować Linuksa, by honorował tą opcję IP.

/proc/sys/net/ipv4/conf/DEV/bootp_relay

Akceptuj pakiety z adresem źródłowym 0.b.c.d i adresem docelowym nie kierowanym do tego hosta tak, jakby były to pakiety lokalne. Zakłada się, że demon BOOTP przechwyci je i przekaże dalej.

Domyślną wartością jest 0, ponieważ ta funkcjonalność nie jest jeszcze zaimplementowana (kernel w wersji 2.2.12).

/proc/sys/net/ipv4/conf/DEV/forwarding

Włącz lub wyłącz przekazywanie pakietów IPv4 przez ten interfejs.

/proc/sys/net/ipv4/conf/DEV/log_martians

Zajrzyj do sekcji *Reverse Path Filtering*.

/proc/sys/net/ipv4/conf/DEV/mc_forwarding

Jeśli przekazujemy multicasty na danym interfejsie.

/proc/sys/net/ipv4/conf/DEV/proxy_arp

Jeśli ustawisz na ‘1’, wszystkie interfejsy będą odpowiadały na zapytania ARP dla adresów znajdujących się na tym interfejsie. Może być bardzo użyteczne przy budowaniu ‘pseudo-mostów ip’. Upewnij się, że maski sieciowe są poprawne zanim to włączysz! Weź również pod uwagę, że wspomniany poniżej ‘rp_filter’ również działa na zapytaniach ARP!

/proc/sys/net/ipv4/conf/DEV/rp_filter

Zajrzyj do sekcji *Reverse Path Filtering*.

/proc/sys/net/ipv4/conf/DEV/secure_redirects

Akceptuj przekierowania ICMP tylko dla bramek wymienionych w liście domyślnych bramek. Domyślnie włączone.

/proc/sys/net/ipv4/conf/DEV/send_redirects

Jeśli wysyłamy wymienione wcześniej przekierowania.

/proc/sys/net/ipv4/conf/DEV/shared_media

Jeśli nie jest ustawiona, kernel nie zakłada że różne podsieci na tym urządzeniu mogą komunikować się bezpośrednio. Domyślnie ustawione na ‘tak’.

/proc/sys/net/ipv4/conf/DEV/tag

FIXME: wypełnić to

Polityka dotycząca sąsiadów

‘DEV’ oznacza albo prawdziwy interfejs, lub ‘all’ (wszystkie) lub ‘default’ (domyślny). Ostatnie znaczenie zmienia również ustawienia dla interfejsów, które będą stworzone w przyszłości.

/proc/sys/net/ipv4/neigh/DEV/anycast_delay

Maksimum dla losowych opóźnień wysyłania odpowiedzi na komunikaty sąsiadów, wyrażane w ‘jiffies’. Nie zaimplementowana (Linuks nie wspiera obecnie anycastów).

/proc/sys/net/ipv4/neigh/DEV/app_solicit

Określa ilość żądań do wysłania do demona ARP w przestrzeni użytkownika. Ustaw na 0 by wyłączyć.

/proc/sys/net/ipv4/neigh/DEV/base_reachable_time

Podstawowa wartość używana do wyliczania losowych czasów osiągalności tak jak określono to w RFC 2461 (http://www.faqs.org/rfcs/rfc2461.html).

/proc/sys/net/ipv4/neigh/DEV/delay_first_probe_time

Opóźnienie dla pierwszej próby sprawdzenia, czy sąsiad jest osiągalny (patrz gc_stale_time).

/proc/sys/net/ipv4/neigh/DEV/gc_stale_time

Określa jak często sprawdzać aktualność wpisów ARP. Po wygaśnięciu wpisu ARP zostanie podjęta próba rozwiązania go ponownie - co przydaje się w sytuacji gdy adres przejęła inna maszyna. Gdy ‘ucast_solicit’ jest większe od 0, kernel wyśle najpierw pakiet ARP bezpośrednio do ostatniego znanego hosta. Jeśli się to nie powiedzie, a ‘mcast_solicit’ jest również ustawione powyżej 0, wysyłane jest rozgłoszenie ARP.

/proc/sys/net/ipv4/neigh/DEV/locktime

Stary wpis o ARP sąsiada jest zastępowany nowym tylko wtedy, gdy stary ma przynajmniej równowartość ‘locktime’. Zapobiega to zaśmiecaniu pamięci podręcznej ARP.

/proc/sys/net/ipv4/neigh/DEV/mcast_solicit

Maksymalna ilość prób rozgłoszeń multicast.

/proc/sys/net/ipv4/neigh/DEV/proxy_delay

Maksymalny czas (czas rzeczywisty jest losową z zakresu [0..proxytime]) zanim odpowiemy na zapytanie ARP dla którego posiadamy wpis w proxy ARP. W niektórych przypadkach pomaga to zalewaniu sieci potokiem ramek.

/proc/sys/net/ipv4/neigh/DEV/proxy_qlen

Maksymalna długość kolejki licznika proxy-arp działającego z opóźnieniem (patrz również proxy_delay).

/proc/sys/net/ipv4/neigh/DEV/retrans_time

Czas, wyrażony w ‘jiffies’ pomiędzy transmisją wiadomości do sąsiadów. Używane do rozwiązywania adresów i określenia, czy sąsiad jest faktycznie osiągalny.

/proc/sys/net/ipv4/neigh/DEV/ucast_solicit

Maksymalna ilość rozgłoszeń unicastowych o sąsiedztwo.

/proc/sys/net/ipv4/neigh/DEV/unres_qlen

Maksymalna długość kolejki dla zaległych wywołań ARP - ilość pakietów akceptowana z innych warstw, w czasie gdy adres jest nadal rozwiązywany.

Internet QoS: Architectures and Mechanisms for Quality of Service, Zheng Wang, ISBN 1-55860-608-4

Książka opisująca zagadnienia związane z Jakością Usług w sieciach. Bardzo dobra do zrozumienia podstawowych koncepcji.

Ustawienia dla routingu

/proc/sys/net/ipv4/route/error_burst

Parametr kontrolujący i ograniczający ilość wiadomości z ostrzeżeniami zapisywanymi do logu kernela z kodu routującego. Im wyższa wartość ‘error_cost’ tym mniej wiadomości zostaje zapisanych. ‘error_burst’ kontroluje kiedy wiadomości będą odrzucane. Domyślne ustawienia ograniczają generowanie wiadomości do jednej na pięć sekund.

/proc/sys/net/ipv4/route/error_cost

Parametr kontrolujący i ograniczający ilość wiadomości z ostrzeżeniami zapisywanymi do logu kernela z kodu routującego. Im wyższa wartość ‘error_cost’ tym mniej wiadomości zostaje zapisanych. ‘error_burst’ kontroluje kiedy wiadomości będą odrzucane. Domyślne ustawienia ograniczają generowanie wiadomości do jednej na pięć sekund.

/proc/sys/net/ipv4/route/flush

Zapis do tego pliku powoduje wyczyszczenie pamięci podręcznej routingu.

/proc/sys/net/ipv4/route/gc_elasticity

Wartości kontrolujące częstotliwość i zachowanie algorytmu zbierającego śmieci w kodzie routingu. Może to być istotne w sytuacji gdy pracujesz w konfiguracji redundantnej. Przynajmniej ‘gc_timeout’ sekund upłynie, zanim Linuks przejdzie do kolejnej dostępnej trasy z uwagi na niedostępność poprzedniej. Domyślnie ustawione na 300 - być może będziesz chciał zmniejszyć tą wartość by przyśpieszyć przełączenie tras.

Zobacz również ten post (http://mailman.ds9a.nl/pipermail/lartc/2002q1/002667.html) autorstwa Ard van Breemen.

/proc/sys/net/ipv4/route/gc_interval

Zobacz `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_min_interval

Zobacz `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_thresh

Zobacz `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/gc_timeout

Zobacz `/proc/sys/net/ipv4/route/gc_elasticity`.

/proc/sys/net/ipv4/route/max_delay

Opóźnienie w opróżnianiu pamięci podręcznej routingu.

/proc/sys/net/ipv4/route/max_size

Maksymalny rozmiar pamięci podręcznej routingu. Stare wpisy będą usuwane jak tylko pamięć podręczna osiągnie rozmiar wskazany tą zmienną.

/proc/sys/net/ipv4/route/min_adv_mss

FIXME: wypełnić to

/proc/sys/net/ipv4/route/min_delay

Opóźnienie w opróżnianiu pamięci podręcznej routingu.

/proc/sys/net/ipv4/route/min_pmtu

FIXME: wypełnić to

/proc/sys/net/ipv4/route/mtu_expires

FIXME: wypełnić to

/proc/sys/net/ipv4/route/redirect_load

Wartości określające czy przekierowania ICMP powinny być wysyłane do danego hosta. Przekierowania nie będą wysyłane jeśli zostanie osiągnięty określony limit.

/proc/sys/net/ipv4/route/redirect_number

Zobacz `/proc/sys/net/ipv4/route/redirect_load`.

/proc/sys/net/ipv4/route/redirect_silence

Czas wygasania dla przekierowań. Po tym okresie przekierowania zostaną wysłane jeszcze raz, nawet jeśli zostało to zablokowane z uwagi na osiągnięcie limitu.

Rozdział 14. Zaawansowane i mniej znane kolejki z dyscyplinami

Jeśli okaże się, że wymienione wcześniej kolejki nie zaspokajają twoich potrzeb, kernel zawiera jeszcze parę bardziej specjalizowanych, o których napiszemy tutaj.

`bfifo/pfifo`

Te bezklasowe kolejki są prostsze nawet niż pfifo_fast, ponieważ brakuje im wewnętrznych pasm - cały ruch jest równy. Mają jednak ważną zaletę, ponieważ dostarczają statystyk. Możesz więc użyć tej kolejki, jeśli nie potrzebujesz kształtowania ruchu czy priorytetyzowania ale logowania tego co dzieje się na interfejsie.

pfifo ma długość wyrażaną w pakietach, bfifo w bajtach.

Parametry i użycie

limit

Określa długość kolejki. Mierzona w bajtach w przypadku kolejki bfifo a w pakietach dla pfifo. Domyślnie równa wartości parametru ‘txqueuelen’ w pakietach lub iloczynowi ‘txqueuelen’ * ‘mtu’ jeśli chodzi o wartość w bajtach dla bfifo.

Algorytm Clarka-Shenkera-Zhanga (CSZ)

Jest to tak teoretyczne, że nawet Aleksiej (główny autor CBQ) nie twierdzi, że ją rozumie. Cytując go:

> David D. Clark, Scott Shenker i Lixia Zhang *Obsługa aplikacji czasu rzeczywistego w zintegrowanej sieci usług pakietowych: Architektura i Mechanizmy*.
>
> Jeśli dobrze rozumiem, główną ideą jest stworzenie potoków WFQ dla każdej gwarantowanej usługi i zaalokowanie reszty pasma sieciowego dla pasma flow-0. Każdy taki potok będzie zawierał usługi przewidywalne oraz ruch typu best-effort, a obsługiwany będzie przez planer priorytetow, z pasmem najwyższego priorytetu zarezerwowanym dla usług przewidywalnych a resztę - dla pakietów ruchu typu best-effort.
>
> Zauważcie, że potoki CSZ NIE są ograniczone do ich nominalnej przepustowości. Zakładamy, że potok przeszedł już kontrolę na granicy sieci QoS i nie potrzebuje dalszego kształtowania. Jakakolwiek próba poprawienia potoku lub dostosowania go do wiadra żetonów na węzłach pośrednich wprowadzi niepożądane zwłoki i zwiększy zakłócenia.
>
> Na razie CSZ tylko planuje, dostarczając prawdziwie gwarantowanych usług. Inne schematy (włącznie z CBQ) nie zapewniają gwarantowanych zwłok i wprowadzają losowe zaklocenia.
>
> Nie wygląda to na razie na dobrego kandydata, chyba, że przeczytałeś i zrozumiałeś powyższy artykuł.

DSMARK

Esteve Camps

<marvin%grn.es>

Ten tekst to streszczenie moich tez zebranych w *Wsparcie dla QoS w Linuksie*, wrzesień 2000.

Dokumenty źródłowe:

- Draft-almesberger-wajhak-diffserv-linux-01.txt (ftp://icaftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt.gz).
- Przykłady z dystrybucji iproute2.
- White Paper-QoS protocols and architectures (http://www.qosforum.com/white-papers/qosprot_v3.pdf) i IP QoS Frequently Asked Questions (http://www.qosforum.com/docs/faq) oba autorstwa *Quality of Service Forum*.

Ten rozdział jest autorstwa Esteve Camps <esteve%hades.udg.es>.

Wprowadzenie

Po pierwsze, dobrze byłoby przeczytać RFC poświęcone temu tematowi (RFC2474, RFC2475, RFC2597 i RFC2598) ze strony IETF DiffServ working Group (http://www.ietf.org/html.charters/diffserv-charter.html) oraz strony autorstwa Wernera Almesbergera (http://diffserv.sf.net/) (napisał kod odpowiedzialny za różnicowanie usług w Linuksie).

Z czym jest związany Dsmark?

Dsmark jest kolejką z dyscypliną, która oferuje możliwości potrzebne w ‘Usługach Zróżnicowanych’ (ang. „Differentiated Services”) (nazywanych również DiffServ lub po prostu DS). DiffServ jest jedną z dwóch architektur QoS (druga nazywa się Usługami Zintegrowanymi) bazujących na wartościach przenoszonych przez pakiet w polu DS nagłówka IP.

Jednym z pierwszych rozwiązań zaprojektowanych dla IP z myślą o QoS było pole Typ Usługi (bajt ToS) umieszczone w nagłówku IP. Zmieniając tą wartość, możemy wybrać wysoki/niski poziom przepustowości, zwłokę czy też niezawodność. Nie zapewnia to wymaganej elastyczności jeśli chodzi o nowe usługi (takie jak aplikacje czasu rzeczywistego, interaktywne i inne). Po tym pomyśle pojawiły się nowe architektury. Jedną z nich jest DiffServ, która zachowała bity ToS i przemianowane pole DS.

Wskazówki dotyczące Usług Zróżnicowanych

Usługi Zróżnicowane są zorientowane na grupy. Oznacza to, że nie wiemy nic o potokach (którymi zajmą się Usługi Zróżnicowane); wiemy o agragacjach potoków i zastosujemy w stosunku do nich różne zasady, w zależności od tego, do której agregacji należy pakiet.

Kiedy pakiet dociera do węzła brzegowego (węzła wejściowego do domeny DiffServ) można w stosunku do niego zastosować zdefiniowane zasady, poddać go kształtowaniu ruchu i/lub zaznaczyć go (oznaczanie dotyczy przydzielania wartości polu DS. Całkiem jak z krowami :-)). Jest to znacznik na podstawie którego węzły domeny DiffServ będą decydowały, który poziom QoS zastosować.

Jak pewnie wydedukujesz, Usługi Zróżnicowane zawierają domenę, w której reguły DS będą stosowane. Tak naprawdę możesz o niej myśleć w ten sposób: ‘Pakiety docierające do mojej domeny będą poddane działaniu reguł, które dyktują zasady klasyfikujące a każdy przemierzany węzeł zastosuje w stosunku do takiego pakietu poziom QoS’.

Tak naprawdę, możesz stosować własne zasady w swoich lokalnych domenach, ale pewne ‘Ustalenia Poziomu Usług’ (ang. „Service Level Agreements”) powinny zostać rozważone przy podłączaniu do innych domen DS.

Możesz mieć w tym momencie masę pytań. DiffServ to więcej niż do tej pory powiedziałem. Proszę zrozumieć, że nie potrafię streścić 3 RFC w 50 liniach :-).

Praca z Dsmark

Jak podaje bibliografia DiffServ, rozróżniamy węzły brzegowe i wewnętrzne. Są to dwa ważne punkty w ścieżce obsługiwanego ruchu. Oba typy wykonują klasyfikację gdy otrzymują pakiety. Rezultaty tej klasyfikacji mogą zostać wykorzystane w różnych miejscach procesu DS zanim pakiet zostanie wypuszczony do sieci. Dlatego architektura DiffServ udostępnia strukturę ‘sk_buff’, zawierającą nowe pole nazwane ‘skb->tc_index’, w którym przechowywana jest wartość początkowej klasyfikacji dla późniejszego wykorzystania w trakcie procesu DS.

Wartość ‘skb->tc_index’ zostaje początkowo ustawiona przez qdisc DSMARK, przez pobranie pola DS z nagłówka pakietu IP. Poza tym, klasyfikator ‘cls_tcindex’ przeczyta całość lub część pola ‘skb->tc_index’ i użyje tej wartości by wybrać klasy.

Przyjrzyjmy się najpierw komendzie qdisc DSMARK i jej parametrom:

```
... dsmark indices INDICES [ default_index DEFAULT_INDEX ] [ set_tc_index ]
```

Co oznaczają?

- *indices*: rozmiar tabeli par (maska,wartość). Maksymalna wartość to 2^n, gdzie n>=0.
- *Default_index*: domyślny wskaźnik na wpis w tabeli, jeśli klasyfikator nie dopasuje żadnego warunku.
- *Set_tc_index*: instruuje proces dsmark by pobrać pole DS i zapisać je do ‘skb->tc_index’.

Przyjrzyjmy się procesowi DSMARK.

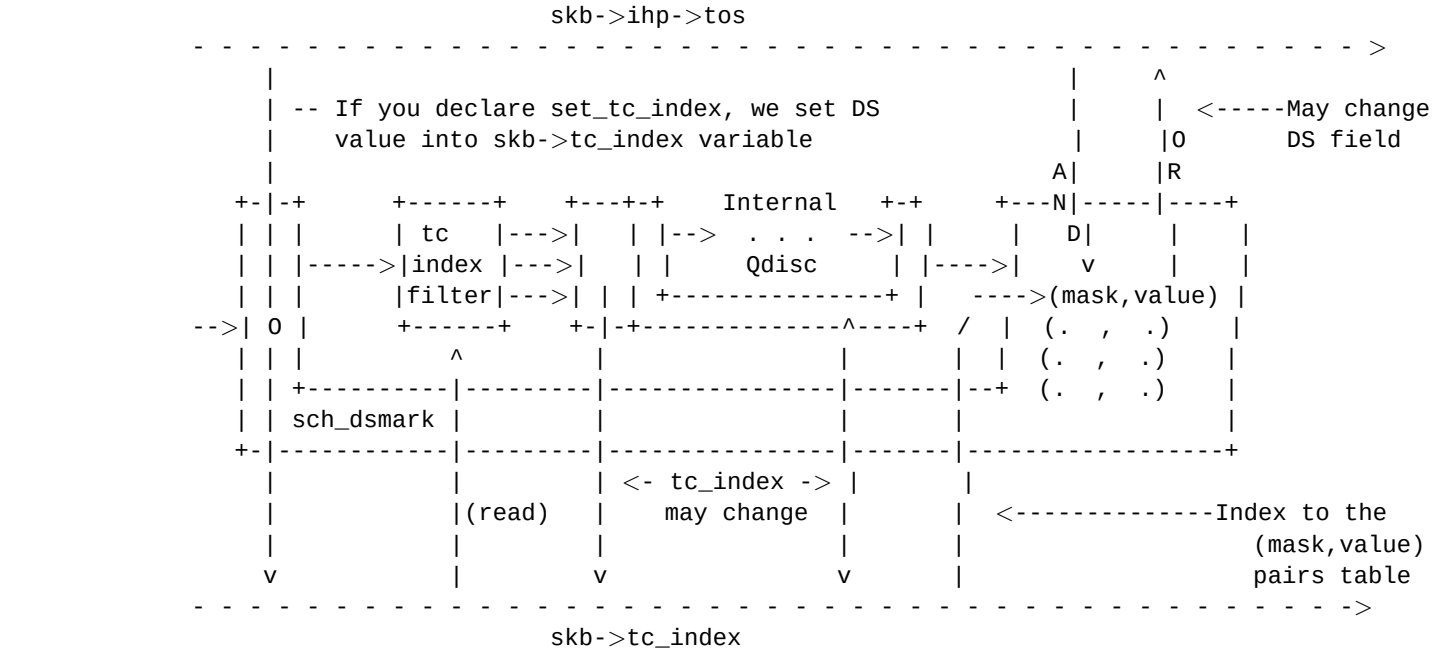
Jak działa SCH_DSMARK.

Kolejka wykonuje następujące kroki:

- Jeśli podano opcję ‘set_tc_index’ w komendzie qdisc, pole DS jest zczytywane z pakietu i przechowywane w zmiennej ‘skb->tc_index’.
- Wywoływany jest klasyfikator. Wynikiem jego działania jest identyfikator klasy, który zapisany zostanie w zmiennej ‘skb->tc_index’ (jeśli nie znajdzie się żaden filtr, który pasuje, używana jest wartość ‘default_index’ - a jeśli jej nie ma, zachowanie może być nieprzewidywalne).
- Po wysłaniu do wewnętrznych qdisc, w których możesz użyć wartości w któregoś z filtrów, identyfikator klasy zwrócony przez wewnętrzne qdisc zostaje zapisany do ‘skb->tc_index’. Wartość tej zmiennej zostanie użyta w przyszłości, by wskazać indeks w tabeli maska-wartość. Końcowa wartość przypisana pakietowi zostanie uzyskana po wykonaniu następującej operacji:

```
  Nowe_Pole_DS = ( Stare_Pole_DS & maska ) | wartość
```

- To znaczy, wartość końcowa będzie wynikiem operacji logicznej ‘i’ na starym polu DS i masce, oraz wynikiem operacji logicznej ‘lub’ z parametrem wartości. Spójrz na poniższy diagram by zrozumieć ten proces:



Jak wykonać oznaczanie pakietu? Zmień maskę i wartość klasy, którą chciałbyś komentować. Spójrz na następującą linijkę:

```
tc class change dev eth0 classid 1:1 dsmark mask 0x3 value 0xb8
```

Zmienia to parę (maska,wartość) w tablicy mieszającej powodując oznaczenie pakietów należących do klasy 1:1. Musisz ‘zmienić’ te wartości z uwagi na domyślne wartości, które para (maska,wartość) otrzymuje przy inicjalizacji (zobacz tabelę poniżej).

Teraz wytłumaczymy jak działa filtr TC_INDEX i jak wpasowuje się w to wszystko. Oprócz pracy z usługami DS, TC_INDEX może być używany w innych konfiguracjach.

Filtr TC_INDEX

Poniżej podstawowa komenda deklarująca filtr TC_INDEX:

```
... tcindex [ hash ROZMIAR ] [ mask MASKA ] [ shift PRZESUNIĘCIE ]
            [ pass_on | fall_through ]
            [ classid CLASSID ] [ police OKREŚLENIE_POLITYKI ]
```

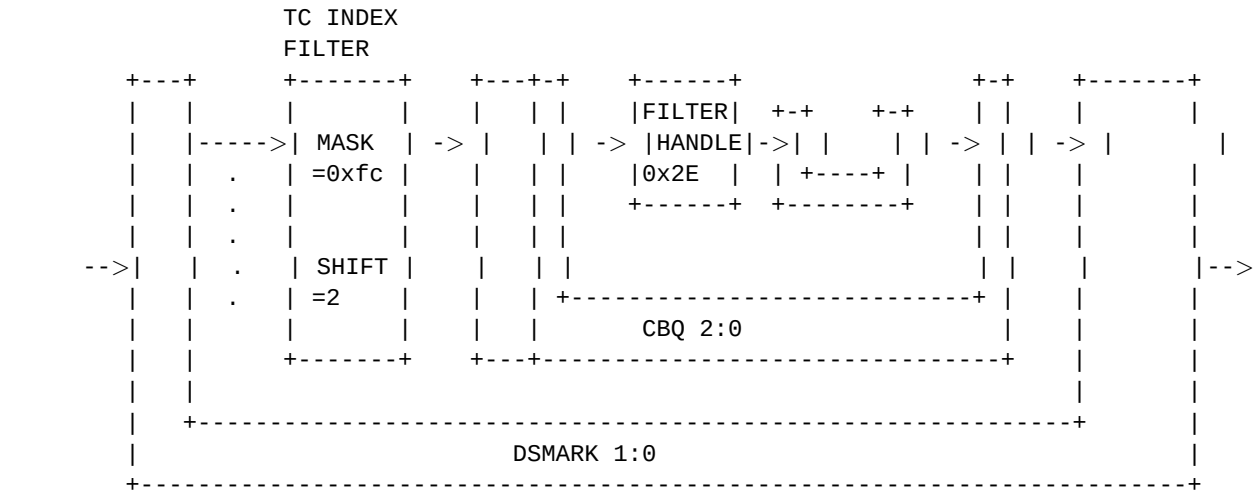
Pokażemy teraz przykład by wyjaśnić tryb pracy TC_INDEX. Zwróć uwagę na wyróżnione słowa:

```
tc qdisc add dev eth0 handle 1:0 root dsmark indices 64 \
 set_tc_index
tc filter add dev eth0 parent 1:0 protocol ip prio 1 \
 tcindex mask 0xfc shift 2
tc qdisc add dev eth0 parent 1:0 handle 2:0 cbq bandwidth 10Mbit \
```

```
 cell 8 avpkt 1000 mpu 64
# EF traffic class
tc class add dev eth0 parent 2:0 classid 2:1 cbq bandwidth 10Mbit \
 rate 1500Kbit avpkt 1000 prio 1 bounded isolated allot 1514 weight 1 maxburst 10
# Packet fifo qdisc for EF traffic
tc qdisc add dev eth0 parent 2:1 pfifo limit 5
tc filter add dev eth0 parent 2:0 protocol ip prio 1 \
 handle 0x2e tcindex classid 2:1 pass_on
```

(Powyższy kod nie jest kompletny. Pochodzi z przykładu dla EFCBQ dostarczanego z dystrybucją iproute2).

Po pierwsze, załóżmy że otrzymaliśmy pakiet oznaczony ‘EF’. Jeśli zapoznasz się z RFC2598 dowiesz się, że rekomendowaną przez DSCP wartością dla ruchu ‘EF’ jest 101110. Oznacza to, że pole DS będzie miało wartość 10111000 (pamiętaj, że mniej znaczące bity w bajcie ToS nie są używane przez DS), lub w zapisie heksdecymalnym 0xB8.



A więc dociera pakiet, z polem DS ustawionym na wartość 0xB8. Tak jak wytłumaczyliśmy wcześniej, kolejka DSMARK identyfikowana w tym przykładzie przez wartość 1:0, pobiera wartość z pola DS i zapisuje je w zmiennej ‘skb->tc_index’. Następny krok w przykładzie będzie dotyczył filtra skojarzonego z tą qdisc (druga linijka w przykładzie). Wykona to następujące operacje:

```
Wartość1 = skb->tc_index & MASKA
Klucz = Value1 >> PRZESUNIĘCIE
```

W tym przykładzie, MASKA=0xFC a PRZESUNIĘCIE=2.

```
Wartość1 = 10111000 & 11111100 = 10111000
Klucz = 10111000 >> 2 = 00101110 -> 0x2E heksdecymalnie
```

Zwrócona wartość odpowiadać będzie uchwytowi wewnętrznego filtra w qdisc (w tym przykładzie, identyfikatorowi 2:0). Jeśli istnieje filtr z taką wartością id, sprawdzone zostaną odpowiednie instrukcje dotyczące polityki oraz ograniczeń przepustowości/częstotliwości (w tym przypadku filtr je zawiera) i zwrócona zostanie wartość identyfikatora klasy (w naszym przypadku, 2:1), która z kolei zostanie zapisana do zmiennej ‘skb->tc_index’.

Jednak gdy znaleziony zostanie filtr z danym identyfikatorem, rezultat zależeć będzie od deklaracji flagi ‘fall_through’. Jeśli jest ona ustawiona, jako identyfikator klasy zwracany jest identyfikator klasy. Jeśli nie, zwracany jest błąd i proces kontynuowany jest na reszcie filtrów. Użycie flagi ‘fall_through’ wymaga jednak ostrożności; powinno odbywać się tylko gdy istnieje prosta relacja między wartościami ‘skb->tc_index’ a identyfikatorem klasy.

Ostatnie parametry, które skomentujemy to wartość mieszająca i ‘pass_on’. Pierwszy odnosi się do rozmiaru tabeli mieszającej. ‘pass_on’ jest z kolei używany do wskazania, że jeśli nie znaleziony zostanie identyfikator klasy równy zwróconemu wynikowi, należy kontynuować sprawdzanie na następnym filtrze. Domyślną akcją jest ‘fall_through’ (spójrz na następną tabelę).

Spójrzmy na koniec na wartości możliwe do ustawienia w parametrach TC_INDEX:

```
Nazwa TC                Wartość         Domyślnie
-------------------------------------------------------------
Hash                    1...0x10000     Zależne od implementacji
Maska                   0...0xffff      0xffff
Przesunięcie            0...15          0
Fall through / Pass_on  Flaga           Fall_through
Identyfikator klasy     Major:minor     żadna
Polityka                .....           żadna
```

Ten rodzaj filtra jest bardzo potężny i niezbędne jest zapoznanie się z wszystkimi jego możliwościami. Poza tym, nie jest to jedyny filtr, którego można użyć w konfiguracjach DiffServ - można go używać tak jak każdego innego rodzaju filtru.

Polecam zapoznanie się z przykładami DiffServ zawartymi w dystrybucji iproute2. Obiecuję, że uzupełnie ten tekst tak szybko jak będę mógł, poza tym wszystko co wytłumaczyłem jest rezultatem całej masy testów. Chciałbym podziękować wszystkim, którzy wytkną mi ewentualne błędy.

Przychodzące qdisc

Wszystkie omawiane do tej pory kolejki qdisc to kolejki dla ruchu wychodzącego. Każdy interfejs ma jednak również kolejki dla ruchu przychodzącego, które nie zajmują się wysyłaniem pakietów do karty sieciowej. Zamiast tego, kolejki te umożliwiają zastosowanie filtrów kontroli ruchu dla pakietów przychodzących do interfejsu, niezależnie czy adresowanych lokalnie czy przekazywanych dalej.

Ponieważ filtry **tc** zawierają pełną implementację Filtra Wiadra Żetonów (TBF) i mogą dopasowywać pakiety na podstawie estymatora przepływu kernela, mają całą masę funkcjonalności. Umożliwia to stosowanie definiowalnych zasad do nadchodzącego ruchu nawet zanim dotrze do stosu IP.

Parametry i użycie

Sama kolejka przychodząca nie wymaga żadnych parametrów. Różni się od innych kolejek qdisc tym, że nie zajmuje korzenia urządzenia. Dołącza się ją w ten sposób:

```
# tc qdisc add dev eth0 ingress
```

Umożliwia to posiadanie innej, wysyłającej kolejki qdisc na urządzeniu fizycznym oprócz dołączenia do niego kolejki przychodzącej.

By zobaczyć na dobry przykład jak można użyć kolejkę dla ruchu przychodzącego, zajrzyj do Książki Kucharskiej.

Losowe Wczesne Wykrywanie (ang. „Random Early Detection”, RED)

Ta sekcja jest rodzajem wprowadzenia do routingu w obszarze ‘szkieletu sieci’ (ang. „backbone”), który zwykle dotyczy <100 megabitowych przepustowości i wymaga trochę innego podejścia niż w przypadku operacji na modemie ADSL w domu.

Normalne zachowanie kolejek na routerach w Internecie nazywa się ‘odrzucaniem nadmiaru’ (ang. „tail-drop”). Sposób ten odbiera pakiety do pewnej wielkości a następnie odrzuca cały ruch, który ‘wylewa się’ ponad. Nie jest to oczywiście sprawiedliwe i prowadzi do synchronizacji retransmisji. Kiedy to się dzieje, nagła seria odrzuconych pakietów z routera, który osiągnął swój limit wypełnienia, spowoduje serię opóźnionych retransmisji, które z kolei znów zapchają router.

By poradzić sobie z chwilowymi stanami zagęszczenia na łączach, w routerach szkieletowych implementuje się zwykle długie kolejki. Niestety, o ile takie kolejki są dobre dla przepustowości, mogą zwiększyć opóźnienia i sprawić, że sesje TCP będą rozsyłały pakiety ‘seriami’.

Problem związany z odrzucaniem nadmiaru jest coraz bardziej poważny, w związku ze wzrostem w Internecie aplikacji nieprzyjaznych dla sieci. Kernel Linuksa oferuje RED, czyli ‘Losowe Wczesne Wykrywanie’, zwane również ‘Losowym Wczesnym Odrzucaniem’ co lepiej oddaje specyfikę działania algorytmu.

Co prawda RED nie jest uniwersalnym lekarstwem, aplikacje które nie potrafią zapewnić odpowiedniego algorytmu wysyłania pakietów otrzymują nierówny udział w przepustowości, ale zastosowanie algorytmu RED znacząco zmniejsza szkodę dla przepustowości i opóźnienia nakładane na ruch z różnych aplikacji.

RED odrzuca pakiety na podstawie statystyk dla przepływów, zanim ich liczba osiągnie górne ograniczenie. Powoduje to, że zagęszczenia na routerach szkieletowych spowalniają ruch bardziej przyjaźnie i unika się synchronizacji retransmisji. Pomaga to również protokołowi TCP znaleźć ‘właściwą’ prędkość szybciej, przez odrzucenie niektórych pakietów wcześniej - rozmiary kolejek są małe a opóźnienie pod kontrolą. Prawdopodobieństwo, że pakiet z danego połączenia zostanie odrzucony, jest proporcjonalne do przepustowości tego połączenia a nie do liczby przesyłanych pakietów.

RED jest dobrą kolejką dla szkieletów sieci, w których nie możesz pozwolić sobie na złożoność śledzenia stanów dla każdej sesji, czego wymagałoby zapewnienie sprawiedliwego kolejkowania.

By używać RED, musisz określić trzy parametry: Min, Max i seria. Min określa minimalny rozmiar kolejki w bajtach zanim rozpocznie się odrzucanie. Max to miękkie maksimum, pod którym algorytm będzie się starał utrzymać, a seria to maksymalna liczba pakietów która może ‘przejść w serii’.

Ustawiając min powinieneś wyliczyć najwyższe akceptowalne opóźnienie w kolejce i pomnożyć tą wartość przez przepustowość. Na przykład, na moim 64kbit/s połączeniu ISDN chciałbym dopuścić maksymalne opóźnienie 200ms, więc ustawiam wartość ‘min’ na 1600 bajtów. Jeśli ustawię ją za nisko, spadnie przepustowość a jeśli zbyt wysoko - wzrosną opóźnienia. Ustawienie zbyt małej wartości ‘min’ nie jest zamiennikiem dla zmniejszaniu MTU na wolnym łączu, by polepszyć jakość połączeń interaktywnych.

Powinieneś ustawić ‘max’ na dwukrotność ‘min’ by zapobiec synchronizacjom. Na wolnych połączeniach z małymi wartościami ‘min’ dobrym pomysłem będzie ustawienie wartości ‘max’ na czterokrotność wartości ‘min’.

‘Seria’ kontroluje jak algorytm RED odpowiada na serie. Seria musi być ustawiona wyżej niż wartość ‘min’. Eksperymentując, doszedłem do następującego wzoru: (min+min+max)/(3*rozmiar średniego pakietu).

Dodatkowo, musisz ustawić limit i średni rozmiar pakietu. Limit to granica bezpieczeństwa, po przekroczeniu której RED włącza odrzucanie nadmiaru. Ustawiam limit zwykle na ośmiokrotność wartości ‘max’. Średni rozmiar pakietu (‘avpkt’) powinien być ustawiony tak jak sugeruje nazwa - na średni rozmiar pakietu. 1000 działa dobrze na szybkich połączeniach z Internetem, dla których wartość MTU wynosi 1500 bajtów.

Zapoznaj się z >dokumentem o kolejkowaniu RED (http://www.aciri.org/floyd/papers/red/red.html) autorstwa Sally Floyd i Van Jacobson, aby poznać więcej szczegółów technicznych.

Ogólne Losowe Wczesne Wykrywanie (ang. „Generic Random Early Detection”, GRED)

Niewiele wiadomo o GRED. Wygląda jak RED z wieloma kolejkami wewnętrznymi, a aktualna kolejka wewnętrzna wybierana jest na podstawie wartości pola TC_INDEX Diffserv. Zgodnie ze slajdami znajdującymi się pod tym (http://www.davin.ottawa.on.ca/ols/img22.htm) adresem, GRED zawiera możliwości implementacji algorytmu DWRED Cisco, jak również implementacji RIO Davea Clarka.

Każda wirtualna kolejka może mieć swoje własne parametry odrzucania.

FIXME: Postarajmy się by Jamal lub Werner powiedział coś więcej.

Emulacja VC/ATM

Jest to duży projekt Wernera Almesbergera umożliwiający budowanie ‘obwodów wirtualnych’ (ang. „Virtual Circuits”) ponad gniazdami TCP/IP. Obwody wirtualne to koncepcja zapożyczona z teorii sieci ATM.

Po więcej informacji zajrzyj pod ten adres (http://linux-atm.sourceforge.net/).

Ważony Round Robin (ang. „Weighted Round Robin”, WRR)

Ta kolejka nie jest włączona do standardowych kerneli ale można ją ściągnąć spod tego adresu (http://wipl-wrr.dkik.dk/wrr/). Aktualnie testowano ją tylko z kernelami 2.2 ale prawdopodobnie będzie działała również z kernelami 2.4/2.5.

Kolejka WRR dystrybuuje przepustowość pomiędzy swoje klasy, przy użyciu schematu ważonego round robin. To znaczy, że podobnie jak kolejka CBQ zawiera klasy do których można dołączać inne kolejki. Wszystkie klasy posiadające wystarczające zapotrzebowanie otrzymają pasmo sieciowe proporcjonalne to wag skojarzonych z ich klasami. Wagi mogą być ustawione ręcznie przy użyciu programu **tc**, ale mogą być również automatycznie zmniejszane dla klas przesyłających zbyt dużo danych.

Kolejka ma wbudowany klasyfikator przydzielający pakiety przychodzące lub rozsyłane do różnych maszyn, do różnych kolejek. Możesz używać adresu MAC lub IP oraz adresu źródłowego lub docelowego. Jednak adresów MAC możesz używać tylko wtedy, gdy Linux działa jako most ethernetowy. Klasy są automatycznie przydzielane do komputerów, na podstawie pakietów które zostały przesłane.

Kolejka może być bardzo przydatna w sytuacjach, gdy wiele niezwiązanych ze sobą osoób współdzieli połączenie Internetowe. Zestaw skryptów do skonfigurowania zachowania algorytmu WRR dla takiej konfiguracji jest główną częścią dystrybucji WRR.

Rozdział 15. Książka kucharska

Sekcja ta zawiera wpisy ‘książki kucharskiej’, które mogą pomóc ci rozwiązać problemy. Nie może być ona jednak traktowana jako substytut zrozumienia całości materiału, więc postaraj się to najpierw zrobić zanim tu zajrzysz.

Praca z wieloma lokalizacjami z różnymi SLA

Możesz to zrobić na wiele sposobów. Apache oferuje wsparcie tego trybu modułem, ale pokażemy jak sam Linux może zrobić to dla ciebie oraz dla innych usług. Przykłady skradziono z prezentacji Jamal’a Hadiego, o którym wspomniano poniżej.

Załóżmy, że mamy dwóch klientów, obu z http, ftp i dźwiękiem transmitowanym strumieniami. Chcemy sprzedać im określoną część pasma sieciowego. Konfiguracje ustalamy na serwerze.

Klient A powinien otrzymać najwyżej 2 megabity, a klient B zapłacił za 5 megabitów. Oddzielamy klientów, tworząc wirtualne adresy IP na serwerze.

```
# ip address add 188.177.166.1 dev eth0
# ip address add 188.177.166.2 dev eth0
```

Skonfigurowanie odrębnych serwerów z odpowiednimi adresami IP jest twoim zadaniem. Wszystkie popularne demony oferują dla tego wsparcie.

Na początek, dołączamy kolejkę CBQ do eth0:

```
# tc qdisc add dev eth0 root handle 1: cbq bandwidth 10Mbit cell 8 avpkt 1000 \
  mpu 64
```

Tworzymy następnie klasy dla naszych klientów:

```
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 10Mbit rate \
  2MBit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
# tc class add dev eth0 parent 1:0 classid 1:2 cbq bandwidth 10Mbit rate \
  5Mbit avpkt 1000 prio 5 bounded isolated allot 1514 weight 1 maxburst 21
```

Dodajemy filtry dla obu klas:

```
##FIXME: Po co ta linia i co robi? Co to jest podzielnik?
##FIXME: Podzielnik ma coś do czynienia z tablicą mieszającą i z liczbą
##       wiader - ahu
# tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 1: u32 divisor 1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.1
  flowid 1:1
# tc filter add dev eth0 parent 1:0 prio 5 u32 match ip src 188.177.166.2
  flowid 1:2
```

I to już.

FIXME: dlaczego nie ma filtra wiadra żetonów? czy jest gdzieś domyślna wartość pfifo_fast?

Ochrona komputera przed powodziami SYN

Z dokumentacji dla iproute Aleksieja, zaadoptowanej do netfilter i bardziej prawdopodobnymi ścieżkami. Jeśli tego użyjesz, weź pod uwagę dostosowanie numerów do sensownych wartości dla twojego systemu.

Jeśli chcesz chronić całą sieć, pomiń ten skrypt, ponieważ jest on zaprojektowany dla pojedyńczego hosta.

Okazuje się, że potrzebujesz najnowszej wersji narzędzi iproute2 by to pracowało z 2.4.0.

```
#! /bin/sh -x
#
# przykładowy skrypt demonstrujący możliwości kontrolowania
# ruchu przychodzącego; pokazuje jak można ograniczyć
# częstotliwość nadchodzących pakietów SYN;
# jest użyteczny jako zabezpieczenie przed atakami TCP-SYN;
# możesz użyć ipchains by rozszerzyć ochronę przed pakietami SYN
#
# ścieżki do różnych narzędzi
# zmień by odzwierciedlały twoją konfiguracje
#
TC=/sbin/tc
IP=/sbin/ip
IPTABLES=/sbin/iptables
INDEV=eth2
#
# zaznacz wszystkie przychodzące przez urządzenie $INDEV pakiety SYN
# wartością ‘1’
################################################################
$iptables -A PREROUTING -i $INDEV -t mangle -p tcp --syn \
  -j MARK --set-mark 1
################################################################
#
# zainstaluj kolejkę qdisc na interfejsie wchodzącym
################################################################
$TC qdisc add dev $INDEV handle ffff: ingress
################################################################

# Pakiety SYN mają 40 bajtów (320 bitów), więc trzy pakiety SYN
# to 960 bitów (prawie jeden kilobit); ograniczamy częstotliwość
# nadchodzących pakietów SYN do 3/sekundę (niezbyt użyteczne ale
# to przykład - JHS
################################################################
$TC filter add dev $INDEV parent ffff: protocol ip prio 50 handle 1 fw \
police rate 1kbit burst 40 mtu 9k drop flowid :1
################################################################


#
echo "---- qdisc parameters Ingress  ----------"
$TC qdisc ls dev $INDEV
echo "---- Class parameters Ingress  ----------"
$TC class ls dev $INDEV
echo "---- filter parameters Ingress ----------"
$TC filter ls dev $INDEV parent ffff:
```

```
# kasowanie kolejki dla ruchu przychodzącego
#$TC qdisc del $INDEV ingress
```

Ograniczenie częstotliwości ICMP by zapobiec atakom DDoS

Ataki typu DDoS stały się ostatnio poważnym kłopotem w Internecie. Poprzez odpowiednie filtrowanie i ograniczanie częstotliwości w twojej sieci, możesz zarówno zapobiec staniu się ofiarom takich ataków jak i ich powodem.

Powinieneś filtrować swoje sieci tak, by nie umożliwiać nie-lokalnym adresom źródłowym IP opuszczać twoją sieć. Powstrzymuje to ludzi przed anonimowym rozsyłaniem śmieci po Internecie.

Ograniczanie częstotliwości pokazano już w zasadzie powyżej. By odświeżyć ci pamięć, ponownie nasz rysunek:

```
[The Internet] ---<E3, T3, cokolwiek>--- [router linuksowy] --- [Office+ISP]
                                      eth1          eth0
```

Ustawiamy wstępne reguły:

```
# tc qdisc add dev eth0 root handle 10: cbq bandwidth 10Mbit avpkt 1000
# tc class add dev eth0 parent 10:0 classid 10:1 cbq bandwidth 10Mbit rate \
  10Mbit allot 1514 prio 5 maxburst 20 avpkt 1000
```

Jeśli masz interfejsy 100Mbit'owe lub szybsze, odpowiednio zmodyfikuj wartości. Teraz musisz określić jak wiele ruchu ICMP chcesz przepuszczać. Można to zmierzyć programem **tcpdump**, każąc mu zapisywać przez chwilę do pliku pakiety i policzyć ile z nich to ICMP. Nie zapomnij zwiększyć długości zbieranych pakietów!

Jeśli taki pomiar jest niepraktyczny, możesz wybrać na przykład 5% dostępnej przepustowości. Ustawmy naszą klasę:

```
# tc class add dev eth0 parent 10:1 classid 10:100 cbq bandwidth 10Mbit rate \
  100Kbit allot 1514 weight 800Kbit prio 5 maxburst 20 avpkt 250 \
  bounded
```

Ustawia ona limit na pułapie 100Kbitów. Potrzebujemy teraz filtrować, by przypisać ruch ICMP do odpowiedniej klasy:

```
# tc filter add dev eth0 parent 10:0 protocol ip prio 100 u32 match ip
  protocol 1 0xFF flowid 10:100
```

Priorytetyzacja ruchu interaktywnego

Jeśli masz wiele danych przechodzących przez łącze a chcesz zrobić coś za pomocą telnetu lub ssh, może być to bardzo utrudnione. Inne pakiety będą blokować przesyłanie informacji o twoich klawiszach. Czy nie byłoby

fajnie móc określić, że pakiety sesji interaktywnych przemykają się obok głównego ruchu? Linux może to zrobić!

Jak poprzednio, musimy obsłużyć ruch w obie strony. Oczywiście, działa to najlepiej gdy po obu stronach połączenia mamy Linuksa, chociaż inne Uniksy też to potrafią. Skonsultuj się ze swoim lokalnym guru od Solarisa/BSD.

Standardowa planer kolejki ‘pfifo_fast’ ma trzy różne ‘pasma’. Ruch w paśmie 0 przesyłany jest pierwszy, następnie ruch z pasm 1 i 2. Oczywiście ważne jest, by nasz ruch interaktywny wpisać do pasma 0!

Adaptujemy ten przykład z ipchains HOWTO:

Są cztery rzadko używane bity w nagłówku IP, nazywające się bitami Typu Usługi (ang. „Type of Service”). Określają one jak traktowane są pakiety; te cztery bity to ‘Minimalna Zwłoka’, ‘Maksymalna Przepustowość’, ‘Maksymalna Niezawodność’ i ‘Minimalny Koszt’. Tylko jeden z tych bitów może być ustawiony. Rob van Nieuwerk, autor kodu interpretującego pole ToS określa to tak:

```
Dla mnie szczególnie ważny jest bit ‘Minimalna Zwłoka’. Ustawiam go dla pakietów
należących do sesji ‘interaktywnych’ w routerze przekazującym ruch wyżej. Łącze
to  modemowe 33k. Linuks priorytezuje pakiety w 3 kolejki. W ten sposób mam
akceptowalną wydajność ruchu interaktywnego i jednocześnie dobre transfery
innych danych.
```

Zwykle używa się tego do kontrolowania połączeń telnet i kontroli ftp przez ustawienie ‘Minimalnej Zwłoki’ a dla danych ftp ‘Maksymalna Przepustowość’. Robi się to jak poniżej, na routerze obsługującym ruch wysyłany:

```
# iptables -A PREROUTING -t mangle -p tcp --sport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A PREROUTING -t mangle -p tcp --sport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

Działa to tylko dla danych wychodzących sesją telnet to zdalnych komputerów z twojego. W drugą stronę obsługę tych informacji zapewniają same aplikacje - telnet i ssh ustawiają odpowiednie pola ToS w pakietach wychodzących automatycznie.

Jeśli miałbyś aplikację, która tego nie robi możesz zawsze ustawić to ręcznie za pomocą netfilter. Na lokalnym komputerze:

```
# iptables -A OUTPUT -t mangle -p tcp --dport telnet \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp \
  -j TOS --set-tos Minimize-Delay
# iptables -A OUTPUT -t mangle -p tcp --dport ftp-data \
  -j TOS --set-tos Maximize-Throughput
```

Przezroczyste cache przy użyciu netfilter, iproute2, ipchains i squida

Tą sekcję podesłał czytelnik Ram Narula z akcji Internet dla Edykacji (Tajlandia).

Standardową techniką by to osiągnąć w Linuksie jest użycie ipchains PO upewnieniu się, że ruch na port wychodzący 80 routowany jest przez serwer na którym pracuje squid.

Są trzy podstawowe sposoby by upewnić się, że ruch wychodzący na port 80 jest tak obsługiwany, a czwarty podamy tutaj.

Niech router bądący bramą domyślną to robi

> Jeśli możesz poinstruować swój router będący bramą by pakiety pasujące do portu docelowego 80 były wysyłane na adres IP serwera ze squidem.
>
> ALE
>
> Spowoduje to dodatkowe obciążenie rutera, a poza tym niektóre komercyjne routery mogą tego nawet nie obsługiwać.

Użycie przełącznika warstwy czwartej

> Przełączniki warstwy czwartej mogą to obsłużyć.
>
> ALE
>
> Taki przełącznik kosztuje zwykle niemało. Zwykle oznacza to więcej niż koszt typowego routera + dobrego serwera linuksowego.

Użycie serwera cacheującego jako bramy sieciowej

> Możesz wymusić by CAŁY ruch przechodził przez serwer cache.
>
> ALE
>
> Jest to trochę ryzykowne, ponieważ Squid zużywa całkiem dużo mocy roboczej procesora, co może spowodować spowolnienie całego ruchu sieciowego lub w ogóle załamanie się serwera i odcięcie całej sieci od Internetu.

Router Linuksowy + netfilter

> Zastosowanie netfilter udostępnia kolejną technikę, dzięki znaczeniu pakietów przeznaczonych do docelowego portu 80 oraz użyciu iproute2 dla kierowania zaznaczonych pakietów do serwera ze squid'em.

```
|----------------|
| Implementacja  |
|----------------|
```

```
 Addresses used
 10.0.0.1 naret (serwer NetFilter)
 10.0.0.2 silom (serwer Squid)
 10.0.0.3 donmuang (ruter podłączony do Internetu)
 10.0.0.4 kaosarn (inny serwer w sieci)
 10.0.0.5 RAS
 10.0.0.0/24 główna sieć
```

```
 10.0.0.0/19 cała sieć

|---------------|
| Diagram sieci |
|---------------|

            Internet
                |
             donmuang
                |
------------hub/switch----------
|        |             |       |
naret   silom        kaosarn  RAS etc.
```

Po pierwsze, cały ruch musi przechodzić przez naret a zapewnimy to, ustawiając go jako domyślną bramę dla wszystkich maszyn oprócz silom. Domyślną bramą dla silom będzie donmuang (10.0.0.3) bo jeśli nie - stworzymy pętle dla ruchu WWW.

Wszystkie serwery w mojej sieci mają 10.0.0.1 jako domyślną bramę, adres ten miał poprzednio ruter donmuang, więc zmieniłem adres IP donmuang na 10.0.0.3 i nadałem naret adres IP 10.0.0.1.

```
Silom
-----
-ustawiamy squida i ipchains
```

Skonfiguruj serwer squida na silom i upewnij się, że wspiera przeźroczyste cache/proxy. Domyślnym portem jest zwykle 3128, więc cały ruch na port 80 powinien być przekierowany na port lokalny 3128. Można to wykonać przy użyciu ipchains wykonując co następuje:

```
silom# ipchains -N allow1
silom# ipchains -A allow1 -p TCP -s 10.0.0.0/19 -d 0/0 80 -j REDIRECT 3128
silom# ipchains -I input -j allow1
```

Lub na sposób netfilter:

```
silom# iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j REDIRECT --to-port 3128
```

Zwróć uwagę na to, że możesz mieć również inne wpisy.

Po więcej informacji jak ustawić Squida zajrzyj na stronę FAQ Squida (http://squid.nlanr.net).

Upewnij się że przekazywanie IP jest na tym serwerze włączone a domyślną bramą dla tego serwera jest ruter donmuang (NIE naret).

```
Naret
-----
-ustawiamy iptables i iproute2
-wyłączamy wiadomości ICMP REDIRECT (jeśli potrzeba)
```

1. Zaznaczamy pakiety przeznaczone do portu docelowego 80 wartością 2

   ```
   naret# iptables -A PREROUTING -i eth0 -t mangle -p tcp --dport 80 \
    -j MARK --set-mark 2
   ```

2. Konfigurujemy iproute2 tak, by kierowało pakiety oznaczone wartością ‘2’ do silom

   ```
   naret# echo 202 www.out >> /etc/iproute2/rt_tables
   naret# ip rule add fwmark 2 table www.out
   naret# ip route add default via 10.0.0.2 dev eth0 table www.out
   naret# ip route flush cache
   ```

   Jeśli donmuang i naret są w ten samej podsieci, neret nie powinien wysyłać wiadomości ICMP REDIRECT. W tym przypadku tak jest, więc wyłączamy je pisząc:

   ```
   naret# echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
   naret# echo 0 > /proc/sys/net/ipv4/conf/default/send_redirects
   naret# echo 0 > /proc/sys/net/ipv4/conf/eth0/send_redirects
   ```

Konfiguracja jest kompletna, sprawdzamy ją:

```
Na naret:

naret# iptables -t mangle -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
MARK       tcp  --  anywhere             anywhere           tcp dpt:www MARK set 0x2

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

naret# ip rule ls
0:      from all lookup local
32765:  from all fwmark        2 lookup www.out
32766:  from all lookup main
32767:  from all lookup default

naret# ip route list table www.out
default via 203.114.224.8 dev eth0

naret# ip route
10.0.0.1 dev eth0  scope link
10.0.0.0/24 dev eth0  proto kernel  scope link  src 10.0.0.1
127.0.0.0/8 dev lo  scope link
default via 10.0.0.3 dev eth0

(upewnij się, że silom należy do jednej z powyższych linii, w tym przypadku
 do 10.0.0.0/24)
```

```
|-----------|
|-SKOŃCZONE-|
|-----------|
```

Diagram przepływu ruchu po implementacji

```
|---------------------------------------|
|Diagram przepływu ruchu po implementacji|
|---------------------------------------|

INTERNET
/\
||
\/
-----------------donmuang router---------------------
/\                                    /\         ||
||                                    ||         ||
||                                    \/         ||
naret                                silom      ||
*ruch do portu 80 ==================>(cache)     ||
/\                                    ||         ||
||                                    \/         \/
\\==================================kaosarn, RAS, etc.
```

Zauważ że sieć jest asymetryczna i mamy dodatkowy ‘hop’ na ścieżce ruchu wychodzącego.

Poniżej opis przepływu pakietu przechodzącego sieć z kaosarn do i z Internetu.

```
Dla ruchu HTTP:
kaosarn http wywołuje->naret->silom->donmuang->internet
http odpowiada z Internetu->donmuang->silom->kaosarn

Dla wywołań nie będących wywołaniami HTTP (np. telnet):
wychodzące dane z kaosarn->naret->donmuang->internet
przychodzące dane z Internetu->donmuang->kaosarn
```

Unikanie problemów z rozpoznaniem przez MTU trasy a ustawienia MTU dla tras

Jeśli chodzi o normalny ruch, Internet generalnie działa lepiej dla większych pakietów. Każdy z nich to decyzja dotycząca routingu. Wysłanie 1 megabajtowego pliku może oznaczać około 700 pakietów jeśli pakiety będą maksymalnej wielkości lub 4000 jeśli pakiety będą najmniejszej domyślnej wielkości.

Jednak nie wszystkie hosty w Internecie obsługują wysyłanie pełnych 1460 bajtów ładunku na pakiet. Niezbędne staje się więc określenie największego ładunku który ‘zmieści się’ w pakiecie by zapewnić optymalne połączenie.

Proces ten nazywany jest ‘Określaniem MTU ścieżki’, gdzie MTU oznacza ‘Maksymalną Jednostkę Transmisji’ (ang. „Maximum Transfer Unit.”).

Kiedy router otrzyma pakiet, który jest zbyt duży by zostać wysłanym w jednym kawałku a ustawiono w nim bit ‘Nie fragmentować’, zwraca pakiet ICMP mówiący, że musiał odrzucić pakiet w związku z niemożnością podzielenia go. Dla autora pierwotnego pakietu jest to podpowiedź by wysyłać mniejsze pakiety, a przez konsekwentne pomniejszanie rozmiaru pakietu może znaleźć optymalny rozmiar pakietu dla połączenia przez tą konkretną ścieżkę.

Ten sposób działał dobrze, dopóki Internetu nie odkryli huligani, którzy robią wszystko by przerwać komunikację. Spowodowało to, że administratorzy albo zablokowali zupełnie albo poddali kontroli ruch ICMP w niezbyt trafionej próbie podniesienia bezpieczeństwa usług internetowych.

Obecnie metoda ta działa mniej skutecznie i w ogóle zawodzi dla niektórych tras, co prowadzi do różnych dziwnych sesji TCP/IP które po jakimś czasie umierają.

Co prawda nie mam na to dowodu, ale dwie lokalizacje których używałem miały ten problem. Obie używały Alteon Acedirectors przed powodującymi takie zachowanie systemami - być może ktoś z większą wiedzą może dostarczyć informacji dlaczego tak się dzieje.

Rozwiązanie

Jeśli spotkasz się z tym problemem, możesz wyłączyć Rozpoznawanie MTU trasy ustawiając je ręcznie. Koos van den Hout pisze:

> Następujący problem: ustawiam mtu/mru mojego łącza pracującego na PPP na wartość 296, ponieważ to tylko 33k i nie mam wpływu na kolejkowanie po drugiej stronie. Przy 296 odpowiedź na wciśnięcie klawisza jest w sensownych ramach czasowych. A po mojej stronie mam router prowadzący maskaradę na Linuksie. Ostatnio rozdzieliłem ‘serwer’ i ‘router’ tak by większość aplikacji pracowała na innej maszynie niż ta, która prowadzi trasowanie. I pojawił się wielki problem z zalogowaniem na IRC. Wielka panika! Po chwili kopania zorientowałem się, że łącze się z serwerem IRC, dostaję nawet komunikat ‘połączony’ ale nie ma już wiadomości dnia. Sprawdziłem co może być źle i zauważyłem, że już wcześniej miałem problemy z osiągnięciem pewnych witryn WWW w związku z MTU - nie było problemu gdy MTU wynosiło 1500 a pojawił się gdy MTU wynosi 296. Ponieważ serwery IRC blokują prawie każdy rodzaj ruchu nie potrzebny w normalnej pracy, blokują również ICMP. Udało mi się przekonać operatorów serwerów WWW że to była przyczyna problemu, ale administratorzy serwerów IRC nie chcieli tego poprawiać. Musiałem się zatem upewnić, że wychodzący maskaradowany ruch ma mniejsze MTU na łączu zewnętrznym. Z kolei lokalny ruch ethernetowy ma mieć normalne MTU (dla czegoś takiego jak np. ruch NFS).
>
> Rozwiązanie:
>
> ```
> ip route add default via 10.0.0.1 mtu 296
> ```
>
> (10.0.0.1 to domyślna brama i wewnętrzny adres routera prowadzącego maskaradę)

Generalnie, możliwe jest wymuszenie rozpoznawania trasy przez MTU poprzez określenie specyficznych tras. Na przykład, jeśli tylko określona podsieć powoduje problemy, to powinno pomóc:

```
ip route add 195.96.96.0/24 via 10.0.0.1 mtu 1000
```

Unikanie problemów z rozpoznaniem przez MTU trasy a Zmniejszanie MSS (dla użytkowników kablowych ADSL, PPPoE i PPPtP)

Jak już wyjaśniono powyżej, PMTUD nie działa już tak dobrze jak powinno. Jeśli wiesz na pewno, że ‘hop’ gdzieś w twojej sieci ma ograniczone (<1500) MTU, nie możesz polegać na tym mechaniźmie PMTUD by to

sprawdzić. Poza MTU, jest jeszcze inny sposób by ustalić maksymalny rozmiar pakietu, tzw. ‘Maksymalny Rozmiar Segmentu’ (ang. „Maximum Segment Size”, MSS). Jest to jedno z pól opcji TCP w pakiecie SYN.

Ostatnie kernele Linuksa i parę sterowników PPPoE (zwłaszcza sterownik Roaring Penguin’a) udostępnia możliwość ‘Zmniejszania MSS’

Dobra strona tego jest taka, że przez ustawienie wartości MSS sygnalizujesz drugiej stronie ‘nie próbuj wysyłać mi pakietów większych niż ta wartość’. Nie potrzeba żadnego ruchu ICMP by zapewnić działanie w takich warunkach.

Zła strona to oczywiste odejście od standardu - przez modyfikowanie pakietów. Po dostarczeniu ci tej wiedzy, używamy tego triku w wielu miejscach i działa to bardzo ładnie.

Aby tak było, potrzebujesz narzędzia iptables przynajmniej w wersji 1.2.1a i kernela w wersji 2.4.3 lub wyższej. Podstawowa składnia wygląda tak:

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS  --clamp-mss-to-pmtu
```

Polecenie to wylicza odpowiednią wartość MSS dla połączenia. Jeśli czujesz się mocny, lub wiesz lepiej co dla ciebie dobre, możesz spróbować czegoś takiego:

```
# iptables -A FORWARD -p tcp --tcp-flags SYN,RST SYN -j TCPMSS --set-mss 128
```

Polecenie to ustawia MSS przechodzących pakietów SYN na wartość 128. Można go użyć w przypadku konfiguracji z VoIP używającym małych pakietów oraz wielkich pakietów http powodujących przeskoki w przesyłanej mowie.

Najlepszy ‘udrażniacz’ dla ruchu sieciowego: małe opóźnienia, szybkie wrzucanie i ściąganie

Uwaga: ten skrypt został ostatnio udoskonalony, ponieważ poprzednio działał tylko dla hostów działających w oparciu o Linuksa. Możesz chcieć go uaktualnić jeśli masz jakieś maszyny na Windows czy Macintoshe, a zauważyłeś że nie mogły wysyłać danych szybciej niż inni je ściągali.

Starałem się stworzyć święty Grall:

Zapewnić i utrzymać niskie opóźnienia dla ruchu interaktywnego

> Oznacza to, że ściąganie czy wrzucanie gdzieś plików nie powinno przeszkadzać sesjom SSH czy telnet. Są to najważniejsze rzeczy i nawet opóźnienia rzędu 200ms to już dużo.

Zapewnienie możliwości ‘serfowania’ przy sensownych prędkościach mimo ściągania czy wrzucania

> Mimo że ruch ‘http’ zalicza się w zasadzie do ruchu ‘masowego’ inny ruch nie powinien go zalewać.

Upewnienie się, że wrzucanie danych nie przeszkadza ściąganiu

> Bardzo często obserwowany fenomen, który polega na tym, że ruch wychodzący drastycznie zmniejsza prędkość ściągania.

Wychodzi na to, że to wszystko jest możliwe jeśli poświęcimy mały kawałek przepustowości. Powodem dla których wrzucanie, ściąganie i sesje ssh szkodzą sobie wzajemnie jest obecność długich kolejek w urządzeniach dostępowych, takich jak modemy kablowe czy DSL.

Następna sekcja wyjaśnia dokładniej co powoduje opóźnienia i jak można to naprawić. Możesz ją spokojnie pominąć i przejść dalej jeśli nie obchodzi cię jak robi się całą tą magię.

Dlaczego nie działa to najlepiej w domyślnej konfiguracji

Dostawcy Internetowi doskonale zdają sobie sprawę, że często sprawdza się jak szybko można przez nich ściągać dane. Poza dostępną szerokością pasma, prędkość ściągania cierpi również od gubionych pakietów co mocno odbija się na wydajności TCP/IP. Oczywiście duże kolejki mogą temu zapobiec i zwiększyć szybkość ściągania. I w związku z tym, zwykle dostawcy Internetowi tak konfigurują swoje urządzenia.

Odbija się to jednak na usługach interaktywnych. Informacja o przyciśnięciu klawisza musi przejść kolejkę wysyłającą co może zająć nawet całe sekundy (!) i trafić do zdalnej maszyny. Znak odpowiadający kodowi klawisza jest następnie wyświetlany, co powoduje powrót pakietu przez kolejkę odbierającą zlokalizowaną u dostawcy i dopiero dociera on do twojego komputera.

Ten dokument wyjaśnia jak manipulować procesem kolejkowania na wiele sposobów, ale niestety nie możemy tego robić na wszystkich kolejkach. Oczywiście kolejki w urządzeniach dostawców są poza naszym zasięgiem, ale kolejka wysyłająca prawdopodobnie znajduje się w naszym modemie kablowym lub DSL. Możesz mieć lub możesz nie mieć do niej dostępu. Zwykle nie masz.

I co zrobić? Ponieważ nie możemy kontrolować żadnej z tych kolejek, muszą zostać wyeliminowane i przeniesione na router linuksowy. To na szczęście jest możliwe.

Ograniczamy prędkość wysyłania danych

> Przez ograniczenie tej prędkości do wartości niewiele niższej niż faktyczna dostępna przepustowość zapobiegamy budowaniu kolejek w modemie. Znajduje się ona teraz na Linuksie.

Ograniczamy prędkość odbierania danych

> To jest trochę trudniejsze, ponieważ tak naprawdę nie możemy wpłynąć na to jak szybko internet wysyła do nas dane. Ale możemy odrzucać pakiety przychodzące za szybko, co spowoduje że TCP/IP zwolni do częstotliwości, którą jesteśmy w stanie zaakceptować. Ponieważ nie chcemy tego robić niepotrzebnie, konfigurujemy rozmiar ‘serii’, którą jesteśmy skłonni przyjąć przy większych prędkościach.

Gdy już to zostało zrobione, wyeliminowaliśmy zupełnie kolejkę przychodzącą (poza krótkimi seriami) i uzyskaliśmy możliwość zarządzania kolejką wychodzącą z całą potęgą którą oferuje Linux.

To co pozostaje, to upewnić się, że ruch interaktywny obsługiwany jest przed kolejką wychodzącą. Aby zapewnić sytuację, w której jednoczesne wrzucanie i ściąganie danych nie szkodzi sobie, przesuwamy również na czoło tej kolejki pakiety ACK. To właśnie to powoduje zwykle poważne spowolnienia obserwowane gdy generowane jest dużo ruchu w obie strony. ‘Potwierdzenia’ (ang. „acknowledgements”, ACK) dla ruchu przychodzącego muszą konkurować z ruchem wychodzącym i w związku z tym ich dotarcie jest opóźnianie.

Jeśli zrobimy to wszystko, dostaniemy następujące pomiary na doskonałym łączu ADSL z xs4all w Holandii:

```
Podstawowe opóźnienia:
round-trip min/avg/max = 14.4/17.1/21.7 ms

Bez ‘udrożniacza’ ruchu podczas pobierania danych:
round-trip min/avg/max = 560.9/573.6/586.4 ms

Bez ‘udrożniacza’ ruchu podczas wysyłania danych:
round-trip min/avg/max = 2041.4/2332.1/2427.6 ms
```

```
Z ‘udrożniaczem’, podczas wysyłania danych z prędkością 220kbit/s:
round-trip min/avg/max = 15.7/51.8/79.9 ms

Z ‘udrożniaczem’, podczas pobierania danych z prędkością 850kbit/s:
round-trip min/avg/max = 20.4/46.9/74.0 ms

Wysyłanie i pobieranie danych jednocześnie na prawie maksymalnej prędkości.
Opóźnienia skaczą do 850ms, choć nadal nie wiem dlaczego.
```

To czego możesz spodziewać się po tym skrypcie bardzo zależy od faktycznej prędkości wysyłania danych. Podczas wysyłania danych z pełną prędkością, zawsze będzie pakiet poprzedzający twój wciśnięty klawisz. Jest to minimalna wartość opóźnienia, którą możesz osiągnąć - podziel MTU prez prędkość wysyłania danych. Wartości typowe będą oscylowały trochę powyżej tej wartości. Zmniejszenie MTU powinno przynieść poprawę wyników!

Poniżej dwie wersje tego skryptu, jeden z doskonałym HTB Devik’a a drugi z CBQ dostępną w każdym jądrze Linuksa, w przeciwieństwie do HTB. Oba są przetestowane i pracują dobrze.

Skrypt (CBQ)

Działa na wszystkich kernelach. W obrębie kolejki CBQ umieszczamy dwie ‘Kolejki ze Stochastycznym Równym Podziałem’ (SFQ) by upewnić się, że wiele strumieni przenoszących dane ‘masowe’ nie będą sobie przeszkadzać.

Ruch przychodzący podlega polityce wykonywanej przez filtr tc zawierających TBF.

Możesz ulepszyć ten skrypt przez dodanie słowa kluczowego ‘bounded’ do linii zaczynającej się od ‘tc class add .. classid 1:20’. Jeśli zmniejszysz wartość swojego MTU, pamiętaj o obniżeniu wartości ‘allot’ i ‘avpkt’!.

```
#!/bin/bash

# Konfiguracja dla połączenia internetowego w domu
#
#
# Ustaw poniższe wartości trochę poniżej faktycznych prędkości
# ściągania i wysyłania (w kilobitach)
DOWNLINK=800
UPLINK=220
DEV=ppp0

# wyczyść kolejki dla wysyłania i ściągania danych, nie informuj o błędach
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev $DEV ingress 2> /dev/null > /dev/null

###### wysyłanie danych

# instalujemy w korzeniu CBQ

tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 10mbit

# kształtujemy wszystko do prędkości $UPLINK - zapobiega to tworzeniu
# się dużych kolejek na modemie DSL co zniszczyłoby panowanie nad
# opóźnieniami
```

```
tc class add dev $DEV parent 1: classid 1:1 cbq rate ${UPLINK}kbit \
allot 1500 prio 5 bounded isolated

# klasa z dużym priorytetem 1:10:

tc class add dev $DEV parent 1:1 classid 1:10 cbq rate ${UPLINK}kbit \
   allot 1600 prio 1 avpkt 1000

# klasa domyślna dla ruchu ‘masowego’ 1:20 - otrzymuje trochę mniej
# ruchu i ma mniejszy priorytet

tc class add dev $DEV parent 1:1 classid 1:20 cbq rate $[9*$UPLINK/10]kbit \
   allot 1600 prio 2 avpkt 1000

# obie klasy kontrolowane są przez SFQ:
tc qdisc add dev $DEV parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $DEV parent 1:20 handle 20: sfq perturb 10

# włączamy filtry
# TOS Minimum Delay (ssh, NOT scp) in 1:10:
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
      match ip tos 0x10 0xff  flowid 1:10

# ICMP (protokół IP numer 1) w klasie interaktywnej 1:10 tak, byśmy
# mogli wykonywać pomiary i pochwalić się naszym przyjaciołom
tc filter add dev $DEV parent 1:0 protocol ip prio 11 u32 \
       match ip protocol 1 0xff flowid 1:10

# By zwiększyć prędkość ściągania danych w trakcie wysyłania, pakiety
# ACK umieszczamy do klasy interaktywnej

tc filter add dev $DEV parent 1: protocol ip prio 12 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   match u8 0x10 0xff at 33 \
   flowid 1:10

# cała ‘nie-interaktywna’ reszta trafia do 1:20

tc filter add dev $DEV parent 1: protocol ip prio 13 u32 \
   match ip dst 0.0.0.0/0 flowid 1:20

########## ściąganie ##############
# spowolnij ściąganie do wartości trochę mniejszej niż prawdziwa prędkość
# połączenia by zapobiec kolejkowaniu u dostawcy internetowego.
# Poeksperymentuj by dopasować się do maksymalnej wartości która jest
# akceptowalna. Dostawcy mają zwykle potężne kolejki by upewnić się, że
# duże transfery odbywają się szybko
#
# dołączamy określanie polityki dla ruchu przychodzącego

tc qdisc add dev $DEV handle ffff: ingress

# filtrujemy do niego wszystko (0.0.0.0/0) i odrzucamy wszystko co
# przychodzi zbyt szybko:
```

```
tc filter add dev $DEV parent ffff: protocol ip prio 50 u32 match ip src \
   0.0.0.0/0 police rate ${DOWNLINK}kbit burst 10k drop flowid :1
```

Jeśli chcesz by ten skrypt uruchamiany był po połączeniu ppp, skopiuj go do `/etc/ppp/ip-up.d`.

Jeśli ostatnie dwie linijki powodują błąd, uaktualnij swoje narzędzie **tc** do nowszej wersji!

Skrypt (HTB)

Poniższy skrypt osiąga cele przy użyciu wspaniałej kolejki HTB, o której możesz poczytać więcej wyżej. Warto pomęczyć się z łataniem kernela!

```
#!/bin/bash

# Konfiguracja dla połączenia internetowego w domu
#
#
# Ustaw poniższe wartości trochę poniżej faktycznych prędkości
# ściągania i wysyłania (w kilobitach)
DOWNLINK=800
UPLINK=220
DEV=ppp0

# wyczyść kolejki dla wysyłania i ściągania danych, nie informuj o błędach
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev $DEV ingress 2> /dev/null > /dev/null

###### wysyłanie danych

# zainstaluj w korzeniu kolejkę HTB, skieruj domyślnie ruch do 1:20:

tc qdisc add dev $DEV root handle 1: htb default 20

# kształtujemy wszystko na prędkości $UPLINK - zapobiega to tworzeniu
# się dużych kolejek na modemie DSL co zniszczyłoby panowanie nad
# opóźnieniami

tc class add dev $DEV parent 1: classid 1:1 htb rate ${UPLINK}kbit burst 6k

# klasa z dużym priorytetem 1:10:

tc class add dev $DEV parent 1:1 classid 1:10 htb rate ${UPLINK}kbit \
   burst 6k prio 1

# klasa domyślna dla ruchu ‘masowego’ 1:20 - otrzymuje trochę mniej
# ruchu i ma mniejszy priorytet

tc class add dev $DEV parent 1:1 classid 1:20 htb rate $[9*$UPLINK/10]kbit \
   burst 6k prio 2

# obie klasy kontrolowane są przez SFQ:
tc qdisc add dev $DEV parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev $DEV parent 1:20 handle 20: sfq perturb 10

# Minimalna zwłoka w polu ToS (ssh, NIE scp) dla 1:10:
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
```

```
      match ip tos 0x10 0xff  flowid 1:10

# ICMP (protokół IP numer 1) w klasie interaktywnej 1:10 tak byśmy
# mogli wykonywać pomiary i pochwalić się naszym przyjaciołom
tc filter add dev $DEV parent 1:0 protocol ip prio 10 u32 \
       match ip protocol 1 0xff flowid 1:10

# By zwiększyć prędkość ściągania danych w trakcie wysyłania, pakiety
# ACK wsadzamy do klasy interaktywnej

tc filter add dev $DEV parent 1: protocol ip prio 10 u32 \
   match ip protocol 6 0xff \
   match u8 0x05 0x0f at 0 \
   match u16 0x0000 0xffc0 at 2 \
   match u8 0x10 0xff at 33 \
   flowid 1:10

# cała 'nie-interaktywna' reszta trafia do 1:20


########## ściąganie ##############
# spowolnij ściąganie do wartości trochę mniejszej niż prawdziwa prędkość
# połączenia by zapobiec kolejkowaniu u dostawcy internetowego. Poeksperymentuj
# by dopasować się do maksymalnej wartości która jest akceptowalna.
# Dostawcy mają zwykle potężne kolejki by upewnić się, że duże transfery
# odbywają się szybko
#
# dołączamy określanie polityki dla ruchu przychodzącego

tc qdisc add dev $DEV handle ffff: ingress

# filtrujemy do niego wszystko (0.0.0.0/0) i odrzucamy wszystko co
# przychodzi zbyt szybko:

tc filter add dev $DEV parent ffff: protocol ip prio 50 u32 match ip src \
   0.0.0.0/0 police rate ${DOWNLINK}kbit burst 10k drop flowid :1
```

Jeśli chcesz by ten skrypt uruchamiany był po połączeniu ppp, skopiuj go do `/etc/ppp/ip-up.d`.

Jeśli ostatnie dwie linijki powodują błąd, uaktualnij swoje narzędzie **tc** do nowszej wersji!

Ograniczanie ruchu dla pojedyńczego hosta lub podsieci

O ile opisano to w wielu miejscach na stronach podręcznika, kwestia ta jest bardzo często podnoszona. Rozwiązanie tego problemu nie wymaga wcale pełnego zrozumienia dla idei kontroli ruchu.

Poniższe trzy komendy rozwiązują problem:

```
tc qdisc add dev $DEV root handle 1: cbq avpkt 1000 bandwidth 10mbit

tc class add dev $DEV parent 1: classid 1:1 cbq rate 512kbit \
allot 1500 prio 5 bounded isolated
```

```
tc filter add dev $DEV parent 1: protocol ip prio 16 u32 \
match ip dst 195.96.96.97 flowid 1:1
```

Pierwsza linijka instaluje dyscyplinę kolejkowania z klasami na interfejsie. Dla kernela jest to interfejs o przepustowości 10mbit/s. Jeśli ustawisz prędkość niedokładnie, nie stanie się wielka krzywda, ale lepiej ustawić to dokładnie.

Druga linijka konfiguruje 512kbitową klasę z pewnymi sensownymi domyślnymi wartościami. Detale znajdziesz na stronach podręcznikowych i tu: Rozdział 9.

Ostatnia linijka okresla, gdzie konfigurowany ruch powinien trafiać. Ruch, który nie pasuje do tej reguły NIE będzie kontrolowany i kształtowany. By wykonać bardziej skomplikowane testy (podsieci, porty źródłowe czy docelowe) zajrzyj do sekcja *Wszystkie komendy filtrujące, których będziesz normalnie potrzebował* w Rozdział 9.

Jeśli cokolwiek zmienisz i chcesz przeładować skrypt, wykonaj polecenie **tc qdisc del dev $DEV root** by wyczyścić aktualną konfigurację.

Skrypt można dalej ulepszyć dodając ostatnią linijkę w brzmieniu `tc qdisc add dev $DEV parent 1:1 sfq perturb 10`. Zajrzyj do sekcja *Sprawiedliwe Kolejkowanie Stochastyczne (ang. „Stochastic Fairness Queueing”)* w Rozdział 9 po więcej informacji co ona robi i jak wpływa na działanie kompletu.

Przykład rozwiązania z QoS i NATem

Nazywam się Pedro Larroy

<piotr%member.fsf.org>

. Opiszę dosyc powszechną konfigurację, w której mamy bardzo dużo użytkowników w sieci prywatnej połączonych do Internetu przez router z Linuksem (i publicznym adresem IP), który wykonuje NAT. Sam używam takiej konfiguracji w topologii ze 198 użytkownikami na uniwersytecie - gdzie sam żyję i jednocześnie jestem administratorem sieci. Użytkownicy jak to studenci używają bardzo dużej ilości oprogramowania typu peer-to-peer, więc prawidłowa konfiguracja jakości usług jest praktycznie koniecznością. Mam nadzieję że przyda się to jako praktyczny przykład dla wszystkich dociekliwych czytelników tego HOWTO.

Rozpoczniemy od praktycznego podejścia krok po kroku do konfiguracji, a później wyjaśnię jak zrobić to wszystko automatycznie podczas startu maszyny. Sieć, która jest przedmiotem konfiguracji to prywatny LAN połączony do Internetu przez router Linuksowy z pojedyńczym publicznym adresem IP. Rozszerzenie puli adresów publicznych nie stanowi problemu - należy tylko dodać parę poleceń do **iptables**. Aby wszystko poprawnie działało, potrzebujemy:

Linuksa 2.4.18 lub z wyższą wersją

> Jeśli zastosujesz dokładnie wersję 2.4.18 będziesz musiał zastosować łatkę dla HTB.

iproute

> Upewnij się również że masz wersję narzędzia **tc** zgodną z HTB, prekompilowana binarna wersja rozprowadzana jest z HTB.

iptables

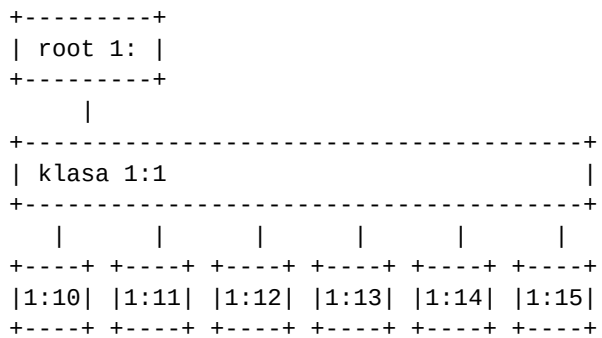
Zoptymalizujmy cenne pasmo

Na początek stworzymy trochę qdisc do których klasyfikować będziemy ruch. Tworzymy qdisc HTB z 6 klasami z malejącymi priorytetami. Mamy klasy które zawsze będą przynajmniej wypełniały dostępne pasmo, ale mogą używać pasmo innych klas jeśli one go nie wykorzystują. Przypominam, że klasy z wyższym priorytetem (z niższym numerem priorytetu) dostają pierwsze dodatkowe pasmo. Fizycznie połączenie stanowi linia ADSL 2Mbit/s downstream na 300kbit/s upstream. Używam prędkości 240kbit/s jako górnej wartości ponieważ wyższe ustawienie powoduje wzrost opóźnień w związku z buforowaniem pomiędzy nami a zdalnymi hostami. Parametr ten należy oczywiście dobrać eksperymentalnie, podnosząc go i opuszczając obserwując uważnie wartości opóźnień.

Zmień wartość CEIL na 75% twojej przepustowości upstream na początek i tam gdzie ja używam eth0 ty wstaw swój interfejs publiczny. Na początek zaczniemy wpisując w powłoce roota:

```
CEIL=240
tc qdisc add dev eth0 root handle 1: htb default 15
tc class add dev eth0 parent 1: classid 1:1 htb rate ${CEIL}kbit \
 ceil ${CEIL}kbit
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 80kbit \
 ceil 80kbit prio 0
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 80kbit \
 ceil ${CEIL}kbit prio 1
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 20kbit \
 ceil ${CEIL}kbit prio 2
tc class add dev eth0 parent 1:1 classid 1:13 htb rate 20kbit \
 ceil ${CEIL}kbit prio 2
tc class add dev eth0 parent 1:1 classid 1:14 htb rate 10kbit \
 ceil ${CEIL}kbit prio 3
tc class add dev eth0 parent 1:1 classid 1:15 htb rate 30kbit \
 ceil ${CEIL}kbit prio 3
tc qdisc add dev eth0 parent 1:12 handle 120: sfq perturb 10
tc qdisc add dev eth0 parent 1:13 handle 130: sfq perturb 10
tc qdisc add dev eth0 parent 1:14 handle 140: sfq perturb 10
tc qdisc add dev eth0 parent 1:15 handle 150: sfq perturb 10
```

Stworzyliśmy tym samym drzewko HTB głębokie na jeden poziom. Coś wyglądającego mniej więcej tak:



classid 1:10 htb rate 80kbit ceil 80kbit prio 0

Ta linijka odpowiada za klasę z najwyższym priorytetem. Pakiety w tej klasie będą miały najmniejsze opóźnienie i otrzymają dostęp do nadmiarowej przepustowości pierwsze - dobrym pomysłem jest w związku z tym ograniczenie jej odgórnie. Do tej klasy wysyłać będziemy następujące rodzaje pakietów: *ssh, telnet, dns, quake3, irc, i pakiety z ustawioną flagą SYN*.

classid 1:11 htb rate 80kbit ceil ${CEIL}kbit prio 1

Ta linijka odpowiada za ruch ‘masowy’. W naszym przykładzie jest to ruch z lokalnego serwera WWW i zapytania WWW - odpowiednio ze źródłowym portem 80/tcp i z docelowym portem 80/tcp.

classid 1:12 htb rate 20kbit ceil ${CEIL}kbit prio 2

W tej klasie umieszczam ruch z ustawionym bitem ToS na ‘Zmaksymalizuj przepustowość’ i całą resztę ruchu pochodzącą z *procesów lokalnych* routera.

classid 1:13 htb rate 20kbit ceil ${CEIL}kbit prio 2

W tej klasie mamy ruch z NATowanych maszyn które wymagają większego priorytetu niż ruch ‘masowy’.

classid 1:14 htb rate 10kbit ceil ${CEIL}kbit prio 3

Tutaj mamy ruch pocztowy (SMTP, POP3, ...) oraz pakiety z ustawionym bitem ‘Minimalizuj koszt’ w polu ToS.

classid 1:15 htb rate 30kbit ceil ${CEIL}kbit prio 3

Na koniec, ruch ‘masowy’ z NATowanych maszyn - czyli aplikacje typu eDonkey, Kazaa i tym podobne. Dzięki temu nie przeszkadzają innym usługom.

Klasyfikacja pakietów

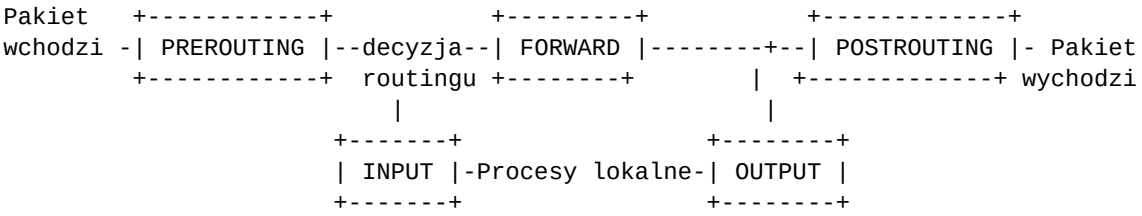
Stworzyliśmy już konfigurację samych qdisc, ale nie wykonujemy jeszcze żadnej klasyfikacji - wszystkie pakiety trafiają w tym momencie do klasy 1:15 (ponieważ użyliśmy `tc qdisc add dev eth0 root handle 1: htb default 15`). Musimy teraz określić, które pakiety mają trafić do której klasy. W zasadzie to najważniejsza część konfiguracji.

Skonfigurujemy teraz filtry, za pomocą których **iptables** sklasyfikują nam ruch. Naprawdę wolę to robić za pomocą **iptables**, ponieważ są bardzo elastyczne i dodatkowo udostępniają ilość trafień dla każdej reguły. Co więcej, cel RETURN powoduje, że nie trzeba przeglądać wszystkich reguł. Wykonujemy co następuje:

```
tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 1 fw classid 1:10
tc filter add dev eth0 parent 1:0 protocol ip prio 2 handle 2 fw classid 1:11
tc filter add dev eth0 parent 1:0 protocol ip prio 3 handle 3 fw classid 1:12
tc filter add dev eth0 parent 1:0 protocol ip prio 4 handle 4 fw classid 1:13
tc filter add dev eth0 parent 1:0 protocol ip prio 5 handle 5 fw classid 1:14
tc filter add dev eth0 parent 1:0 protocol ip prio 6 handle 6 fw classid 1:15
```

Powyższe polecenia określają, że jądro ma kierować pakiety oznaczone określoną wartością FWMARK (*handle X fw*) do właściwej klasy (*classid X:Y*). Teraz przyjrzymy się jak oznaczać pakiety przy pomocy **iptables**.

Na początek, powinieneś zrozumieć jak pakiety podróżują przez filtry w **iptables**:



Zakładał, że wszystkie tabele stworzone zostały z domyślną polityką akceptującą (*ACCEPT* - `-P ACCEPT`). Jeśli nic nie zmieniałeś, powinno tak być domyślnie. Nasza sieć publiczna to klasa B 172.17.0.0/16, a adresem publicznym jest 212.170.21.172.

Następnie każemy jądru *faktycznie wykonywać NAT*, tak by klienci z sieci prywatnej mogli nawiązywać połączenia na zewnątrz:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -t nat -A POSTROUTING -s 172.17.0.0/255.255.0.0 -o eth0 -j SNAT \
  --to-source 212.170.21.172
```

Możesz teraz sprawdzić, że pakiety faktycznie trafiają na razie do 1:15:

```
tc -s class show dev eth0
```

Zaczniemy teraz znaczyć pakiety dodając reguły do łańcucha PREROUTING w tablicy przekształceń:

```
iptables -t mangle -A PREROUTING -p icmp -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -p icmp -j RETURN
```

Powinieneś teraz zobaczyć pakiety ICMP trafiające do klasy 1:10 i zwiększające się liczniki przy ruchu z sieci prywatnej do publicznej.

```
tc -s class show dev eth0
```

Ponieważ wskazaliśmy cel `-j RETURN`, więc pakiety nie muszą przechodzić przez wszystkie reguły - ICMP nie będą sprawdzane poniżej tej linijki. Dodajemy kolejne regułki, dotyczące oznaczania ToS:

```
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay \
  -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Delay \
  -j RETURN
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Cost \
  -j MARK --set-mark 0x5
iptables -t mangle -A PREROUTING -m tos --tos Minimize-Cost \
  -j RETURN
iptables -t mangle -A PREROUTING -m tos --tos Maximize-Throughput \
  -j MARK --set-mark 0x6
iptables -t mangle -A PREROUTING -m tos --tos Maximize-Throughput \
  -j RETURN
```

Prioretyzujemy pakiety SSH:

```
iptables -t mangle -A PREROUTING -p tcp -m tcp --sport 22 \
  -j MARK --set-mark 0x1
iptables -t mangle -A PREROUTING -p tcp -m tcp --sport 22 \
  -j RETURN
```

Dobrym pomysłem jest w tym momencie priorytetyzacja początków połączeń TCP, czyli tych z ustawioną tylko flagą SYN:

```
iptables -t mangle -I PREROUTING -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN \
  -j MARK --set-mark 0x1
iptables -t mangle -I PREROUTING -p tcp -m tcp --tcp-flags SYN,RST,ACK SYN \
  -j RETURN
```

I tak dalej. Po skończeniu dodawania reguł, ostatnim wpisem w tej tabeli powinno być:

```
iptables -t mangle -A PREROUTING -j MARK --set-mark 0x6
```

Tak więc poprzednio nieoznaczany ruch trafia do 1:15. Tak naprawdę jest to niepotrzebne (ponieważ i tak ruch domyślnie trafia do 1:15), ale robimy to dla jasności i przejrzystości. Co więcej, dzięki temu będziemy mogli zbierać statystyki trafień w tą regułę.

Kolejnym dobrym pomysłem będzie wykonanie tego samego dla reguł w łańcuchu OUTPUT, powtórz więc wszystkie polecenia podając `-A OUTPUT` zamiast `PREROUTING` (`s/PREROUTING/OUTPUT/`). Dzięki temu ruch generowany lokalnie a nie tylko routowany zostanie również sklasyfikowany. Osobiście kończę łańcuch OUTPUT regułą `-j MARK --set-mark 0x3` tak, by cała reszta ruchu generowanego lokalnie miała większy priorytet.

Improving our setup

Powinniśmy mieć teraz działającą konfigurację. Spędź trochę czasu analizując wykresy i sprawdzając jak dokładnie wykorzystywana jest dostępna przepustowość. Sam spędziłem nad tym wiele godzin i doprowadziłem swoje połączenie do idealnego stanu - jeśli coś ustawisz źle, zaczną pojawiać się błędy wygasania połączeń lub nowo tworzone połączenia TCP nie będa mogły dojść do skutku.

Jeśli ustalisz, że niektóre klasy cały czas są pełne dobrze jest spróbować dołączyć dodatkową qdisc tak by podział pasma był bardziej sprawiedliwy. Na przykład w ten sposób:

```
tc qdisc add dev eth0 parent 1:13 handle 130: sfq perturb 10
tc qdisc add dev eth0 parent 1:14 handle 140: sfq perturb 10
tc qdisc add dev eth0 parent 1:15 handle 150: sfq perturb 10
```

Niech to wszystko dzieje się samo przy starcie

Możemy to osiągnąć na wiele sposobów. Napisałem skrypt w powłoce, `/etc/init.d/packetfilter`, który akceptuje składnię `[start | stop | stop-tables | start-tables | reload-tables]`. Konfiguruje on wszystkie qdisc i ładuje potrzebne moduły jądra - zachowując się jak demon. Ten sam skrypt ładuje reguły dla **iptables** z osobnego pliku `/etc/network/iptables-rules`. Trochę go ulepszę i udostępnie pod tym adresem (http://omega.resa.es/piotr/files/packetfilter.tar.bz2)

Rozdział 16. Budowanie mostów i pseudo-mostów z Proxy ARP

Mosty to urządzenia możliwe do zainstalowania w sieci bez żadnych rekonfiguracji. Przełącznik sieciowy to zwykle wielo-portowy most. Most to z kolei zwykle 2-portowy przełącznik. Linuks wspiera wiele interfejsów dla mostu co w połączeniu z jego funkcjonalnością czyni go prawdziwym przełącznikiem.

Mosty zwykle umieszcza się tam, gdzie trzeba poprawiać błędy konstrukcyjne sieci, bez wprowadzania żadnych poważniejszych zmian. Ponieważ most jest urządzeniem pracującym w oparciu o warstwę drugą - warstwę pod IP, routery i serwery nie zdają sobie sprawy z jego obecności. Oznacza to, że możesz w sposób transparentny modyfikować i blokować określone pakiety, czy też w jakiś sposób kształtować ruch.

Kolejną zaletą jest to, że most może być zwykle zastąpiony przez kabel krosujący lub koncentrator jeśli zostanie uszkodzony.

Złe wieści są takie, że most potrafi narobić masę bałaganu jeśli jego funkcje nie są dobrze udokumentowane. Nie pojawia się w wydrukach traceroute ale w jakiś magiczny sposób pakiety znikają gdzieś po drodze, lub są zmieniane na drodze z punktu A do B (‘ta sieć jest NAWIEDZONA!’). Powinieneś również zastanowić się czy firma która ‘nie chce nic zmieniać’ postępuje słusznie.

Most dla Linuksa 2.4/2.5 udokumentowano pod tym adresem (http://bridge.sourceforge.net/).

Stan mostkowania i iptables

Od Linuksa 2.4.14 procesy mostkowania i **iptables** nie ‘widzą’ się wzajemnie bez specjalnych zabiegów. Jeśli prowadzisz mostkowanie pakietów z eth0 do eth1, nie przechodzą one przez **iptables**. Oznacza to, że nie możesz filtrować, prowadzić NATu czy manipulować pakietami. W Linuksie 2.5.45 ma to zostać poprawione.

Możesz spotkać się z nazwą ‘ebtables’ - to osobny projekt, który umożliwia wykonywanie najdzikszych rzeczy takich jak MACNAT i ‘brouting’. Naprawdę przerażające ;)

Mostkowanie i kształtowanie ruchu

Działa dokładnie tak jak się to opisuje. Upewnij się, że wiesz w którą stronę skierowany jest każdy interfejs. Jeśli się pomylisz, możesz zacząć kontrolować ruch wychodzący na interfejsie wewnętrznym co może nie dać spodziewanych rezultatów. Jeśli potrzebujesz, użyj tcpdumpa.

Pseudo-mosty i Proxy-ARP

Jeśli chcesz zaimplementować pseudo-most, pomiń tych parę sekcji aż do ‘Uruchamianie’, dobrze byłoby jednak byś przeczytał trochę jak to działa w praktyce.

Pseudo-most działa trochę inaczej. Domyślnie, most przesyła pakiety niezmienione z jednego interfejsu do innego. Sprawdza tylko adresy sprzętowe pakietów by określić, gdzie co wysłać. Oznacza to, że możesz mostkować ruch, którego Linuks nie jest w stanie zrozumieć pod warunkiem że posiada on adresy sprzętowe.

Pseudo-most zachowuje się bardziej jak ukryty router niż most, ale podobnie jak most nie ma wpływu na budowę sieci.

Zaletą faktu, że nie jest mostem jest to, że pakiety przechodzą przez kernel, mogą zatem być filtrowane, zmieniane, przekierowywane i przetrasowywane.

Prawdziwy most może również wykonywać takie zadania, ale potrzebuje specjalnego kodu, czegoś w rodzaju Przekierowywacza Ramek Ethernetowych lub czegoś na wzór wspomnianej łaty.

Inną zaletą pseudo-mostu jest to, że nie przekazuje pakietów których nie potrafi zrozumieć - oczyszcza w ten sposób sieć ze śmieci. W przypadku gdybyś jednak potrzebował tych ‘śmieci’ (czegoś w rodzaju pakietów SAP czy Netbeui) musisz użyć prawdziwego mostu.

ARP & Proxy-ARP

Kiedy komputer chce rozmawiać z innym w tym samym segmencie sieci fizycznej, wysyła pakiet ARP brzmiący w uproszczonej wersji tak: ‘kto ma 10.0.0.1 niech powie 10.0.0.7’. W odpowiedzi na taki pakiet, 10.0.0.1 odpowiada krótkim ‘ja mam’.

10.0.0.7 wysyła pakiet z adresem sprzętowym wymienionym w pakiecie ‘ja mam’. Zapisuje również ten adres na relatywnie długi czas a gdy on upłynie, ponownie rozsyła takie zapytanie.

Podczas budowy pseudo-mostu, instruujemy most by odpowiadał na takie pakiety ARP odsyłając swój adres, co spowoduje, że komputery będą wysyłały swoje pakiety przez niego. Most przetwarza otrzymane w ten sposób pakiety i rozsyła je do odpowiednich interfejsów.

Krótko mówiąc, kiedykolwiek komputer po jednej stronie mostu zapyta o adres sprzętowy komputera po drugiej stronie, most odpowiada pakietem ‘przekaż go mnie’.

W ten sposób, cały ruch przesyłany jest do prawidłowej lokalizacji i przechodzi przez most.

Skonfigurujmy to!

W zamierzchłych czasach można było rozkazać kernelowi Linuksa by prowadził ‘proxy-ARP’ dla każdej podsieci. By skonfigurować pseudo-most, musiałeś podać zarówno odpowiednie trasy do obu stron mostu jak i stworzyć reguły pasujące dla proxy-ARP. Było to o tyle niewygodne, że wymagało wiele pisania, co przekładało się wprost na ilość możliwych do popełnienia błędów powodujących wzbudzanie się mostu do odpowiedzi ARP dla sieci, do których nie potrafił przetrasować pakietów.

Wraz z nadejściem Linuksa 2.4/2.5 (być może również 2.2) wycofano się z tego rozwiązania i zastąpiono je flagą w katalogu `/proc` nazwaną `proxy_arp`. Procedura na zbudowanie pseudo-mostu wygląda teraz tak:

1. Przydziel adresy IP obu interfejsom, ‘lewy’ i ‘prawy’
2. Stwórz trasy tak, by twój komputer wiedział jakie komputery są po lewej a jakie po prawej stronie
3. Włącz proxy-ARP na obu interfejsach:

   ```
   echo 1 > /proc/sys/net/ipv4/conf/ethL/proxy_arp
   echo 1 > /proc/sys/net/ipv4/conf/ethP/proxy_arp
   ```

   gdzie ‘L’ i ‘P’ oznaczają interfejsy Lewy i Prawy

Nie zapomnij również ustawić flagi `ip_forwarding`! Podczas konwersji z prawdziwego mostu możesz stwierdzić że flaga była zgaszona ponieważ nie jest potrzebna podczas mostkowania.

Inna rzecz, którą możesz zauważyć to konieczność wyczyszczenia tablic podręcznych adresów ARP komputerów w sieci - może ona zawierać stare, pochodzące sprzed konfiguracji pseudo-mostu adresy sprzętowe, które nie są już aktualne.

Na sprzęcie firmy Cisco robi się to komendą `clear arp-cache`, pod Linuksem `arp -d adres_ip`. Możesz oczywiście również poczekać na to, by czas ich ważności wygasł sam, ale może to trochę zająć.

Możesz również przyśpieszyć ten proces używając narzędzia **arping**, które w wielu dystrybucjach znajduje się w pakiecie **iputils**. Za jego pomocą możesz wysłać samorzutnie rozgłoszenia ARP, nadpisując zdalne wpisy w pamięci podręcznej ARP.

Narzędzie to jest bardzo potężne, używane często przez ‘czarne kapelusze’ do przekonfigurowywania twojego routingu!

> **Notatka:** Na Linuksie 2.4 możesz zostać zmuszony do wykonania polecenia `echo 1 > /proc/sys/net/ipv4/ip_nonlocal_bind` zanim będziesz mógł wysłać takie ‘podrobione’ pakiety!

Możesz również zauważyć, że twoja sieć jest źle skonfigurowana jeśli masz lub miałeś zwyczaj stosować trasy bez masek sieciowych. Dokładniej, niektóre wersje programu ‘route’ poprawnie odgadywały maski a inne źle - bez powiadamiania cię o tym fakcie. Podczas wykonywania naprawdę chirurgicznego trasowania jak opisano powyżej, *bardzo* ważne jest sprawdzenie poprawności masek!

Rozdział 17. Routing dynamiczny - OSPF i BGP

Gdy twoja sieć rozrośnie się poważnie lub gdy zaczynasz postrzegać ‘internet’ jako swoją sieć, potrzebujesz narzędzi dynamicznie trasujących dane. Lokalizacje połączone są ze sobą zwykle wieloma łączami a ich liczba stale rośnie.

Internet ustandaryzował przede wszystkim OSPF (RFC 2328) i BGP4 (RFC 1771). Linuks wspiera oba protokoły za pomocą programów gated oraz zebra.

O ile nie dotyczy to obecnie tego dokumentu, chciałbym skierować cię do pewnych dobrych źródeł:

Ogólnie:

Cisco Systems Designing large-scale IP Internetworks (http://www.cisco.com/univercd/cc/td/doc/cisintwk/idg4/nd2003.htm)

Dla OSPF:

Moy, John T. "OSPF. The anatomy of an Internet routing protocol" Addison Wesley. Reading, MA. 1998.

Halabi napisał również dobry przewodnik o projektowaniu rutingu OSPF ale najwyraźniej został on usunięty ze stron WWW Cisco.

Dla BGP:

Halabi, Bassam "Internet routing architectures" Cisco Press (New Riders Publishing). Indianapolis, IN. 1997.

oraz

Cisco Systems Using the Border Gateway Protocol for interdomain routing (http://www.cisco.com/univercd/cc/td/doc/cisintwk/ics/icsbgp4.htm)

O ile przykłady są specyficzne dla Cisco, są bardzo podobne do języka konfiguracji programu Zebra :-).

Setting up OSPF with Zebra

Proszę, daj mi (mailto:piotr%member.fsf.org) znać jeśli znajdziesz błąd lub uznasz, że jakieś informacje są niedokładne. Chętnie przyjmę wszelkie sugestie.

Zebra (http://www.zebra.org) to świetne oprogramowanie zajmujące się dynamicznym routingiem, napisane przez Kunihiro Ishiguro, Toshiaki Takada i Yasuhiro Ohara. Za pomocą Zebry mozna szybko i prosto uruchomić routing OSPF, ale w praktyce jest do dyspozycji tyle parametrów że jeśli tylko masz jakieś specyficzne potrzeby prawdopodobnie da się do nich dostosować. OSPF oznacza Open Shortest Path First (w wolnym tłumaczeniu „Użyj Najpierw Najkrótszą Ścieżkę”) a podstawowymi cechami tego protokołu są:

Hierarchiczność

> Sieci grupowane są w *obszary* (ang. „areas”) połączone centralnym *obszarem szkieletu* (ang. „backbone area”) który oznaczany jest jako *area 0*. Cały ruch przechodzi przez ten zerowy obszar i wszystkie routery w nim się znajdujące mają informacje o routingu do wszystkich innych obszarów.

Krótka konwergencja

> Trasy propagowane są bardzo szybko w porównaniu np. z protokołem RIP.

Oszczędny dla pasma

> Używa multicastów zamiast rozgłoszeń, więc nie zalewa innych hostów informacją o routingu która zwykle przecież w ogóle jest im niepotrzebna. Co więcej, *routery wewnętrzne* (te, które mają wszystkie interfejsy tylko w jednym obszarze) nie przetwarzają ani nie posiadają informacji o routingu w innych obszarach.

Routery które mają interfejsy w więcej niż jednym obszarze nazywane są *routerami brzegowymi obszaru* (ang. „Area Border Routers”, ABR) i przechowują informacje o topologii obszarów, do których są bezpośrednio podłączone.

Obciąża CPU

OSPF bazuje na algorytmie najkrótszej ścieżki Dijkstry (http://www.soi.wide.ad.jp/class/99007/slides/13/07.html), który jest dosyć kosztowny w porównaniu do innych algorytmów routingu. Tak naprawdę nie jest tak źle, ponieważ najkrótsze ścieżki obliczane są tylko w obrębie obszarów i dla małych oraz średnich sieci w ogóle nie będzie to problemem.

Protokół typu łącze-stan

OSPF zlicza wiele charakterystyk sieci i interfejsów tworząc metrykę do danej lokalizacji. Można tu wymienić takie parametry jak przepustowość, ilość awarii łącza czy koszt.

Protokół otwarty i oprogramowanie na licencji GPL

OSPF jest protokołem otwartym, a Zebra tworzona jest na licencji GPL co ma oczywiste znaczenie w porównaniu do firmowych protokołów i oprogramowania.

Założenia

Jądro Linuksa:

Skompilowane z opcjami ‘CONFIG_NETLINK_DEV’ i ‘CONFIG_IP_MULTICAST’ (nie jestem pewien czy nie potrzeba czegoś jeszcze).

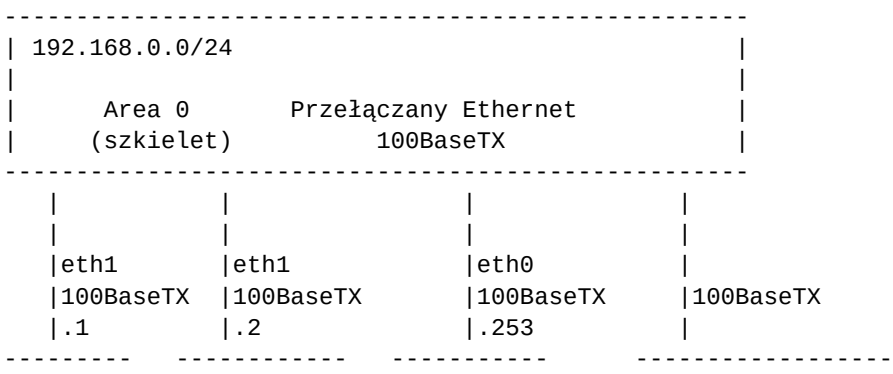
iproute

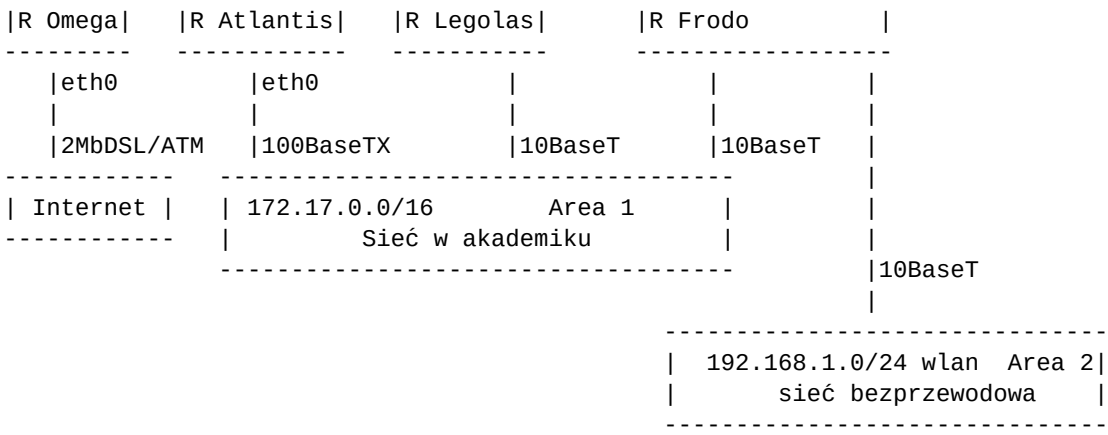
Zebra

Ściągnij przy użyciu ulubionego menadżera paczek lub po prostu spod tego adresu (http://www.zebra.org).

Konfiguracja Zebry

Zajmijmy się taką siecią:





Nie przejmuj się tym rysunkiem, Zebra pracuje praktycznie automatycznie, więc nie będzie to wymagało wiele pracy. Wpisanie i utrzymanie tych zaledwie kilku wpisów w statycznej tabeli routingu byłoby nieco bolesne. Najważniejsze w tym momencie to posiadanie jasnego obrazu topologii sieci. I oczywiście uważaj na obszar zerowy, ponieważ jest najważniejszy.

Najpierw skonfigurujemy zebrę, dostosowując plik `zebra.conf` do naszych potrzeb:

```
hostname omega
password xxx
enable password xxx
!
! Opis interfejsów
!
!interface lo
! description test opisu
!
interface eth1
multicast
!
! Statyczna trasa domyślna
!
ip route 0.0.0.0/0 212.170.21.129
!
log file /var/log/zebra/zebra.log
```

Na Debianie musiałem również wyedytować plik `/etc/zebra/daemons` tak by Zebra startowała przy starcie maszyny:

```
zebra=yes
ospfd=yes
```

Teraz zajmiemy się edycją pliku `ospfd.conf` jeśli nadal korzystasz z IPv4 lub pliku `ospf6d.conf` jeśli przeszedłeś już na IPv6. Mój `ospfd.conf` wygląda tak:

```
hostname omega
password xxx
enable password xxx
!
router ospf
 network 192.168.0.0/24 area 0
 network 172.17.0.0/16 area 1
!
```

```
! log stdout
log file /var/log/zebra/ospfd.log
```

Konfigurujemy tutaj topologię sieci tak, jak chcemy by demon OSPF ją widział.

Uruchomienie Zebry

Wystartujemy teraz Zebrę; albo bezpośrednio z powłoki pisząc ‘`zebra -d`’, albo za pomocą skryptu wykonując polecenie `/etc/init.d/zebra start`. Następnie obserwuj uważnie logi procesu **ospdfd**:

```
2002/12/13 22:46:24 OSPF: interface 192.168.0.1 join AllSPFRouters Multicast group.
2002/12/13 22:46:34 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:46:44 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:46:54 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:47:04 OSPF: SMUX_CLOSE with reason: 5
2002/12/13 22:47:04 OSPF: DR-Election[1st]: Backup 192.168.0.1
2002/12/13 22:47:04 OSPF: DR-Election[1st]: DR     192.168.0.1
2002/12/13 22:47:04 OSPF: DR-Election[2nd]: Backup 0.0.0.0
2002/12/13 22:47:04 OSPF: DR-Election[2nd]: DR     192.168.0.1
2002/12/13 22:47:04 OSPF: interface 192.168.0.1 join AllDRouters Multicast group.
2002/12/13 22:47:06 OSPF: DR-Election[1st]: Backup 192.168.0.2
2002/12/13 22:47:06 OSPF: DR-Election[1st]: DR     192.168.0.1
2002/12/13 22:47:06 OSPF: Packet[DD]: Negotiation done (Slave).
2002/12/13 22:47:06 OSPF: nsm_change_status(): scheduling new router-LSA origination
2002/12/13 22:47:11 OSPF: ospf_intra_add_router: Start
```

Na razie zignoruj wiadomość SMUX_CLOSE w której chodzi o obsługę SNMP. Widzimy że 192.168.0.1 zostaje *wyznaczonym routerem* (ang. „Designated Router”) a 192.168.0.2 zostaje *zapasowym wyznaczonym routerem* (ang. „Backup Designated Router”).

Możemy oczywiście połączyć się z interfejsem aplikacji **zebra** i **ospfd** wykonując:

```
$ telnet localhost zebra
lub
$ telnet localhost ospfd
```

Przyjrzyjmy się propagowanym trasom. Zaloguj się do demona **zebra** i wykonaj:

```
root@atlantis:~# telnet localhost zebra
Trying 127.0.0.1...
Connected to atlantis.
Escape character is '^]'.

Hello, this is zebra (version 0.92a).
Copyright 1996-2001 Kunihiro Ishiguro.

User Access Verification

Password:
atlantis> show ip route
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

K>* 0.0.0.0/0 via 192.168.0.1, eth1
C>* 127.0.0.0/8 is directly connected, lo
O   172.17.0.0/16 [110/10] is directly connected, eth0, 06:21:53
```

```
C>* 172.17.0.0/16 is directly connected, eth0
O   192.168.0.0/24 [110/10] is directly connected, eth1, 06:21:53
C>* 192.168.0.0/24 is directly connected, eth1
atlantis> show ip ospf border-routers
============ OSPF router routing table =============
R    192.168.0.253         [10] area: (0.0.0.0), ABR
                            via 192.168.0.253, eth1
                           [10] area: (0.0.0.1), ABR
                            via 172.17.0.2, eth0
```

Możemy to oczywiście zrobić również z systemu dla tego konkretnego routera:

```
root@omega:~# ip route
212.170.21.128/26 dev eth0  proto kernel  scope link  src 212.170.21.172
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.1
172.17.0.0/16 via 192.168.0.2 dev eth1  proto zebra  metric 20
default via 212.170.21.129 dev eth0  proto zebra
root@omega:~#
```

Widzimy trasę której wcześniej nie było, wpisaną przez zebrę. Miło i wygodnie jest widzieć kolejne trasy pojawiające się w ciągu sekund po restarcie obu demonów. Możesz oczywiście sprawdzić łączność z poszczególnymi podsieciami za pomocą chociażby narzędzia **ping**. Zebra automatycznie wprowadzi do tablicy routingu systemu nowe trasy - dodajesz po prostu kolejne sieci i uruchamiasz w nich kolejny proces **zebra**.

Podpowiedź: możesz użyć:

```
tcpdump -i eth1 ip[9] == 89
```

By wyłapać pakiety OSPF do dalszej analizy. OSPF posługuje się pakietami IP protokół 89 a pole protokół to 9 oktet w nagłówku IP.

OSPF ma wiele parametrów którymi można dostosować jego pracę, specjalnie jeśli mamy do czynienia z dużą siecią. W przyszłych wersjach tego HOWTO pokażemy zapewne metodologię dostosowywania OSPF.

Konfiguracja BGPv4 w Zebrze

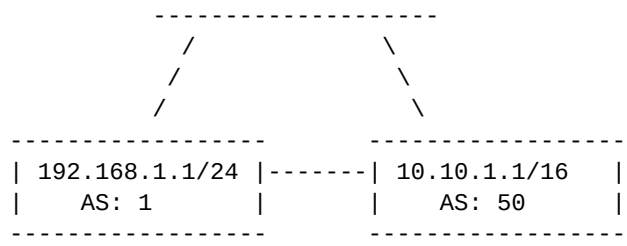
Protokół BGP w wersji 4 („Border Gateway Protocol”) jest protokołem routingu dynamicznego opisanym w RFC 1771. Umożliwia dystrybucję informacji o osiągalności, tj. tabel routingu, do innych węzłów obsługujących BGPv4. Można go używać zarówno jako EGP i IGP. W trybie EGP każdy węzeł musi posiadać swój numer *Systemu Autonomicznego* (ang. „Autonomous System”, AS). BGPv4 wspiera notację CIDR oraz agregacje (połączenie rozgłoszenia wielu tras w jedną sumaryczną).

Topologia przykładowej sieci

Poniższa topologia użyta zostanie do następnych przykładów. AS 1 i 50 mają więcej sąsiadów, ale będziemy konfigurować tylko je. Węzły w tym przykładzie komunikują się za pośrednictwem tuneli, ale w rzeczywistości nie jest to zupełnie potrzebne.

Uwaga: Numery AS użyte w tym przykładzie są zarezerwowane, jeśli chcesz uruchomić BGP w Internecie musisz zwrócić się do RIPE.

```
                    --------------------
                    | 192.168.23.12/24 |
                    |    AS: 23        |
```



Przykładowa konfiguracja

Poniższa konfiguracja została napisana dla węzła 192.168.23.12/24 - względnie prosto jest ją zaadaptować na potrzeby innych węzłów.

Na początek konfigurujemy podstawowe parametry, takie jak nazwa hosta, hasło i ustawienia debugingu:

```
! nazwa hosta
hostname anakin

! hasło do logowania
password xxx

! hasło trybu enable (odpowiednik roota)
enable password xxx

! ścieżka do pliku z logami
log file /var/log/zebra/bgpd.log

! debugging: bardzo szczegółowy (oczywiście można
! później wyłączyć)
debug bgp events
debug bgp filters
debug bgp fsm
debug bgp keepalives
debug bgp updates
```

Następnie definiujemy listy dostępu, które posłużą nam do ograniczenia redystrybucji do tylko podsieci zdefiniowanych w RFC 1918 (prywatnych).

```
! sieci RFC 1918
access-list local_nets permit 192.168.0.0/16
access-list local_nets permit 172.16.0.0/12
access-list local_nets permit 10.0.0.0/8
access-list local_nets deny any
```

Następnym krokiem jest konfiguracja konkretnego AS:

```
! Nasz numer AS
router bgp 23

    ! Adres IP routera którym będzie się identyfikował
    bgp router-id 192.168.23.12

    ! Jakie sieci rozgłaszamy sąsiadom
    network 192.168.23.0/24
```

```
    ! Rozgłaszamy wszystkie sieci podłączone
    ! (bezpośrednio)
    redistribute connected

    ! Rozgłaszamy również trasy z tablicy kernela
    ! (wpisane ręcznie/statycznie)
    redistribute kernel
```

Każdy blok ‘`router bgp`’ zawiera definicję konkretnego AS i listę sąsiadów do których podłączony jest nasz router.

```
    ! (sąsiad o IP 192.168.1.1 to AS 1)
    neighbor 192.168.1.1 remote-as 1
    ! (do tras otrzymywanych od tego sąsiada nałóż
    !  filtr ‘local_nets’)
    neighbor 192.168.1.1 distribute-list local_nets in
    ! (sąsiad o IP 10.10.1.1 to AS 50)
    neighbor 10.10.1.1   remote-as 50
    ! (do tras otrzymywanych od tego sąsiada nałóż
    !  filtr ‘local_nets’)
    neighbor 10.10.1.1   distribute-list local_nets in
```

Sprawdzanie konfiguracji

Uwaga: **vtysh** to multiplekser i służy do połączeń ze wszystkimi aktywnymi interfejsami **Zebry**.

```
anakin# sh ip bgp summary
BGP router identifier 192.168.23.12, local AS number 23
2 BGP AS-PATH entries
0 BGP community entries

Neighbor        V    AS MsgRcvd MsgSent   TblVer  InQ OutQ Up/Down  State/PfxRcd
10.10.0.1       4    50      35      40        0    0    0 00:28:40        1
192.168.1.1     4     1   27574   27644        0    0    0 03:26:04       14

Total number of neighbors 2
anakin#
anakin# sh ip bgp neighbors 10.10.0.1
BGP neighbor is 10.10.0.1, remote AS 50, local AS 23, external link
  BGP version 4, remote router ID 10.10.0.1
  BGP state = Established, up for 00:29:01
  ....
anakin#
```

Sprawdźmy jakie trasy dostaliśmy od sąsiadów:

```
anakin# sh ip ro bgp
Codes: K - kernel route, C - connected, S - static, R - RIP, O - OSPF,
       B - BGP, > - selected route, * - FIB route

B>* 172.16.0.0/14 [20/0] via 192.168.1.1, tun0, 2d10h19m
B>* 172.30.0.0/16 [20/0] via 192.168.1.1, tun0, 10:09:24
B>* 192.168.5.10/32 [20/0] via 192.168.1.1, tun0, 2d10h27m
B>* 192.168.5.26/32 [20/0] via 192.168.1.1, tun0, 10:09:24
B>* 192.168.5.36/32 [20/0] via 192.168.1.1, tun0, 2d10h19m
B>* 192.168.17.0/24 [20/0] via 192.168.1.1, tun0, 3d05h07m
```

```
B>* 192.168.17.1/32 [20/0] via 192.168.1.1, tun0, 3d05h07m
B>* 192.168.32.0/24 [20/0] via 192.168.1.1, tun0, 2d10h27m
anakin#
```

Rozdział 18. Inne możliwości

Ten rozdział stanowi listę projektów, mające coś wspólnego z tym tematem tego dokumentu. Niektóre z tych linków zasługują na osobne rozdziały, niektóre są doskonale udokumentowane same w sobie i nie potrzebują nawet HOWTO.

802.1Q VLAN Implementation for Linux (WWW) (http://scry.wanfear.com/~greear/vlan.html)

VLAN to fajny sposób na oddzielenie sieci bardziej na sposób wirtualny niż fizycznie. Dobre informacje na ten temat można znaleźć pod tym (ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-97/virtual_lans/index.htm) adresem. Dzięki tej implementacji, Linuks może wymieniać informacje z systemami takimi jak Cisco Catalyst, 3Com: {Corebuilder, Netbuilder II, SuperStack II switch 630}, Extreme Networks Summit 48, Foundry: ServerIronXL i FastIron.

Doskonałe HOWTO dotyczące VLANów dostępne jest pod tym (http://scry.wanfear.com/~greear/vlan/cisco_howto.html) adresem.

Uaktualnienie: łata została włączona do kernela 2.4.14 (a może 13).

Alternate 802.1Q VLAN Implementation for Linux (WWW) (http://vlan.sourceforge.net)

Alternatywna implementacja VLAN dla Linuksa. Projekt zaczął się od nieporozumień przy projekcie architektury VLANów i stylu kodowania. Jest generalnie rzecz biorąc ‘czyściej’ napisany.

Linux Virtual Server (WWW) (http://www.LinuxVirtualServer.org/)

Ci ludzie są błyskotiliwi. LVS to skalowalny i wydajny serwer zbudowany na klastrze prawdziwych serwerów, z równoważeniem obciążenia pracującym pod Linuksem. Architektura klastra jest przeźroczysta dla użytkowników. Końcowi użytkownicy widzą jeden wirtualny serwer.

W dużym skrócie: jeśli potrzebujesz równoważenia ruchu, na dowolnym poziomie ruchu, LVS w jakiś sposób sobie z tym poradzi. Niektóre z technik zastosowanych w tym rozwiązaniu są naprawdę diabelskie! Na przykład, pozwalają wielu maszynom przypisać ten sam adres IP w jednym segmencie, ale wyłączyć dla nich ARP. Tylko maszyna LVS obsługuje ARP - decyduje który serwer powinien obsłużyć pakiet i wysyła pakiet pod odpowiedni adres MAC. Ruch wychodzący przepływa bezpośrednio do routera a nie przez maszynę LVS, która w związku z tym nie potrzebuje nadzorować zawartości np. 5Gbit’owego strumienia danych wypływającego do świata, a w związku z tym nie może być wąskim gardłem.

LVS zaimplementowano jako łata na Linuksa 2.0 i 2.2 i jako moduł dla netfilter w 2.4/2.5 więc nie potrzebne są już łaty! Co prawda wsparcie dla 2.4 jest nadal we wczesnym stadium rozwoju ale komentarze i uwagi są mile widziane.

CBQ.init (WWW) (ftp://ftp.equinox.gu.net/pub/linux/cbq/)

Konfiguracja CBQ może być trochę zniechęcająca, szczególnie jeśli chcesz tylko trochę ograniczyć pasmo dla komputerów za routerem. CBQ.init może pomóc skonfigurować ci system przy pomocy uproszczonej składni.

Na przykład, jeśli chcesz by wszystkie komputery w sieci 192.168.1.0/24 (na 10mbitowym eth1) ograniczono do prędkości ściągania 28kbit/s zapisz te linijki do pliku konfiguracyjnego CBQ.init:

```
DEVICE=eth1,10Mbit,1Mbit
RATE=28Kbit
```

```
WEIGHT=2Kbit
PRIO=5
RULE=192.168.1.0/24
```

Oczywiście tego skryptu użyj tylko wtedy, gdy ‘jak i gdzie’ cię w ogóle nie interesuje. Używamy CBQ.init na maszynach produkcyjnych i działa dobrze. Może nawet robić pewne zaawansowane rzeczy, jak naprzykład kształtowanie ruchu na podstawie czasu. Razem ze skryptem dostarczana jest dokumentacja, która wyjaśnia wszystko czego nie znajdziesz w README.

Chronox easy shaping scripts (WWW) (http://www.chronox.de)

Stephan Mueller (smueller%chronox.de) napisał dwa użyteczne skrypty, ‘limit.conn’ i ‘shaper’. Pierwszy umożliwia łatwe ograniczenie pojedyńczej sesji ściągania danych tak jak poniżej:

```
# limit.conn -s SERVERIP -p SERVERPORT -l LIMIT
```

Skrypty działają na Linuksie 2.2 oraz 2.4/2.5.

Drugi skrypt jest trochę bardziej skomplikowany i może być użyty do stworzenia całej masy kolejek na podstawie reguł **iptables** oznaczających pakiety, które zostają potem poddane kształtowaniu ruchu.

Virtual Router Redundancy Protocol implementation (WWW) (http://w3.arobas.net/~jetienne/vrrpd/index.html)

Stworzona całkowicie dla zapewnienia redundacji. Dwie maszyny ze swoimi własnymi adresami IP i MAC tworzą razem trzeci adres IP i MAC, który jest wirtualny. Oryginalnie protokół tworzono dla ruterów, które potrzebowały stałych adresów MAC, ale działa również dla innych serwerów.

Piękno tego podejścia przejawia się w niewiarygodnie prostej konfiguracji. Bez kompilacji kernela czy łatania - wszystko w przestrzeni użytkownika.

Uruchom po prostu to polecenie na wszystkich maszynach uczestniczących w implementacji:

```
# vrrpd -i eth0 -v 50 10.0.0.22
```

I już działa! 10.0.0.22 jest teraz obsługiwany przez jeden z twoich serwerów, prawdopodobnie pierwszy na którym uruchomiono demona **vrrp**. Odłącz ten komputer od sieci i zobaczysz, że bardzo szybko drugi komputer przyjmie adres 10.0.0.22 jak również adres MAC.

Próbowałem tego u siebie i miałem działającą konfigurację w ciągu minuty. Z jakiegoś dziwnego powodu implementacja postanowiła odrzucić moją domyślną bramę, ale flaga ‘-n’ temu zapobiegła.

Poniżej ‘na żywo’ symulacja awarii jednej z maszyn:

```
64 bytes from 10.0.0.22: icmp_seq=3 ttl=255 time=0.2 ms
64 bytes from 10.0.0.22: icmp_seq=4 ttl=255 time=0.2 ms
64 bytes from 10.0.0.22: icmp_seq=5 ttl=255 time=16.8 ms
64 bytes from 10.0.0.22: icmp_seq=6 ttl=255 time=1.8 ms
64 bytes from 10.0.0.22: icmp_seq=7 ttl=255 time=1.7 ms
```

Nie straciliśmy nawet *jednego* pinga! Zaraz po pakiecie numer 4 odłączyłem swoje P200 od sieci i 486 przejęło obsługę, co można poznać po większym opóźnieniu.

Rozdział 19. Dalsza lektura

http://snafu.freedom.org/linux2.2/iproute-notes.html (http://snafu.freedom.org/linux2.2/iproute-notes.html)

Zawiera wiele technicznych informacji z komentarzy kernela.

http://www.davin.ottawa.on.ca/ols/ (http://www.davin.ottawa.on.ca/ols/)

Slajdy Jamala Hadi Salim'iego, jednego z autorów kontroli ruchu Linuksa

http://defiant.coinet.com/iproute2/ip-cref/ (http://defiant.coinet.com/iproute2/ip-cref/)

HTMLowa wersja dokumentacji Aleksieja w LaTeXie - wyjaśnia ze szczegółami część iproute2.

http://www.aciri.org/floyd/cbq.html (http://www.aciri.org/floyd/cbq.html)

Dobra strona Sally Floyd'a o CBQ, łącznie z jej pierwotnymi dokumentami. Żaden z nich nie jest specyficzny dla Linuksa, ale opisuje dobrze teorię i używa CBQ. Bardzo techniczne, ale dobre do czytania dla zapaleńców.

Differentiated Services on Linux

ten (ftp://icaftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt.gz) dokument autorstwa Wernera Almesbergera, Jamala Hadi Salim'iego i Aleksieja Kuzniecowa opisuje DiffServ z kernela Linuksa, między innymi TBF, GRED i DSMARK oraz klasyfikator TC_INDEX.

http://ceti.pl/~kravietz/cbq/NET4_tc.html (http://ceti.pl/~kravietz/cbq/NET4_tc.html)

Inne HOWTO, tym razem po Polsku! Możesz kopiować przykłady, działają przecież niezależnie od języka. Autor współpracuje z nami i być może wniesie wkład w niektóre sekcje tego HOWTO.

IOS Committed Access Rate (http://www.cisco.com/univercd/cc/td/doc/product/software/ios111/cc111/car.htm)

Od pomocnych ludzi z Cisco, którzy mają ten chwalebny zwyczaj umieszczania swojej dokumentacji w sieci. Składnia używana przez Cisco jest trochę inna, ale koncepcje są te same, poza tym, że potrafimy zrobić więcej bez ceny którą trzebaby zapłacić za routery w cenie samochodów :-)

Docum experimental site (WWW) (http://www.docum.org)

Stef Coene zajmuje się intensywnie przekonywaniem szefa do sprzedaży komercyjnego wsparcia dla Linuska i dużo eksperymentuje, głównie z zarządzaniem pasmem. Jego strona zawiera wiele praktycznych informacji, przykładów, testów i wytyka trochę błędów w CBQ/tc.

TCP/IP Illustrated, volume 1, W. Richard Stevens, ISBN 0-201-63346-9

Obowiązkowa lektura, jeśli chcesz naprawdę zrozumieć TCP/IP. I całkiem zajmująca.

Rozdział 20. Podziękowania

Naszym celem jest włączenie tu wszystkich ludzi, którzy przyczynili się do powstania tego HOWTO i pomogli odkryć jak to wszystko działa. O ile nie ma na razie planów co do stworzenia tablicy wyników na wzór netfilter, chcielibyśmy po prostu rozpoznawać pomocnych ludzi.

- Junk Alins

  <juanjo%mat.upc.es>
- Joe Van Andel
- Michael T. Babcock

  <mbabcock%fibrespeed.net>
- Christopher Barton

  <cpbarton%uiuc.edu>
- Peter Bieringer

  <pb:bieringer.de>
- Ard van Breemen

  <ard%kwaak.net>
- Ron Brinker

  <service%emcis.com>
- Łukasz Bromirski

  <l.bromirski%mr0vka.eu.org>
- Lennert Buytenhek

  <buytenh%gnu.org>
- Esteve Camps

  <esteve%hades.udg.es>
- Ricardo Javier Cardenes

  <ricardo%conysis.com>
- Stef Coene

  <stef.coene%docum.org>
- Don Cohen

  <don-lartc%isis.cs3-inc.com>
- Jonathan Corbet

  <lwn%lwn.net>
- Gerry N5JXS Creager

  <gerry%cs.tamu.edu>
- Marco Davids

  <marco%sara.nl>
- Jonathan Day

<jd9812%my-deja.com>

- Martin aka devik Devera
  <devik%cdi.cz>
- Hannes Ebner
  <he%fli4l.de>
- Derek Fawcus
  <dfawcus%cisco.com>
- David Fries
  <dfries%mail.win.org>
- Stephan "Kobold" Gehring
  <Stephan.Gehring%bechtle.de>
- Jacek Glinkowski
  <jglinkow%hns.com>
- Andrea Glorioso
  <sama%perchetopi.org>
- Thomas Graaf
  <tgraf%suug.ch>
- Sandy Harris
  <sandy%storm.ca>
- Nadeem Hasan
  <nhasan@usa.net>
- Erik Hensema
  <erik%hensema.xs4all.nl>
- Vik Heyndrickx
  <vik.heyndrickx%edchq.com>
- Spauldo Da Hippie
  <spauldo%usa.net>
- Koos van den Hout
  <koos%kzdoos.xs4all.nl>
- Stefan Huelbrock
  <shuelbrock%datasystems.de>
- Ayotunde Itayemi
  <aitayemi%metrong.com>
- Alexander W. Janssen
  <yalla%ynfonatic.de>
- Andreas Jellinghaus
  <aj%dungeon.inka.de>

- Gareth John
  <gdjohn%zepler.org>
- Dave Johnson
  <dj%www.uk.linux.org>
- Martin Josefsson
  <gandalf%wlug.westbo.se>
- Andi Kleen
  <ak%suse.de>
- Andreas J. Koenig
  <andreas.koenig%anima.de>
- Paweł Krawczyk
  <kravietz%alfa.ceti.pl>
- Amit Kucheria
  <amitk%ittc.ku.edu>
- Edmund Lau
  <edlau%ucf.ics.uci.edu>
- Philippe Latu
  <philippe.latu%linux-france.org>
- Arthur van Leeuwen
  <arthurvl%sci.kun.nl>
- Jose Luis Domingo Lopez
  <jdomingo%24x7linux.com>
- Robert Lowe
  <robert.h.lowe%lawrence.edu>
- Jason Lunz
  <j%cc.gatech.edu>
- Stuart Lynne
  <sl%fireplug.net>
- Alexey Mahotkin
  <alexm%formulabez.ru>
- Predrag Malicevic
  <pmalic%ieee.org>
- Patrick McHardy
  <kaber%trash.net>
- Andreas Mohr
  <andi%lisas.de>
- James Morris

<jmorris%intercode.com.au>

- Andrew Morton

  <akpm%zip.com.au>

- Wim van der Most
- Stephan Mueller

  <smueller%chronox.de>

- Togan Muftuoglu

  <toganm%yahoo.com>

- Chris Murray

  <cmurray%stargate.ca>

- Patrick Nagelschmidt

  <dto%gmx.net>

- Ram Narula

  <ram%princess1.net>

- Jorge Novo

  <jnovo%educanet.net>

- Patrik

  <ph%kurd.nu>

- P?l Osgy?ny

  <oplab%westel900.net>

- Lutz Preszligler

  <Lutz.Pressler%SerNet.DE>

- Jason Pyeron

  <jason%pyeron.com>

- Rod Roark

  <rod%sunsetsystems.com>

- Pavel Roskin

  <proski@gnu.org>

- Rusty Russell

  <rusty%rustcorp.com.au>

- Mihai Rusu

  <dizzy%roedu.net>

- Rob Pitman

  <rob%pitman.co.za>

- Jamal Hadi Salim

  <hadi%cyberus.ca>

- Rene Serral

  <rserral%ac.upc.es>

- David Sauer

  <davids%penguin.cz>
- Sheharyar Suleman Shaikh

  <sss23%drexel.edu>
- Stewart Shields

  <MourningBlade%bigfoot.com>
- Nick Silberstein

  <nhsilber%yahoo.com>
- Konrads Smelkov

  <konrads%interbaltika.com>
- William Stearns

  <wstearns%pobox.com>
- Andreas Steinmetz

  <ast%domdv.de>
- Matthew Strait

  <straitm%mathcs.carleton.edu>
- Jason Tackaberry

  <tack%linux.com>
- Charles Tassell

  <ctassell%isn.net>
- Glen Turner

  <glen.turner%aarnet.edu.au>
- Sponsor herbaty: Eric Veldhuyzen

  <eric%terra.nu>
- Thomas Walpuski

  <thomas%bender.thinknerd.de>
- Song Wang

  <wsong%ece.uci.edu>
- Chris Wilson

  <chris%netservers.co.uk>
- Lazar Yanackiev

  <Lyanackiev%gmx.net>
- Pedro Larroy

  <piotr%member.fsf.org>

  - Chapter 15, section 10: Example of a full nat solution with QoS
  - Chapter 17, section 1: Setting up OSPF with Zebra