

# Kształtowanie Ruchu i Zaawansowany Ruting HOWTO

etherlabs BV (bert hubert <bert.hubert@netherlabs.nl>)

Gregory Maxwell <greg@linuxpower.cx>

Remco van Mook <remco@virtu.nl>

Martijn van Oosterhout <kleptog@cupid.suninternet.com>

Paul B Schroeder <paulsch@us.ibm.com>

Jasper Spaans <jasper@spaans.ds9a.nl>

howto@ds9a.nl

Wersja oryginalna: v0.9.0, 2001/12/19 22:43:38

Oryginał tego dokumentu znajduje się pod adresem: <http://ds9a.nl>

**Tłumaczenie: Łukasz Bromirski, l.bromirski@mr0vka.eu.org**

Wersja tłumaczenia: 1.0, \$Date: 2002/08/22 21:29:40 \$

Oryginał tego tłumaczenia znajduje się pod adresem: <http://mr0vka.eu.org/docs/tlumaczenia/2.4routing/2.4routing.big.html>

---

*Bardzo praktyczne podejście do iproute2, kontroli ruchu i trochę do netfilter.*

---

## 1. Uwagi od tłumacza

Zagadnienia opisywane w tym HOWTO obejmują szeroki zakres pojęć i zagadnień technicznych. Ponieważ niewiele jest w Polsce publikacji dotyczących tematu i w zasadzie niektóre z zagadnień poruszanych poniżej mają wiele różnych oznaczeń nawet w języku angielskim, jednolite tłumaczenie tego HOWTO było dosyć trudne. Jeśli masz jakieś uwagi co do tłumaczenia, chciałbyś zaproponować inne tłumaczenie jakiś terminów lub w ogóle koncepcję pojęciową, proszę, skontaktuj się ze mną e-mailem. Będę bardzo wdzięczny za każdą uwagę.

Na potrzeby tego HOWTO przyjąłem pewne umowne oznaczenia, z uwagi na ich powszechność. Skrót `tc' oznaczający w oryginale `traffic shaping' tłumaczony jest w tym HOWTO jako `kształtowanie ruchu' ale również często po prostu `tc'. Nie chciałem silić się po prostu na stosowanie polskiego skrótu, który i tak się raczej nie przyjmie a wprowadzi tylko zamieszanie (narzędzie również nazywa się `tc').

Analogicznie jest ze skrótem 'qdisc' - 'queueing discipline' - zdecydowałem się na pojęcie 'kolejka z dyscypliną', ale często pozostaję przy angielskim 'qdisc' jako oznaczenie tego pojęcia.

## 2. Dedykacja

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

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

## 3. Wprowadzenie

Witam, Szanowny Czytelniku.

To HOWTO 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 spektakularne rzeczy. Komendy takie jak 'route' czy 'ifconfig' są zaledwie cieniutkimi **osłonami** (ang. *wrapper*) na bardzo potężną 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 [HOWTO@ds9a.nl](mailto:HOWTO@ds9a.nl). Prosimy jednak byś rozważył wysłanie postu na listę pocztową jeśli masz pytania nie dotyczące bezpośrednio tego HOWTO.

Zanim zgubisz się czytając ten dokument, a wszystko co chcesz robić to kształtowanie ruchu, pomiń wszystko i przejdź bezpośrednio do rozdziału 'Inne możliwości' i poczytaj o CBQ.init.

### 3.1 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 rozdystrybuuje pornografię do najbardziej cenionych klientów - to nie była nasza wina. Przykro nam.

Copyright (c) 2001 przez Bert Hubert, Gregory Maxwell, Martijn van Oosterhout, Remco van Mook, Paul B. Schroeder i inni. Materiał ten może być dystrybuowany tylko na zasadach określonych w Otwartej Licencji Publikacji, v1.0 lub późniejszej (ostatnia wersja jest obecnie dostępna pod adresem <http://www.opencontent.org/openpub/>).

Prosimy o darmowe kopiowanie i dystrybuowanie (sprzedawanie lub rozdawanie) 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 papierowej, autorzy otrzymali próbki na `potrzeby recenzji' :-).

## 3.2 Wymagana wiedza

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

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

### Zagadnienia sieciowe HOWTO

[Zagadnienia sieciowe HOWTO, Rusty Russell'a](#) Bardzo ładne wprowadzenie, wyjaśniające co to jest sieć i jak łączy się z innymi sieciami.

### Sieć Linuksa (kiedyś Net-3 HOWTO)

Doskonała rzecz, mimo że bardzo szczegółowa. Uczy wielu rzeczy, które są już skonfigurowane jeśli możesz połączyć się z Internetem. Powinno znajdować się w `/usr/doc/HOWTO/NET3-4-HOWTO.txt`, ale można je również znaleźć pod adresem <http://www.linuxports.com/howto/networking>.

## 3.3 Co Linuks może zrobić dla ciebie

Krótką listą 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** (*Denial of Service*, Ataki Zaprzeczenia Usługi)
- bronić Internet przed twoimi klientami
- wykorzystywać wiele serwerów jak jeden, by uzyskać **równoważenie obciążenia** (ang. *load balancing*) i **zwiększoną dostępność** (ang. *enhanced availability*)
- ograniczyć dostęp do twoich komputerów
- ograniczyć dostęp twoich użytkowników do innych komputerów
- prowadzić ruting w oparciu o identyfikator użytkownika, adres MAC, źródłowy 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.

## 3.4 Notatki gospodyni

Warto zaznaczyć parę rzeczy dotyczących tego dokumentu. Mimo że napisałem jego większość, 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 jesteś lepiej wykwalifikowany 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, chciałbym by zrobiło tak więcej ludzi.

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, proszę daj nam znać byśmy mogli usunąć notkę FIXME.

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

## 3.5 Dostęp, CVS & wysyłanie uaktualnień

To HOWTO znajduje się pod adresem <http://www.ds9a.nl/lartc>.

Mamy anonimowe CVS dostępne z całego świata. Jest to przydatne z wielu powodów. Uaktualnienie twojej kopii do najnowszej wersji czy też wysłanie nam poprawek nie powinno być żadnym problemem.

Co więcej, system ten umożliwia wielu autorom na pracę nad tekstem źródłowym niezależnie.

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

Jeśli zauważysz błąd, lub chciałbyś coś dodać, popraw to na swojej lokalnej kopii, a następnie uruchom `cvs diff -u` i wyślij rezultat do nas.

Razem z dokumentem źródłowym 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 `sgml-tools`, `ghostscript` i `tetex` by otrzymać wszystkie formaty.

## 3.6 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.

## 3.7 Układ tego dokumentu

Zacniemy robić interesujące rzeczy praktycznie zaraz, co oznacza, że na początku będzie trochę rzeczy od razu nie wytłumaczonych, lub napisanych niezbyt dokładnie. Przeczytaj je zakładając, że wszystko stanie się jasne potem.

Ruting i filtrowanie to dwie różne rzeczy. Filtrowanie zostało bardzo dobrze udokumentowane w HOWTO Rusty'ego, które dostępne jest pod adresem <http://netfilter.samba.org/unreliable-guides/>.

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

## 4. Wprowadzenie do iproute2

### 4.1 Dlaczego iproute2?

Większość dystrybucji Linuksa, i większość UNIX'ów, 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 routinga 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 rutujący, filtrujący i klasyfikujący jest potężniejszy niż ten dostarczany w większości dedykowanych ruterów, ścian ogniowych i produktów kształtujących pasma.

W trakcie rozwijania nowych koncepcji sieciowych, ludzie znajdowali zawsze sposoby by dołożyć je do istniejących szkieletów już stworzonych systemów operacyjnych. To ciągle 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, Linuks 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.

### 4.2 Wycieczka po iproute2

Linuks ma bardzo wyrafinowany system kształtowania pasma, nazywany **Kształtowaniem Ruchu** (ang. *Traffic Control*, *TC*). System ten udostępnia wiele metod klasyfikowania, priorytetowania, współdzielenia i ograniczania zarówno ruchu wychodzącego jak i przychodzącego.

Zacniemy od krótkiej wycieczki po możliwościach iproute2.

## 4.3 Wymagania wstępne

Powinieneś upewnić się, że wszystkie narzędzia z **przestrzeni użytkownika** (ang.*userland*) są zainstalowane. Paczka nazywa się `iproute` zarówno w przypadku RedHat'a jak i Debiana, można ją znaleźć pod adresem <ftp://ftp.inr.ac.ru/ip-routing/iproute2-2.2.4-now-ss?????.tar.gz>.

Możesz również próbować pod adresem <ftp://ftp.inr.ac.ru/ip-routing/iproute2-current.tar.gz>, by uzyskać najnowszą wersję.

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 z większością opcji odpowiedzialnych za kontrolę ruchu domyślnie w konfiguracji kernela.

RedHat 7.2 ma domyślnie włączone wszystko.

Upewnij się również, że masz wsparcie dla netlink, jeśli będziesz musiał przebudować swój kernel. Jest on wymagany przez `iproute2`.

## 4.4 Badanie obecnej konfiguracji

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

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

### `ip` pokazuje nam powiązania

```
[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 rutera NAT. Wytlumaczę 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 - Maksymalnej Jednostki**

**Transmisji** (ang.*Maximum Transfer Unit*) 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 'powiązań' z 'adresami IP'. Po wprowadzeniu 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 nie interesują one nas teraz.

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 są połączone do tego interfejsu, więc `10.250.3.13` jest bezpośrednio osiągalny z `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'.

Jest bardzo ważne, byś zrozumiał zawartości tych wydruków. Zajrzyj do wspomnianej wcześniej dokumentacji, jeśli masz problemy.

Możesz również zauważyć `qdisc`, co oznacza **Dyscyplinę Kolejowania** (ang. *Queueing Discipline*). Znaczenie tej funkcji stanie się jasne później.

## ip pokazuje nam wpisy z tablicy routingu

Cóż, wiemy jak znaleźć adresy `10.x.y.z` i jesteśmy w stanie dotrzeć do `212.64.94.1`. To jednak nie wystarczy, musimy mieć jeszcze instrukcje jak dotrzeć do 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 odpowiedzi i dostarczał je nam.

```
[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 raczej samoumieszczający 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
```

## 4.5 ARP

ARP, to **Protokół Rozwiązywania Adresów** (ang. *Address Resolution Protocol*), opisany w [RFC 826](#). ARP używany jest w komputerach podłączonych do sieci do **rozwiązywania** (ang. *resolve*) 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 może 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 jest `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. Odpowiedź 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 znaleźć kolegę w

sieci, foo będzie w stanie komunikować się z bar dopóki zapomni (lub pamięć podręczna arp wygaśnie) gdzie jest bar (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 znaleźć 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, który próbował znaleźć espa043, adres tego ostatniego został dodany do pamięci podręcznej arp. Dopóki ten wpis nie wygaśnie (w rezultacie braku komunikacji pomiędzy obydwojema komputerami), espa041 wie gdzie znaleźć espa043 i nie ma potrzeby rozsyłania zapytań ARP.

Teraz wykasujemy 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 znaleźć 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.

## 5. Reguły - baza danych polityki routingu

Jeśli posiadasz duży ruter, możesz zaspokajać potrzeby różnych ludzi, którzy chcą być obsługiwani w różny sposób. Baza danych polityki routingu pozwala właśnie na to, 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 opisują dowolne pakiety ( `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 rutowania - 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.

## 5.1 Prosty routing na podstawie źródła (*source routing*)

Wróćmy do prawdziwego przykładu. Mam 2 (dokładniej 3, w momencie gdy je zwracałem) modemy kablowe połączone do linuxowego rutera 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, między innymi 212.64.78.148 w tym przykładzie, podłączony jest 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
```

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 wszystko co zostało, to wygenerować tabelę John'a 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.

## 6. GRE i inne tunele

W Linuksie istnieją trzy typy tuneli. Jest tunelowanie IP w IP, tunelowanie GRE i tunele, które żyją poza kernelem (takie, jak na przykład PPTP).

### 6.1 Parę ogólnych uwag o tunelach

Tunele można użyć do bardzo niezwykłych i fajnych rzeczy. Mogą również spowodować koszmarnie 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, 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. Niekoniecznie jest to od razu problemem, ale pomyśl o fragmentacji i składaniu pakietów jeśli chcesz łączyć duże sieci tunelami. Aha, oczywiście najszybszym sposobem wykopania tunelu jest kopanie po obu stronach.

## 6.2 Tunelowanie IP w IP

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

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

```
sieć          10.0.1.0
maska sieci   255.255.255.0
ruter         10.0.1.1
```

Ruter ma adres `172.16.17.18` w sieci C.

i sieć B:

```
sieć          10.0.2.0
maska sieci   255.255.255.0
ruter         10.0.2.1
```

Ruter ma adres `172.19.20.21` w sieci C.

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
```

I w zasadzie skończyłeś tunel:

```
ifconfig tunl0 down
```

Presto, koniec. Nie możesz jednak przekazywać broadcast'ów 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 kompatybilność, kod realizujący tę funkcjonalność był obecny już od dawna, działa więc nawet jeszcze w kernelach serii 1.3. O ile wiem, tunelowanie IP-w-IP Linuksa nie działa z innymi Systemami Operacyjnymi czy routerami. Jest proste, działa. Używaj jeśli musisz, a najlepiej przejdź na GRE.

## 6.3 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 multicast'owy i IPv6 przez tunel GRE.

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

### Tunelowanie IPv4

Najpierw zrealizujmy tunelowanie IPv4:

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

```
sieć          10.0.1.0
maska sieci   255.255.255.0
ruter         10.0.1.1
```

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

w sieci B:

```
sieć          10.0.2.0
maska sieci   255.255.255.0
ruter         10.0.2.1
```

Ruter ma adres `172.19.20.21` w sieci C. Nazwijmy ją *netb* (nadał mało oryginalne).

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 chodzi o protokół GRE ( `mode gre` ), że zdalny adres to `172.19.20.21` (ruter po drugiej stronie), że nasze tunelowane pakiety powinny pochodzić z `172.16.17.18` (co umożliwia twojemu routerowi posiadać wiele adresów w sieci C ale określić którego używać do tunelowania) 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`. Jest to w porządku dla małych sieci, 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 marszrutę 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 jedyńki. 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 rutera B.

## Tunelowanie IPv6

Zajrzyj do sekcji 6 po krótkie wprowadzenie o adresach IPv6.

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 ruter 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) i wskazaliśmy gdzie ma być skierowany ( *remote* ) i skąd pochodzić ( *local* ). TTL ustawione jest na maksymalną wartość 255. Następnie, podnieśliśmy interfejs ( *up* ). Kolejną czynnością było dodanie naszego własnego adresu sieciowego i ustawienie marszruty na *3ffe::/15* (która jest obecnie taka sama dla całej sieci 6bone) w tunelu.

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

## 6.4 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.

## 7. Tunelowanie IPv6 z Cisco i/lub 6bone

Autorstwa Marco Davids <marco@sara.nl>

UWAGA do koordynatora:

Na tyle ile wiem, ten rodzaj tunelowania IPv6-IPv4 nie jest tunelowaniem GRE według definicji. Można tunelować IPv6 przez IPv4 w urządzeniach tunelujących GRE (GRE tuneluje WSZYSTKO przez IPv4), ale urządzenie używane w tym rozwiązaniu tuneluje tylko IPv6 przez IPv4 i w związku z tym to co innego.

### 7.1 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 jedyne go 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 dokładność. Oprócz tego, IPv6 (lub IPng, co oznacza IP Next Generation) ma zapewnić mniejsze tablice routingu na ruterach szkieletowych, prostszą konfigurację i sprzęt, 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. W związku z tym, by życie stało się prostsze, są pewne reguły:

- Nie używaj wiodących zer. Tak jak w IPv4.
- Używaj średników do oddzielania każdych 16 bitów, lub dwóch bajtów.
- Jeśli masz dużo kolejnych zer, możesz zapisać je jako ::. Ale możesz zrobić to tylko raz w adresie 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 przyjaźniejsze.

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 enkapsulowanie protokołu IPv6 w ramki IPv4 i wysyłanie tak spreparowanych pakietów przez istniejącą infrastrukturę IPv4 z jednej sieci IPv6 do innej.

To jest dokładnie miejsce, 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:

- Zdobądź którąś z nowszych dystrybucji Linuksa z odpowiednią glibc.
- Zdobądź nowe źródła kernela.

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

- Przejdź do `/usr/src/linux` i napisz:
- `make menuconfig`

- Wybierz "Networking Options"
- Wybierz "The IPv6 protocol", "IPv6: enable EUI-64 token format", "IPv6: disable provider based addresses"

**UWAGA:** 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.

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

Jest bardzo dużo dobrej dokumentacji dotyczącej kompilowania i instalowania kernela, 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 twój 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 ruter 6bone ma adres `145.100.1.5`:

```
# ip tunnel add sixbone mode sit remote 145.100.1.5 [local 145.100.24.181 ttl 225]
# 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*. Później ustawiamy go w tryb *sit*, mówimy gdzie ma wskazywać ( *remote* ) i skąd prowadzić ( *local* ). TTL ustawione jest na wartość maksymalną - 255.

Następnie podnosimy nasze urządzenie ( *up* ). Potem dodajemy adres naszej sieci i ustawiamy trasę dla `3ffe::/15` (którą obecnie ma 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 rutera 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 bind'a 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 to wszystko wykracza poza ten dokument.

W konfiguracji rutera Cisco należy wpisać coś takiego:

```
!
interface Tunnel1
description IPv6 tunnel
no ip address
no ip directed-broadcast
ipv6 enable
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.

## 8. IPsec: bezpieczne IP przez Internet

FIXME: nie ma autora!

W międzyczasie zajrzyj pod adres projektu FreeS/WAN: <http://www.freeswan.org/>. Inną implementację IPsec oferuje dla Linuksa Cerberus z NIST, ale ich strona nie była uaktualniana już od ponad roku a wersje zwykle miały poważne opóźnienia w stosunku do najnowszych kerneli. Inną implementacją protokołu IPv6 - USAGI, również zawiera implementację IPsec, ale prawdopodobnie tylko dla IPv6.

## 9. Ruting multicast'owy

FIXME: nie ma autora!

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

Zanim zajmiesz się rutingiem multicast'owym, musisz skonfigurować swój kernel. Oznacza to zdecydowanie się na konkretny typ tego rutingu. Są cztery główne "wspólne" rodzaje - DVMRP (multicast'owa 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, który zakłada że grupy użytkowników będą blisko siebie).

W kernelu Linuksa nie ma tych opcji. Jest tak, ponieważ sam protokół obsługiwany jest przez aplikację rutującą, np. Zebra, mroute czy pimd. Z drugiej strony musisz wiedzieć, którego protokołu będziesz używać by wybrać w konfiguracji kernela odpowiednie opcje.

Dla całego rutingu multicast'owego, 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 multicast'owymi. 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 gdzieś również IGMPv1.

Jak na razie szło nieźle. Włączyliśmy multicasting. Teraz poinstruujemy kernel Linuksa by coś z tym zrobił tak, byśmy mogli zacząć rutowanie. Oznacza to dodanie wirtualnej sieci multicast'owej do tabeli rutowania:

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

(zakładając oczywiście że zamierzasz obsługiwać ruch multicast'owy 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 włączonym multicast'ingiem *powinny* odpowiedzieć, ale nic innego. Zauważysz, że żadna maszyna która odpowiedziała, nie miała adresu `224.0.0.1`. Co za niespodzianka! :) Jest to po prostu adres grupowy (broadcast do prenumeratorów) i wszyscy członkowie grupy odpowiadają swoimi adresami.

```
ping -c 2 224.0.0.1
```

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

(Będzie kontynuowane)

## 10. Dyscypliny kolejujące dla Zarządzania Pasmem

Gdy to odkryłem, *naprawdę* mnie to rozwalilo. Linux 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 daje ramka i ATM.

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

```
mb = 1024 kb = 1024 * 1024 b => bajtów/s  
mbit = 1024 kbit = 1024 * 1024 bit => bitów/s.
```

### 10.1 Kolejki i Dyscypliny kolejek 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 zmusić świat by wysyłał ci tylko określoną liczbę poczty bez skontaktowania najpierw ze wszystkimi ludźmi.

Ale Internet 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 szybciej i 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 później.

Jest to porównywalne do nieczytania połowy 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 ruter i chciałbyś zapobiec sytuacji, w której określone komputery ściągają dane za szybko, musisz wprowadzić ograniczenia na **wewnętrznym** interfejsie rutera, tego który wysyła dane do twoich komputerów.

Musisz być również pewien, że kontrolujesz połączenie w najwęższym miejscu. Jeśli masz 100Mbit'ową kartę sieciową a ruter ma łącze o przepustowości 256kbit'ów, musisz upewnić się że nie wysyłasz więcej danych niż ruter jest w stanie obsłużyć. Jeśli o to nie zadbasz, to ruter będzie kontrolował połączenie i ograniczał pasmo. Musimy `zawładnąć

kolejką' mówiąc po prostu i być najwolniejszym połączeniem w łańcuchu. Jest to na szczęście bardzo łatwe.

## 10.2 Proste, bezklasowe Dyscypliny Kolejowania

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

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

Najczęściej używaną dyscypliną jest `pfifo_fast` - jest ona 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 to tradycyjne **Pierwszy Wszedł, Pierwszy Wyjdzie** (ang.*First In First Out*, FIFO), co oznacza, że żaden pakiet nie będzie specjalnie traktowany. Przynajmniej nie wprost. Kolejka ta ma 3 **pasma** (ang.*band*). W każdym paśmie z osobną działającą regułą FIFO. Jednak dopóki 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.

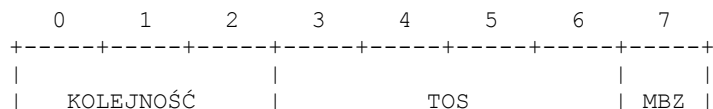
Nie pomylcie tej bezklasowej kolejki z klasową kolejką PRIO! Mimo, że zachowują się podobnie, `pfifo_fast` jest bezklasowa i nie można dodawać innych kolejek do niej przy użyciu polecenia `tc`.

### 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:



```

|-----|-----|-----|
+-----+-----+-----+-----+

```

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 znaleźć 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 'Pakiem 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 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	(większość)
Odpowiedzi	<takie jak zapytania> (większość)	

## 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 tego parametru za pomocą polecenia `tc`.

## Token Bucket Filter

Token Bucket Filter (TBF) 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 to ustawienie.

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, ciągle obciążenie większe od maksymalnego zdefiniowanego spowoduje przeładowanie i pakiety zaczną być opóźniane, a w końcu odrzucane.

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

## Parametry i ich zastosowanie

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 ustawione), częstotliwość szczytową.

### burst/buffer/maxburst - seria/bufor/maksymalna seria

Rozmiar `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 żadaną 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 - częstotliwość

Ustawienie szybkości. 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 - częstotliwość szczytowa**

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 wyślemy dopiero następny - dzięki przeliczeniu przestrzeni czasowej między pakietami tak, aby wysłać dokładnie tyle ile wynosi wartość tego parametru.

Jednak ponieważ timer Unix'owy 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 rozwala interaktywność.

Dzieje się tak dlatego, ponieważ wysyłanie pakietów zapełni kolejkę w modemie, która jest prawdopodobnie **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 Kolejowanie 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 słowem przy omawianiu SFQ jest **konwersacja** (ang.*conversation*) lub **przepływ** (ang.*flow*), które odpowiadają prawie sesjom TCP czy strumieniom UDP. Ruch dzielony jest na dosyć dużą liczbę kolejek FIFO, jedną dla każdej konwersacji, a następnie rozsyłany algorytmem round-robin, dzięki czemu każda sesja ma zawsze szansę coś wysłać.

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

Ponieważ używana jest wartość mieszająca (hash), więcej niż jedna sesja mogą skończyć w tym samym wiadrze. Oznaczałoby to w praktyce zmniejszenie szansy wysłania pakietu o połowę i podzielenie tym samym na pół dostępną efektywną prędkość. By temu zapobiec, SFQ zmienia często algorytm mieszający i nawet jeśli dojdzie do opisanej sytuacji będzie to działało 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 kolejowania 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 zastosowanie

SFQ jest praktycznie samokonfigurowalna:

### perturb

Wartość określająca 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, ta linijka pomoże ci wydłużyć sprawiedliwy podział ruchu:

```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc -s -d qdisc ls
qdisc sfq 800c: dev eth0 quantum 1514b limit 128p flows 128/1024 perturb 10sec
  Sent 4812 bytes 62 pkts (dropped 0, overlimits 0)
```

Numer 800c: jest automatycznie nadawaną 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.

## 10.3 Porady kiedy używać której kolejki

Podsumowując, są to 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 'Zaawansowanych i mniej popularnych dyscyplinach kolejkowania'.

- 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 kontrolować 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.
- 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 w logu.

## 10.4 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'.

### Dyscyplina kolejkowania (ang.*Queueing Discipline*)

Algorytm zarządzający kolejką urządzenia, dla ruchu **przychodzącego** (ang.*ingrees*) lub **wychodzącego** (ang.*egrees*).

### Bezklasowa dyscyplina kolejkowania (ang.*Classless qdisc*)

Dyscyplina kolejkowania bez możliwości tworzenia 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

Dyscyplina kolejkowania z klasami, tak jak wskazuje jej nazwa, zawiera klasy. Mogą one zawierać kolejne dyscypliny i tak dalej i tak dalej.

### 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 (`qdisc`), 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 wypuszczenia pakietu tak by pasował do zdefiniowanych charakterystyk ruchowych i jego maksymalnych wartości. Kształtowanie ruchu wykonywane jest przy opuszczaniu przez pakiet systemu.

### Narzucanie polityki (ang.*Policing*)

Proces odwrotny do kształtowania, wykonywany na ruchu przychodzącym. Może on tylko odrzucać pakiety a nie odwlekać ich obsługę - nie ma 'kolejki dla ruchu przychodzącego'.

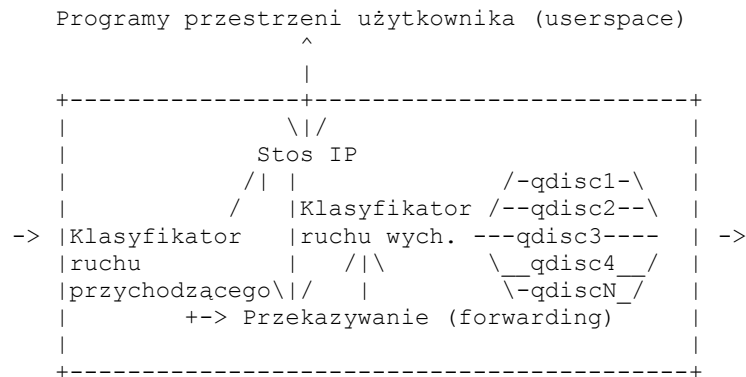
### Kolejka bezstratna (ang.*Work-Conserving*)

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 wysłą pakietu, mimo że gdzieś w nich się znajduje.

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



Duży kwadrat reprezentuje kernel. Lewa strzałka reprezentuje ruch docierający do twojego komputera z sieci. Zostaje on następnie wrzucony do klasyfikatora ruchu przychodzącego, który może zastawać 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** (ang. *enqueueing*).

Pakiet czeka zatem w qdisc na decyzję kernela powodującą wysłanie go przez któryś interfejs sieciowy. Nazywamy to z kolei **opróżnianiem kolejki** (ang. *dequeueing*).

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.

## 10.5 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 sendmail'a' - 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

Gdy ruch dociera 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, która go zawiera.

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 ruter bez żadnej zwłoki: interfejs wyjściowy jest dużo szybszy niż faktyczna przepustowość łącza. Nie ma więc kolejki, którą możnaby kontrolować.

## Rodzina kolejek z klasami: korzenie, uchwyt, rodzeństwo i rodzice

Każdy interfejs ma jedną wychodzącą kolejkę-korzeń, domyślnie jest to wspomniana wcześniej bezklasowa `pfifo_fast`. Każdej z qdisc można nadać uchwyt, który będzie potem używany przez polecenia konfiguracyjne by odwołać się do tej konkretnej kolejki. Poza wychodzącymi kolejkami, interfejs może mieć również przychodzące, które podlegają narzuceniu polityki.

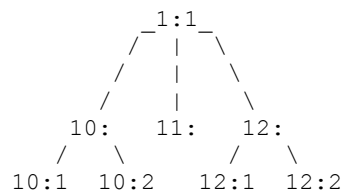
Uchwyt kolejki składa się z dwóch części: starszego i młodszego numeru. Charakterystyczną nazwą dla kolejki-korzenia jest `1:`, które równe jest `1:0`. Młodszy numer kolejki z dyscypliną zawsze równy jest 0.

Klasy muszą mieć ten sam starszy numer, określający ich rodzica.

## Jak filtry używane są do klasyfikowania ruchu

Reasumując, typowa hierarchia może wyglądać tak:

```
root 1:
|
```



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 kolejgowane i usuwane z kolejki w qdisc-korzeniu, który jest jedyną kolejką z którą rozmawia kernel.

Pakiet może być sklasyfikowany w następującym łańcuchu:

```
1: -> 1:1 -> 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, zagnieźdzone 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 kolejgowanie 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 qdisc, które masz dostępne.

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 priorytetyzować określone rodzaje ruchu nie tylko na podstawie flag ToS, ale całej potęgi, którą zapewniają filtry `tc`. Może również zawierać inne kolejki, 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 na 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:

```
root 1: prio
  /  |  \
1:1 1:2 1:3
 |   |   |
10: 20: 30:
sfq  tbf  sfq
band 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
  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
```

```
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
  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 kolejka 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 Linux.

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% beczynne. Jeśli nie jest, musimy je dławić by BYŁO w 90% czasu faktycznie beczynne.

Jest to raczej trudne do zmierzenia, więc CBQ pobiera czas beczynności 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 beczynności?

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 rękę, powinieneś poradzić 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 beczynne i dzięki temu prawdziwa przepustowość spada do skonfigurowanej wielkości. Żeby to robić kalkuluje czas, który powinien upłynąć pomiędzy kolejnymi pakietami.

W czasie pracy, efektywny czas beczynności mierzony jest za pomocą **wykładnika ważonego średniego przesylu** (ang.*exponential weighted moving average, EWMA*), który traktuje świeże pakiety jako wykładniczo ważniejsze niż starsze. Tak samo kalkulowany jest poziom załadowania stacji uniksowych.

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

Przeładowane łącze ma ujemny średni czas beczynności, 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 beczynności, który po paru godzinach ciszy umożliwiłby zajęcie nieskończonej przepustowości. By temu zapobiec, wartość średniego czasu beczynności 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 parametry `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 beczynności.

### 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 (bursts). Nie możesz ustawić bezpośrednio `maxidle`, 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 unixa 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

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

Wewnętrznie, CBQ ma masę parametrów konfiguracyjnych. Na przykład, klasy o których 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 kształtowaniem 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 otrzymuje zgodne 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 urządzenie 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 czegoś w swojej kolejce. 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.

Proszę zapamiętać, że wszystkie klasy w obrębie hierarchii CBQ współdzielą ten sam numer starszy!

## **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 pożyczka przepustowości rodzeństwu. Użyj go jeśli masz konkurujące lub wzajemnie nieprzyjazne działy na jednym łączu. Program `tc` umożliwia 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 pojedynczej jednostki, mogą znaleźć się klasy, którym zezwolimy na wymianę przepustowości.

## Przykładowa konfiguracja

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 dodać je 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, data będą przetwarzane w obrębie klasy 1:0 i w związku z tym nie będą podlegać ograniczeniom.

Jeśli 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  $5/8 * 6\text{mbit} = 3.75\text{mbit'a}$ .

## Inne parametry: split & defmap

Tak jak powiedziano wcześniej, 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 OR z polem defmap by sprawdzić czy istnieje pasujący odpowiednik. Innymi słowy, jest to krótszy sposób na stworzenie bardzo szybkiego filtra, 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. Zjrzyj do sekcji opisującej `pfifo_fast` po więcej szczegółów jak bity ToS konwertowane 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 'BESTEFFORT' 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 (*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/>.

Gdy twoja konfiguracja HTB będzie stawiała się coraz bardziej skomplikowana, będzie się ona dobrze skalować. Jeśli chodzi o CBQ, to wiadomo, że jest już skomplikowana na początku! HTB nie jest jeszcze częścią standardowego kernela, ale niedługo zapewne będzie.

Jeśli akurat masz szansę załatać swój kernel, powinienes 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.

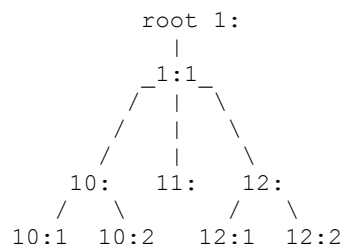
HTB z pewnością wygląda pięknie - jeśli 10: i 20: obie mają swoją gwarantowaną przepustowość i zostało coś do podzielenia, pożyczają w stosunku 5:3, tak jakbyś mógł się spodziewać.

Ruch niepasujący 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!

## 10.6 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:



W momencie kolejowania 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 powinienes dołączać wszystkie filtry do korzenia głównego!

I znowu - pakiety są kolejowane 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
```

Przydziela to 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 pojedynczy 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 ruting pomiędzy interfejsami. Jest to użyteczne dla np. kształtowania ruchu na interfejsie `eth1`, który przyszedł z `eth0`. Składnia:

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

Zwróć uwagę, że to nie jest test u32! Możesz oznaczyć pakiet w ten sposób:

```
# iptables -A FORWARD -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
```

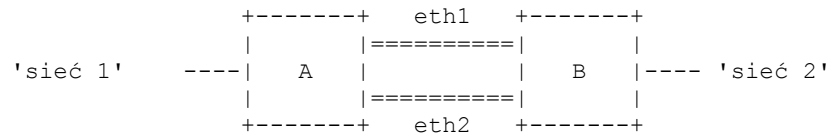
Dla ruchu typowego użyj `0x08 0xff`.

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

# 11. 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 założmy na moment, że pracują w oparciu o Linux. Jeśli ruch przechodzi z sieci 1 do sieci 2, router A będzie musiał rozłożyć pakiety na oba połączenia z routerem B. Router 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.

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
```

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** (ang. *raw*) na interfejsach `eth1` i `eth2`.

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

FIXME: czy to nie potrzebuje czegoś w rodzaju `nobroadcast`? Sieć `/31` jest za mała by pomieścić adres sieci i broadcast'owy - jeśli to nie działa tak jak planowaliśmy, spróbuj `/30` i odpowiednio dostosuj adresy IP. Możesz nawet spróbować ustawić `eth1` i `eth2` na pracę bez adresów IP!

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

## 11.1 Problemy

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

```
# echo 0 > /proc/net/ipv4/conf/eth1/rp_filter
# echo 0 > /proc/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, ruter 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.

## 12. Netfilter & iproute - oznaczanie pakietów

Jak na razie, widzieliśmy jak działa iproute i wspomnieliśmy o netfilter. Jest to dobry moment by przejrzeć kolekcję poradników Rusty'ego: <http://netfilter.samba.org/unreliable-guides/>. Sam netfilter można znaleźć pod adresem <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 założmy, 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 poinstruuujemy bazę danych polityki routingu by odpowiednio pracowała:

```
# 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 rutowania 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 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 <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 kernela:

```
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 [Transparent web-caching using netfilter, iproute2, ipchains and squid](#) w Książce Kucharskiej.

## 13. 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**

Podjekuje 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**

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

#### **route**

Podjmuje decyzję na podstawie trasy, którą pakiet będzie rutowany.

#### **rsvp, rsvp6**

Kieruje pakiety na trasy określone przez standard RSVP: <http://www.isi.edu/div7/rsvp/overview.html>. Użyteczne tylko w sieciach, które w całości kontrolujesz - 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 `1:1` i że wyselekcjonowany ruch chcesz wysłać do `1:1`.

## 13.1 Klasyfikator "u32"

Klasyfikator U32 jest najbardziej zaawansowanym z dostępnych filtrów, w obecnej implementacji. Jest w całości zbudowany w oparciu o tablice **mieszające** (ang. *hashing tables*), które zapewniają 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 mieć kilka filtrów (list reguł) z różnymi priorytetami. Każda lista będzie sprawdzana w kolejności w jakiej dodawano reguły, następnie lista z mniejszym priorytetem (wyższym numerem preferencji) będzie przetwarzana. 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 zaenkapsulowany 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 ffffffff00 at 16`. Chodzi o trzy-bajtowy test, poczynszyszy 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` czy `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 pppl4 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.

```
# tc filter add dev pppl4 parent 1:0 prio 10 u32 \
    match u8 0x10 0xff at nexthdr+13 \
    protocol tcp \
    flowid 1:3
```

FIXME: okazało się, że ta składnia nie działa poprawnie.

Użyj takiego zapisu by testować flagę ACK w pakietach mniejszych niż 64 bajty:

```
## match acks the hard way,
## IP protocol 6,
## IP header length 0x5(32 bit words),
## IP Total length 0x34 (ACK + 12 bytes of TCP options)
## TCP ack set (bit 5, offset 33)
# tc filter add dev pppl4 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
```

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.

## 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
```

Powyższa reguła będzie pasować do pakietów z ustawionym polem ToS 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 kernela. Prowadzi to do konkluzji, że selektory `tcp` i `udp` są dokładnie takie same i dlatego nie możesz użyć pojedynczej komendy `match tcp dst 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 dst 53 0xffff \
    match ip protocol 0x6 0xff \
    flowid 1:2
```

## 13.2 Klasyfikator "route"

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

```
# 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 rutingu 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 rutingu. Sztuczka polega na tym, by zdefiniować 'realm' bazując a adresie źródłowym lub docelowym. Można to zrobić w ten sposób:

```
# ip route add Host/Network via Gateway dev Device realm RealmNumber
```

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.

## 13.3 Filtry określające politykę

By umożliwić tworzenie jeszcze bardziej skomplikowanych konfiguracji, możesz zbudować 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 przesylu) 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 jako **najlepsza opcja** (ang. *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.

## 13.4 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 przyspieszyć 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: 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 wymaga 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 `1:`, 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!.

## 14. Parametry sieciowe kernela

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

Interesujące rzeczy są w `/proc/sys/net` i warto tam zajrzeć. Nie wszystko zostanie udokumentowane od razu, ale pracujemy nad tym.

(FIXME)

### 14.1 Filtrowanie trasy powrotnej

Domyślnie, routery rutują 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 zrobiłby 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 dotarłby do routera Linuksowego interfejsem `eth1` twierdząc, że pochodzi z podsieci ISP+Biurowej, zostałby odrzucony. Podobnie, pakiet z podsieci Biura twierdzący 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 w ruchu satelitarnym lub gdy masz trasy dynamiczne (z bgp, ospf, rip); dane przychodzą łączem satelitarnym a odpowiedzi wychodzą połączeniem naziemnym).

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 `syslog'a`.

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

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

## 14.2 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.

### 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 broadcast'owy spodziewasz się odpowiedzi od wszystkich komputerów. Jest to wygodne narzędzie DoS, ustawienie w tym pliku wartości `1' powoduje, że broadcast'owe wiadomości ICMP echo są ignorowane.

#### **/proc/sys/net/ipv4/icmp\_echo\_reply\_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 broadcast'owe.

#### **/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**

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.

#### **/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).

#### **/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`.

#### **`/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 parametrem - pętle routingu potrafią wyrządzić wiele kłopotu. Czasami nawet warto obniżyć tą wartość.

#### **`/proc/sys/net/ipv4/ip_dynaddr`**

Musisz ustawić tu ``1'` jeśli używasz dzwonienia-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 - techniki pozwalającej określić **Maksymalną Jednostkę Transmisji** (ang. *Maximum Transfer Unit*) możliwą na twojej trasie. Zajrzyj również do rozdziału o tym zagadnieniu w Książce Kucharskiej Linuksa.

#### **`/proc/sys/net/ipv4/ipfrag_high_thresh`**

Maksymalna ilość pamięci przeznaczona na fragmenty IP. Jeśli zaalokowano już `ipfrag_high_thresh` bajtów na ten cel, 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 zabierają 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 połączenie 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. Te 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 tę 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 przekompiłować 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 przypadku 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 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 łączy 1 gigabitowym połączenia mogłyby łatwo zacząć używać już używanych numerów sekwencyjnych, w związku z tym używa się dodatkowo znaczników czasu.

### **/proc/sys/net/ipv4/tcp\_tw\_recycle**

Włącz szybkie odzyskiwanie 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 prawdziwy interfejs, `all' lub `default'. Ostatnie znaczenie zmienia również ustawienia dla interfejsów, które będą stworzone w przyszłości.

#### **/proc/sys/net/ipv4/conf/DEV/accept\_redirects**

Jeśli ruter 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 tę 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 tę opcję IP.

#### **/proc/sys/net/ipv4/conf/DEV/bootp\_relay**

FIXME: wypełnić to

#### **/proc/sys/net/ipv4/conf/DEV/forwarding**

FIXME: wypełnić to

#### **/proc/sys/net/ipv4/conf/DEV/log\_martians**

Zajrzyj do sekcji wyżej.

#### **/proc/sys/net/ipv4/conf/DEV/mc\_forwarding**

Jeśli przekazujemy multicast'y 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!

#### **/proc/sys/net/ipv4/conf/DEV/rp\_filter**

Zajrzyj do sekcji o filtrowaniu trasy powrotnej.

#### **/proc/sys/net/ipv4/conf/DEV/secure\_redirects**

FIXME: wypełnić to

**/proc/sys/net/ipv4/conf/DEV/send\_redirects**

Jeśli wysyłamy wymienione wcześniej przekierowania.

**/proc/sys/net/ipv4/conf/DEV/shared\_media**

FIXME: wypełnić to

**/proc/sys/net/ipv4/conf/DEV/tag**

FIXME: wypełnić to

## **Polityka dotycząca sąsiadów**

`DEV' oznacza prawdziwy interfejs, `all' lub `default'. Ostatnie znaczenie zmienia również ustawienia dla interfejsów, które będą stworzone w przyszłości.

**/proc/sys/net/ipv4/neigh/DEV/anycast\_delay**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/app\_solicit**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/base\_reachable\_time**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/delay\_first\_probe\_time**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/gc\_stale\_time**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/locktime**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/mcast\_solicit**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/proxy\_delay**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/proxy\_qlen**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/retrans\_time**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/ucast\_solicit**

FIXME: wypełnić to

**/proc/sys/net/ipv4/neigh/DEV/unres\_qlen**

FIXME: wypełnić to

## **Ustawienia rutingu**

**/proc/sys/net/ipv4/route/error\_burst**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/error\_cost**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/flush**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/gc\_elasticity**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/gc\_interval**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/gc\_min\_interval**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/gc\_thresh**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/gc\_timeout**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/max\_delay**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/max\_size**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/min\_adv\_mss**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/min\_delay**

FIXME: wypełnić to

**/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**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/redirect\_number**

FIXME: wypełnić to

**/proc/sys/net/ipv4/route/redirect\_silence**

FIXME: wypełnić to

## 15. 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.

### 15.1 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 ich zastosowanie

**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`.

## 15.2 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  
Aplikacje czasu rzeczywistego w Sieci pakietowej zintegrowanych usług:  
Architektura i Mechanizm.

Jeśli dobrze rozumiem, główną ideą jest stworzenie potoków WFQ dla każdej gwarantowanej usługi i zaalokowanie reszty pasma sieciowego dla pasma `flow=0`. Będzie zawierał usługi przewidywalne oraz ruch najlepszego wysiłku a obsługiwany będzie przez planer priorytetów z pasmem najwyższego priorytetu zarezerwowanym dla usług przewidywalnych a resztę - dla usług najlepszego wysiłku.

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 zakłócenia."

Nie wygląda to na razie na dobrego kandydata, chyba, że przeczytałeś i zrozumiałeś powyższy artykuł.

## 15.3 DSMARK

Esteve Camps Chust <marvin@grn.es>

Ten tekst to streszczenie moich też zebranych w "Wsparcie dla QoS w Linuksie", wrzesień 2000.

Dokumenty źródłowe:

- [Draft-almesberger-wajhak-diffserv-linux-01.txt](#).
- Przykłady z dystrybucji `iproute2`
- [White Paper-QoS protocols and architectures](#) i [IP QoS Frequently Asked Questions](#) 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) z <http://www.ietf.org/html.charters/diffserv-charter.html> i <http://ica1www.epfl.ch/~almesber> (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 pomysle pojawiły się nowe architektury. Jedną z nich jest DiffServ, która zachowała bity ToS i przemianowane pole DS.

## Wskazówki dotyczące Zróżnicowanych Usług

Usługi Zróżnicowane są zorientowane na grupy. Oznacza to, że nie wiemy nic o potokach (którymi zajmą się Usługi Zróżnicowane); wiemy o agregacjach potoków i zastosujemy w stosunku do nich różne zasady, w zależności od tego, do której agregacji należy pakiet.

Kiedy pakiet dociera do węzła granicznego (węzła wejściowego do domeny DiffServ) można na nim zastosować politykę, 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 własną politykę 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 graniczne 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  $n \geq 0$ .
- **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 czytywane z pakietu i przechowywane w zmiennej `skb->tc_index`
- Wywołany jest klasyfikator. W wyniku jego wykonania otrzymujemy 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 którychś 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:

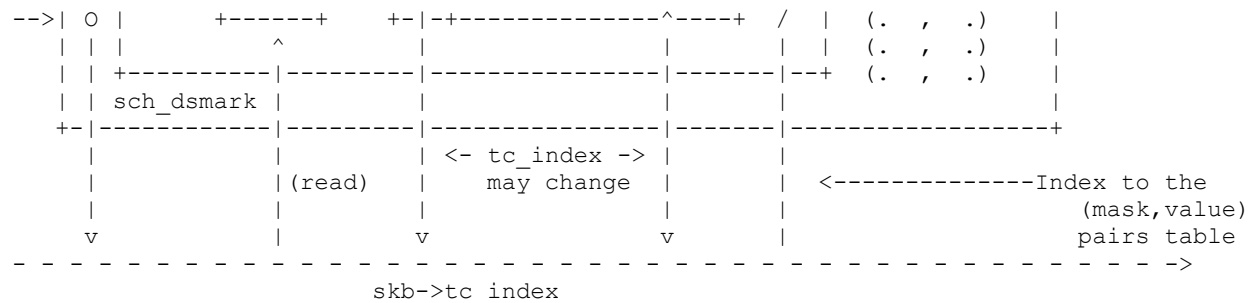
Nowe Pole DS = ( Stare Pole DS & maska ) | wartość

- 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:

```

                                skb->ihp->tos
----->
|
|  -- If you declare set_tc_index, we set DS
|     value into skb->tc_index variable
|
|
+--+  +-----+  +---+--+  Internal  +-+  +---N-----+-----+
| | |  | tc |---->| | |-->  . . . -->| | |  | D |  |  |
| | |  |----->| index |---->| | |  Qdisc |----->| v |  |
| | |  | filter|---->| | |  +-----+-----+  |----->(mask,value) |

```



Jak oznaczać 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 wytłumaczmy 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:

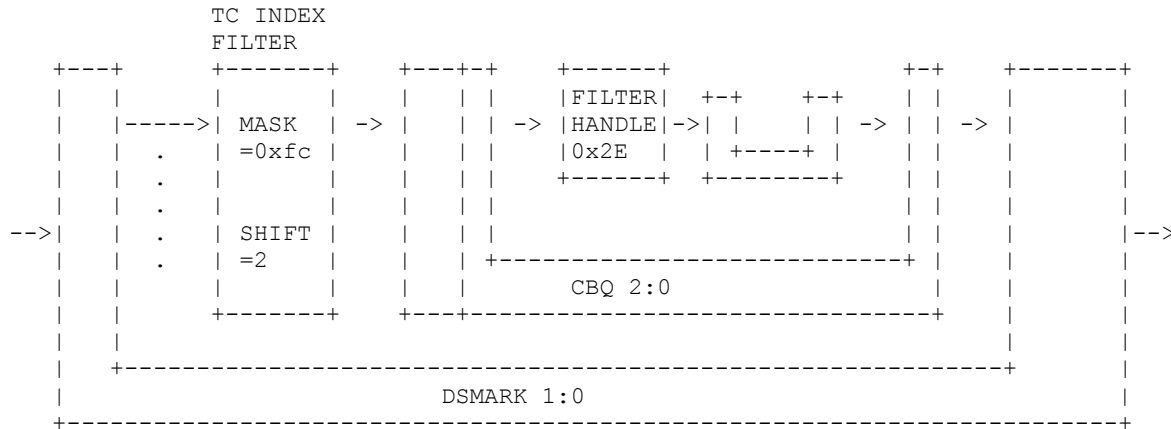
```
... tcindex [ hash ROZMIAR ] [ mask MASKA ] [ shift PRZESUNIĘCIE ]
[ pass_on | fall_through ]
[ classid CLASSID ] [ police OKREŚLENIE_POLITYKI ]
```

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, założmy 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 heksdecymalnie 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 następną operację:

```
Wartość1 = skb->tc_index & MASKA
Klucz = Wartość1 >> 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 do zwróconego wyniku, 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 uzupełnię 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.

## 15.4 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 stosować politykę do nadchodzącego ruchu nawet zanim dotrze do stosu IP.

### Parametry i zastosowanie

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 kolejki dla ruchu przychodzącego, zajrzyj do Książki Kucharskiej.

## 15.5 Losowe Wczesne Wykrywanie (ang.*Random Early Detection*, RED)

Ta sekcja jest rodzajem wprowadzenia do routingu na 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 ruterach w Internecie nazywa się **odrzucaaniem 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, nagle seria odrzuconych pakietów z rutera, który osiągnął swój limit wypełnienia, spowoduje serię opóźnionych retransmisji, które z kolei znów zapchają ruter.

By poradzić sobie z chwilowymi stanami zagęszczenia na łączach, w ruterach 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 bardziej 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 przy użyciu algorytmu RED znacząco ogranicza się szkodę powodowaną przepustowości lub opóźnieniom powodowanym innym aplikacjom/połączeniom.

RED odrzuca pakiety na podstawie statystyk dla przepływów, zanim ich liczba osiągnie górne ograniczenie. Powoduje to, że zagęszczenia na ruterach szkieletowych zwalniają przepustowości 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 od liczby przesyłanych pakietów.

RED jest dobrą kolejką dla szkieletów sieci, w których nie możesz pozwolić sobie 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 'przejszć 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 <http://www.aciri.org/floyd/papers/red/red.html> autorstwa Sally Floyd i Van Jacobson po więcej szczegółów technicznych.

## 15.6 Ogólne Losowe Wczesne Wykrywanie (ang.*Generic Random Early Detection*, RED)

Niewiele wiadomo o GRED. Wygląda jak GRED z wieloma kolejkami wewnętrznymi, a aktualna kolejka wewnętrzna wybierana jest na podstawie wartości pola TC\_INDEX Diffserv. Zgodnie ze slajdami znajdującymi się pod adresem <http://www.davin.ottawa.on.ca/ols/img22.htm>, GRED zawiera możliwości implementacji Cisco 'Dystrybuowane Wązone RED' jak również implementacji Dave Clark'a RIO.

Każda wirtualna kolejka może mieć swoje własne parametry odrzucania.

FIXME: Postarajmy się by Jamal lub Werner powiedział coś więcej.

## 15.7 Emulacja VC/ATM

Jest to duży projekt Wenera 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 adres <http://linux-atm.sourceforge.net/>.

## 15.8 Wązony Round Robin (ang.*Weighted Round Robin*, WRR)

Ta kolejka nie jest włączona do standardowych kerneli ale można ją ściągnąć spod 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 otrzymują 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 rosyłane do różnych maszyn, do różnych kolejek. Możesz używać adresu MAC lub IP i 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 osób współdzieli połączenie Internetowe. Zestaw skryptów do skonfigurowania zachowania algorytmu WRR dla takiej konfiguracji jest główną częścią dystrybucji WRR.

## 16. 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.

### 16.1 Praca z wieloma lokalizacjami z różnymi SLA

Możesz to zrobić na wiele sposobów. Apache oferuje wsparcie tego trybu modulem, ale pokażemy jak sam Linux może zrobić to dla ciebie oraz dla innych usług. Przykłady skradziono z prezentacji Jamal'a Hadi'ego, o którym wspomniano poniżej.

Założmy, że mamy dwóch klientów, obu z http, ftp i dźwiękiem transmitowanym strumieniami. Chcemy sprzedać im określoną część pasma sieciowego. Konfigurację 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`?

## 16.2 Ochrona komputera przed powodziami SYN

Z dokumentacji dla `iproute` Aleksieja, zaadoptowanej do `netfilter` i z większą ilością bardziej prawdopodobnych ścieżek. Jeśli tego użyjesz, weź pod uwagę dostosowanie numerów do sensownych wartości dla twojego systemu.

Jeśli chcesz chronić całą sieć, pomin ten skrypt zaprojektowany dla pojedynczego 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 konfigurację
#
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 przychodzącej
#$TC qdisc del $INDEV ingress
```

## 16.3 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:

```
[Internet] ---<E3, T3, cokolwiek>--- [ruter linuxowy] --- [Biuro+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 'fotek'!

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 100Kbit'ów. 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
```

## 16.4 Priorytetowanie ruchu interaktywnego

Jeśli masz wiele danych przechodzących przez łącze a chcesz zrobić coś za pomocą telnet'a 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 obrabiającego pole ToS określa to tak:

```
Dla mnie szczególnie ważny jest bit 'Minimalna Zwłoka'. Ustawiam go dla pakietów należących do sesji 'interaktywnych' w
ruterze przekazującym ruch wyżej. Łącze to 33k łącze modemowe. 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 ruterze 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 do 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
```

## 16.5 Przezroczyste cache'owanie przy użyciu netfilter, iproute2, ipchains i squid'a

Tą sekcję podesłał czytelnik Ram Narula z akcji Internet dla Edukacji (Tajlandia).

Standardową techniką by to osiągnąć w Linuksie jest użycie ipchains PO upewnieniu się, że ruch na port wychodzący 80 rutowany 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 ruter będący bramą domyślną to robi

Jeśli możesz poinstruować swój ruter będący bramą by pakiety pasujące do portu docelowego 80 były wysyłane na adres IP serwera ze squidem.

ALE

Spowoduje to dodatkowe obciążenie rutera, a niektóre routery komercyjne mogą tego nawet nie obsługiwać.

### Użycie przełącznika warstwy czwartej

Przełączniki warstwy czwartej mogą to obsługiwać.

ALE

Taki przełącznik kosztuje zwykle niemało. Zwykle oznacza to więcej niż koszt typowego rutera + dobrego serwera linuxowego.

### Użycie serwera cache'ującego jako bramy sieciowej

Możesz wymusić by CAŁY ruch przechodził przez serwer cache

ALE

Jest to trochę ryzykowne, ponieważ Squid zużywa całkiem dużo mocy przerobowej procesora co może spowodować spowolnienie całego ruchu sieciowego lub w ogóle załamanie się serwera i odcięcie całej sieci od Internetu w ogóle.

### Ruter 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 |
|-----|

Używane adresy
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) a jeśli nie stworzymy pętli 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 squidą i ipchains
```

Skonfiguruj serwer squidą na silom i upewnij się, że wspiera przeźroczyste cache'owanie/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
```

(zauważ: możesz mieć również inne wpisy)

Po więcej informacji jak ustawić Squidą zajrzyj na stronę FAQ Squidą: <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 konfigurację:

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
*ruch do portu 80 =====>(cache)
/\
||
\|=====kaosarn, RAS, etc.

```

Zauważ że sieć jest asymetryczna i mamy dodatkowy przeskok na ścieżce ruchu wychodzącego.

Poniżej opis przepływu pakietu przechodzącego sieć z kaosarn do i z Internetu.

Dla ruchu HTTP:  
 kaosarn wywołuje http->naret->silom->donmuang->internet  
 http odpowiada z internetu->donmuang->silom->kaosarn

Dla wywołań nie będących wywołaniami HTTP (np. telnet):  
 wychodzące dane kaosarn->naret->donmuang->internet  
 przychodzące dane z internetu->donmuang->kaosarn

## 16.6 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 części internetu wspierają 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 'Określanie MTU ścieżki', gdzie MTU oznacza **Maksymalną Jednostkę Transmisji** (ang. *Maximum Transfer Unit*).

Kiedy ruter otrzyma pakiet, który jest zbyt duży by zostać wysłanym w jednym kawałku i 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 odpowiedź by wysłać mniejsze pakiety, a przez konsekwentne pomniejszanie rozmiaru pakietu może znaleźć optymalny rozmiar pakietu dla połączenia przez tę 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 umierają po chwili.

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 kolejkovanie po drugiej stronie. Przy 296 odpowiedź na wciśnięcie klawisza jest w sensownych ramach czasowych. A po mojej stronie mam ruter prowadzący maskaradę na Linuksie. Ostatnio rozdzieliłem `serwer' i `ruter' 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 łączę 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, 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
```

## 16.7 Unikanie problemów z rozpoznaniem przez MTU trasy a Zaciskanie MSS(dla użytkowników kablowych ADSL, PPPoE i PPPtP)

Jak już wyjaśniono powyżej, rozpoznawanie trasy przez MTU nie działa już tak dobrze jak powinno. Jeśli wiesz na pewno, że **przejście** (ang.hop) gdzieś w twojej sieci ma ograniczone (<1500) MTU, nie możesz polegać na tym mechanizmie by to sprawdzić. Poza MTU, jest jeszcze inny sposób by ustalić maksymalny rozmiar pakietu, tzw.

**Maksymalny Rozmiar Segmentu** (ang. *Maximum Segment Size*). Jest to jedno z pól opcji TCP w pakiecie SYN.

Ostatnie kernele Linuksa i parę sterowników PPPoE (zwłaszcza sterownik Roaring Penguin'a) udostępnia możliwość 'Zaciśnięcia 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 oczywiście 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.

## 16.8 Najlepszy 'udrażniacz' dla ruchu sieciowego: małe opóźnienia, szybkie wrzucanie i ściąganie

Staralem 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 ściągnięcia cierpi również od gubionych pakietów co mocno odbija się na wydajności TCP/IP. Oczywiście duże kolejki mogą temu zapobiec i zwiększyć szybkość ściągnięcia. I w związku z tym tak zwykle dostawcy Internetowi tak konfiguruje 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 ruter 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, konfigurowujemy 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 ściągnięcie 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 opóźniane.

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 `udroźniacza' ruchu podczas pobierania danych:

round-trip min/avg/max = 560.9/573.6/586.4 ms

Bez `udroźniacza' ruchu podczas wysyłania danych:

round-trip min/avg/max = 2041.4/2332.1/2427.6 ms

Z `udroźniaczem', podczas wysyłania danych z prędkością 220kbit/s:

round-trip min/avg/max = 15.7/51.8/79.9 ms

Z `udroźniaczem', 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 zależy bardzo 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 przez 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** (ang.*Stochastic Fairness Queues*) by upewnić się, że wiele strumieni przenoszących dane `masowe' nie będą sobie przeszkadzać.

Ruch przychodzący podlega polityce wykonywanej przez filtr tc zawierających Filtr Wiadra Żetonów.

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/sh
```

```
# Konfiguracja dla połączenia internetowego w domu
```

```
#
```

```
#
```

```
# Ustaw poniższe wartości trochę poniżej faktycznych prędkości
```

```
# ściągania i wysyłania (w kilobitach)
```

```
DOWNLINK=800
```

```

UPLINK=220

# wyczyść kolejki dla wysyłania i ściągania danych, nie informuj
# o błędach
tc qdisc del dev ppp0 root 2> /dev/null > /dev/null
tc qdisc del dev ppp0 ingress 2> /dev/null > /dev/null

##### wysyłanie danych

# instalujemy w korzeniu CBQ

tc qdisc add dev ppp0 root handle 1: cbq avpkt 1000 bandwidth 10mbit

# kształtujemy wszystko na prędkości $UPLINK - zapobiega to tworzeniu
# się dużych kolejek na modemie DSL co zniszczyłoby panowanie nad
# opóźnieniami

# klasa główna

tc class add dev ppp0 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 ppp0 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 ppp0 parent 1:1 classid 1:20 cbq rate ${9*$UPLINK/10}kbit \
allot 1600 prio 2 avpkt 1000

# obie klasy kontrolowane są przez Sprawiedliwy podział stochastyczny:
tc qdisc add dev ppp0 parent 1:10 handle 10: sfq perturb 10
tc qdisc add dev ppp0 parent 1:20 handle 20: sfq perturb 10

# włączamy filtry
# Minimalna zwłoka w polu ToS (ssh, NIE scp) dla 1:10:
tc filter add dev ppp0 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 ppp0 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 wsadzamy do klasy interaktywnej
tc filter add dev ppp0 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 ppp0 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 ppp0 handle ffff: ingress

# filtrujemy do niego wszystko (0.0.0.0/0) i odrzucamy wszystko co
# przychodzi zbyt szybko:

tc filter add dev ppp0 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/sh

# 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 Sprawiedliwy podział stochastyczny:
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
# 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!

## 17. 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 czyni go prawdziwym przełącznikiem.

Mosty zwykle umieszcza się tam, gdzie trzeba poprawiać błędy konstrukcyjne sieci bez 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.

Inna dobra rzecz to 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 adresem <http://bridge.sourceforge.net/>.

## 17.1 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 manipulowania pakietami.

Istnieje obecnie wiele projektów zajmujących się tym, a jednym z lepszych jest projekt autorstwa kodu mostkującego dla Linuksa 2.4, Lennert Buytenhek. Lennert poinformował nas ostatnio, że od wersji bridge-nf 0.0.2 (spójrz wyżej po adres) kod jest stabilny i może być używany w środowiskach produkcyjnych. Jest teraz w trakcie uzgadniania włączenia łąty do kernela.

## 17.2 Mostkowanie i kształtowanie

Działa dokładnie tak jak się to opisuje. Upewnij się, że wiesz w którą stronę skierowany jest każdy interfejs, w innym przypadku możesz zacząć kontrolować ruch wychodzący na interfejsie wewnętrznym co nie będzie działało. Jeśli potrzebujesz, użyj tcpdump'a.

## 17.3 Pseudo-mosty i Proxy-ARP

Jeśli chcesz zaimplementować pseudo-most, pomini 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 działa inaczej, ponieważ zachowuje się bardziej jak ukryty ruter 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 łąty.

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 **Protokołu Rozwiązywania Adresów** (ang. *Address Resolution Protocol packet*), który brzmi 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.

## Uruchamianie

W zamierzonych 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 i stworzyć reguły pasujące dla proxy-ARP. Było to o tyle złe, że wymagało wiele pisania, co przekładało się wprost na ilość możliwych do popełnienia błędów powodujących wzbudzenie 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 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 naprawę chirurgicznego trasowania jak opisano powyżej, **bardzo** ważne jest sprawdzenie masek!

## 18. Ruting 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 i BGP4 (rfc1771). Linuks wspiera oba protokoły za pomocą programów `gated` i `zebra`.

O ile nie dotyczy to obecnie tego dokumentu, chciałbym skierować cię do pewnych dobrych źródeł:

Ogólnie:

Cisco Systems [Projektowanie dużych międzysieci IP](#)

Dla OSPF:

Moy, John T. "OSPF. Anatomia internetowego protokołu rutowania" Addison Wesley. Reading, MA. 1998.

Halabi napisał również dobry przewodnik o projektowaniu routingu OSPF ale najwyraźniej został on usunięty ze stron WWW Cisco.

Dla BGP:

Halabi, Bassam "Architektury rutowania internetu" Cisco Press (New Riders Publishing). Indianapolis, IN. 1997.

Oraz

Cisco Systems

[Użycie BGP dla routingu międzydomenowego](#)

O ile przykłady są specyficzne dla Cisco, są bardzo podobne do języka konfiguracji programu Zebra :-).

## 19. Inne możliwości

Ten rozdział wymienia projekty 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.

Implementacja 802.1Q VLAN dla Linuksa <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 adresem [ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-97/virtual\\_lans/index.htm](ftp://ftp.netlab.ohio-state.edu/pub/jain/courses/cis788-97/virtual_lans/index.htm). Dzięki tej implementacji, Linuks może wymieniać informacje z systemami takimi jak Cisco Catalyst, 3Com: {Corebuilder, Netbuilder II, SuperStack II switch 630}, Extreme Ntwks Summit 48, Foundry: {ServerIronXL, FastIron}.

Uaktualnienie: łąta została włączona do kernela 2.4.14.

### **Alternate 802.1Q VLAN Implementation for Linux** <http://vlan.sourceforge.net>

Alternatywna implementacja VLAN dla Linuksa. Projekt zaczął się od nieporozumień przy projekcie architektury VLANów i stylu kodowania i jest generalnie rzecz biorąc 'czyściej' napisany.

### **Wirtualny Serwer Linux** <http://www.LinuxVirtualServer.org/>

Ci ludzie są błyskotliwi. 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.

Po krótkce 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 posiadać 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 rutera a nie przez maszynę LVS, która w związku z tym nie potrzebuje nadzorować zawartości 5Gbit'owego strumienia danych wypływającego do świata a w związku z tym nie może być wąskim gardłem.

LVS zaimplementowano jako łąta na Linuksa 2.0 i 2.2 i jako moduł dla netfilter w 2.4/2.5 więc nie potrzebne są już łąty! Co prawda wsparcie dla 2.4 jest nadal we wczesnym stadium rozwoju ale komentarze i uwagi są mile widziane.

### **CBQ.init** <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 10mbit'owym 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 na przykład kształtowanie ruchu na podstawie czasu. Razem ze skryptem dostarczana jest dokumentacja, która wyjaśnia wszystko czego nie znajdziesz w README.

### Proste skrypty kształtowania ruchu Chronox <http://www.chronox.de>

Stephan Mueller (smueller@chronox.de) napisał dwa użyteczne skrypty, 'limit.conn' i 'shaper'. Pierwszy umożliwia łatwe ograniczenie pojedynczej sesji ściągania danych tak jak poniżej:

```
# limit.conn -s IP_SERWERA -p PORT_SERWERA -l LIMIT
```

Działa 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.

### Implementacja Protokołu Redunantnego Wirtualnego Rutera (Virtual Router Redundancy Protocol) <http://w3.arobas.net/~jetienne/vrrpd/index.html>

Stworzona całkowicie dla zapewnienia redundancji. Dwie maszyny ze swoimi własnymi adresami IP i MAC tworzą razem trzeci adres IP i MAC, który jest wirtualny. Oryginalnie protokół tworzone 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' zapobiegła temu.

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.

## 20. Dalsza lektura

<http://snafu.freedom.org/linux2.2/iproute-notes.html>

Zawiera wiele technicznych informacji z komentarzy kernela.

<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/>

HTML'owa wersja dokumentacji Aleksieja w LaTeXie - wyjaśnia ze szczegółami część iproute2.

<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.

### Usługi Rozróżniane na Linuksie

Ten dokument <ftp://icaftp.epfl.ch/pub/linux/diffserv/misc/dsid-01.txt.gz> autorstwa Wenera 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.

<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 trzeba by zapłacić za routery w cenie samochodów :-)

**Eksperymentalna strona Docum** <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**

Bardzo zalecana lektura, jeśli chcesz naprawdę zrozumieć TCP/IP. I całkiem zajmująca.

## 21. 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.

- Ron Brinker <service%emcis.com>
- Lennert Buytenhek <buytenh@gnu.org>
- Esteve Camps <esteve@hades.udg.es>
- Stef Coene <stef.coene@docum.org>
- Jonathan Corbet <lwn%lwn.net>
- Gerry Creager N5JXS <gerry%cs.tamu.edu>
- Marco Davids <marco@sara.nl>
- Jonathan Day <jd9812@my-deja.com>
- Martin Devera aka devik <devik@cdi.cz>
- Stephan "Kobold" Gehring <Stephan.Gehring@bechtle.de>
- Jacek Glinkowski <jglinkow%hns.com>
- Andrea Glorioso <sama%perchetopi.org>
- Nadeem Hasan <nhasan@usa.net>
- Vik Heyndrickx <vik.heyndrickx@edchq.com>
- Koos van den Hout <koos@kzdoos.xs4all.nl>
- Martin Josefsson <gandalf%wlug.westbo.se>
- Andi Kleen <ak%suse.de>
- Pawel 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>
- Jason Lunz <j@cc.gatech.edu>
- Stuart Lynne <sl@fireplug.net>
- Alexey Mahotkin <alexm@formulabez.ru>
- Andreas Mohr <andi%lisas.de>
- Andrew Morton <akpm@zip.com.au>
- Wim van der Most
- Stephan Mueller <smueller@chronox.de>

- Patrick Nagelschmidt <dto@gmx.net>
- Ram Narula <ram@princess1.net>
- Jorge Novo <jnovo@educanet.net>
- Patrik <ph@kurd.nu>
- Lutz Pressler <Lutz.Pressler%SerNet.DE>
- Jason Pyeron <jason%pyeron.com>
- Rusty Russell <rusty%rustcorp.com.au>
- Jamal Hadi Salim <hadi%cyberus.ca>
- 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>
- Andreas Steinmetz <ast%domdv.de>
- Jason Tackaberry <tack@linux.com>
- Charles Tassell <ctassell%isn.net>
- Glen Turner <glen.turner%aarnet.edu.au>
- Song Wang <wsong@ece.uci.edu>