

# GNU/Linux w roli routera - zarządzanie przepustowością

## 1. Przygotowanie systemu.

Przed przystąpieniem do instalacji należy zgromadzić sobie potrzebne pakiety zawierające kody źródłowe programów:

- źródła jądra Linux  
<ftp://ftp.kernel.org/>
- źródła netfilter, czyli popularne iptables:  
<http://www.iptables.org/>
- źródła pakietu iproute2 służącego do konfiguracji wszelakich parametrów sieciowych serwera:  
<http://developer.osdl.org/dev/iproute2/download/>
- łątkę IMQ na jądro i iptables:  
<http://www.linuximq.net/>
- źródła modułu do jądra i iptables:  
<http://www.ipp2p.org/>

W pierwszej kolejności przystępujemy do rozpakowania źródeł jądra i zaaplikowania na nie łątki IMQ:

Pobrane przez nas archiwum ze źródłami jądra nosi nazwę: linux-2.6.14.4.tar.bz2, a pobrany plik łątki IMQ na jądro to: linux-2.6.14-imq6.diff.

Wykonujemy więc:

```
# tar xjf linux-2.6.14.4.tar.bz2
# cd linux-2.6.14.4
# patch -p1 < ../linux-2.6.14-imq6.diff
```

Po tej operacji gotowi jesteśmy kompilacji jądra:

```
# make menuconfig
```

I przygotowujemy konfigurację. Interesujące nas opcje znajdują się w menu:

```
Networking ->
-> Networking options
```

Tutaj zainteresuje nas podmenu:

```
- QoS and/or fair queueing
```

a w nim opcje takie jak:

- Packet scheduler clock source: ustawiamy na "CPU cycle counter", tak aby zapewnić jak największą wydajność i rozdzielczość algorytmu zliczającego ruch sieciowy.
- HTB packet scheduler - nasz podstawowy mechanizm kolejkowania pakietów
- SFQ queue - algorytm kolejkowania, którego będziemy używali do utrzymania porządku pakietów wewnątrz utworzonych kolejek.

Wracając o poziom wyżej w menu konfiguracji jądra zwrócimy uwagę również na:

- Firewall based classifier
- U32 classifier

To opcje które pozwolą nam używać iptables oraz filtrów u32 jako metod klasyfikacji pakietów i przypisywania ich do wybranych kolejek. O tym za chwilę.

W głównym menu konfiguracji jądra zajrzemy jeszcze do:

- Device Drivers -> Network device support

Tu dodamy opcję: IMQ, oraz wpiszymy wartość opisaną jako "number of IMQ devices". Wartość ta określa maksymalną ilość urządzeń imq, które będą dostępne w systemie.

Po ustawieniu tych wartości i dodaniu wszystkich innych, niezbędnych nam opcji jądra uruchamiamy kompilację:

```
# make bzImage
# make modules
# make modules_install
```

Po zainstalowaniu modułów i pliku jądra możemy przystąpić do dalszej kompilacji potrzebnego nam oprogramowania.

Kolejnym, koniecznym elementem będzie tu odpowiednie przygotowanie iptables. Rozpakowujemy więc źródła iptables i również na nie aplikujemy łątkę IMQ. Nasze archiwum ze źródłami to: iptables-1.3.4.tar.bz2, zaś łątka IMQ to iptables-1.3.0-imq1.diff

```
# tar xjf iptables-1.3.4.tar.bz2
# cd iptables-1.3.4
# patch -p1 < ../iptables-1.3.0-imq1.diff
```

Po zaaplikowaniu łątki wykonujemy jeszcze dodatkową operację:

```
# chmod +x extensions/.IMQ*
```

która jest wymagana do poprawnej kompilacji dodatkowego modułu iptables, a następnie kompilujemy i instalujemy iptables.

Przy kompilacji dobrze dokonać dodatkowego wskazania na źródła jądra:

```
# make KERNEL_DIR=/usr/src/linux
```

o ile oczywiście /usr/src/linux to ścieżka pod którą kryją się źródła jądra przed chwilą używane przez nas do kompilacji.

```
# make install
```

Po tych operacjach jesteśmy gotowi do przeprowadzenia kompilacji rozszerzenia jądra i iptables wspomagającego detekcję ruchu typu peer-to-peer:

Rozpakowujemy więc źródła:

```
# tar xjf ipp2p-0.8.0.tar.bz2
# cd ipp2p-0.8.0
```

A następnie kompilujemy wskazując w jakich miejscach umieszczone są źródła jądra i iptables, które niedawno przygotowaliśmy:

```
# make KERNEL_SRC=/usr/src/linux IPTABLES_SRC=/usr/src/iptables-1.3.4
# make install
```

Otrzymane dwa pliki: `ipt_ipp2p.ko` - moduł dla jądra kopiujemy do katalogu: `/lib/modules/2.6.14.4` - gdzie 2.6.14.4 jest numerem kompilowanej przez nas wersji jądra i może ulegać zmianie, oraz plik: `libipt_ipp2p.so` - który jest modułem rozszerzającym funkcje iptables i należy go umieścić w katalogu przeznaczonym na takie właśnie pliki. Zwykle jest to `/usr/local/lib/iptables` lub w przypadku niektórych dystrybucji: `/usr/lib/iptables`.

W ten sposób wszystkie narzędzia mamy przgotowane, jeżeli wszystko zostało zainstalowane możemy zrestartować system i uruchomić nowe jądro.

## 2. Przykładowe konfiguracje.

Utwórzmy pierwszą przykładową konfigurację:

```
# Kasujemy stare reguły dotyczące interfejsu eth0
tc qdisc del root dev eth0

# Tworzymy kolejkę główną przypisaną do eth0 - identyfikator 1:0
tc qdisc add dev eth0 root handle 1:0 htb

# Tworzymy klasę ogólną dla interfejsu eth0 o identyfikatorze 1:1
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 2mbit ceil 2mbit

# Tworzymy klasy dla czterech komputerów - 1:10, 1:11, 1:12, 1:13
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 512kbit ceil 2mbit
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 512kbit ceil 2mbit
tc class add dev eth0 parent 1:1 classid 1:12 htb rate 512kbit ceil 2mbit
tc class add dev eth0 parent 1:1 classid 1:13 htb rate 512kbit ceil 2mbit

# Wskazujemy który ruch powinien trafić do wybranej klasy
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.0.10 flowid 1:10
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.0.11 flowid 1:11
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.0.12 flowid 1:12
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.0.13 flowid 1:13

# Zmieniamy mechanizm kolejkowania w obrębie klas na efektywniej działający
tc qdisc add dev eth0 parent 1:10 handle 10:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:11 handle 11:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:12 handle 12:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:13 handle 13:0 sfq perturb 10
```

W ten sposób cały ruch wychodzący z interfejsu eth0 będzie trafiał do kolejki głównej 1:1, która posiada ustawioną maksymalną przepustowość 2mbit/sek. Do kolejki głównej przypisujemy kolejki 1:10, 1:11, 1:12, 1:13 o rozmiarach 512kbit/sek, mogących

maksymalnie osiągać 2mbit/sek. Dzięki temu jeżeli pojawi się ruch w którejkolwiek z tych czterech kolejek, będzie on mógł zająć maksymalne pasmo 2mbit/sek, lecz w momencie stwierdzenia jakiegokolwiek transferu w innej kolejce, podział przepustowości będzie sprawiedliwy. W przypadku gdy wszystkie cztery kolejki będą zapełnione maksymalnie ruchem to będzie on wynosił tyle ile ustalona wartość parametru rate, a więc 512 kbit/sek.

W przypadku takowej konfiguracji należy zwrócić uwagę, aby suma parametrów 'rate' poszczególnych kolejek nie była większa od maksymalnego rozmiaru kolejki głównej. Może to zakłócić pracę mechanizmu HTB, lub wręcz uczynić go zupełnie bezradnym w kwestii podziału przepustowości.

Zastosowany algorytm SFQ pozwala na odpowiednie kształtowanie porządku ruchu wewnątrz danej klasy. Dzięki temu transmisja danych w poszczególnych strumieniach danych każdej z klas ma charakter wyrównany i sprawiedliwy.

W celu zapewnienia większej sprawiedliwości podziału wewnątrz poszczególnych klas (np. w przypadku gdy kilka różnych komputerów korzystających z połączenia sieciowego jest przypisane do tej samej klasy, czyli w przypadkach klasyfikacji po usługach sieciowych) możliwe jest użycie algorytmu ESFQ (<http://fatooh.org/esfq-2.6/>) który pozwala na podział sprawiedliwy nie tylko według strumieni danych, ale również według adresów IP docelowych, czy źródłowych.

W ten sposób mamy podzielony ruch przychodzący do komputerów - wychodzący z interfejsu eth0 naszego routera w stronę komputerów do niego przyłączonych. Aby zająć się ruchem nadchodzącym od komputerów w stronę internetu posłużymy się przygotowanym narzędziem - interfejsem IMQ.

```
# Ładujemy moduł odpowiedzialny za obsługę IMQ do jądra i nakazujemy mu
# utworzenie jednego urządzenia imq. Będzie to imq0.
/sbin/modprobe imq numdevs=1
```

```
# Analogicznie dla przypadku z ruchem wychodzącym z eth0, tutaj utworzymy
# bliźniacze reguły dla imq0.
```

```
# Kasujemy stare reguły dotyczące interfejsu imq0
tc qdisc del root dev imq0
```

```
# Tworzymy kolejkę główną przypisaną do imq0 - identyfikator 1:0
tc qdisc add dev imq0 root handle 1:0 htb
```

```
# Tworzymy klasę ogólną dla interfejsu imq0 o identyfikatorze 1:1
tc class add dev imq0 parent 1:0 classid 1:1 htb rate 2mbit ceil 2mbit
```

```
# Tworzymy klasy dla czterech komputerów - 1:10, 1:11, 1:12, 1:13
tc class add dev imq0 parent 1:1 classid 1:10 htb rate 512kbit ceil 2mbit
tc class add dev imq0 parent 1:1 classid 1:11 htb rate 512kbit ceil 2mbit
tc class add dev imq0 parent 1:1 classid 1:12 htb rate 512kbit ceil 2mbit
tc class add dev imq0 parent 1:1 classid 1:13 htb rate 512kbit ceil 2mbit
```

```
# Wskazujemy który ruch powinien trafić do wybranej klasy
tc filter add dev imq0 protocol ip parent 1:0 u32 match ip src 192.168.0.10 flowid 1:10
tc filter add dev imq0 protocol ip parent 1:0 u32 match ip src 192.168.0.11 flowid 1:11
tc filter add dev imq0 protocol ip parent 1:0 u32 match ip src 192.168.0.12 flowid 1:12
tc filter add dev imq0 protocol ip parent 1:0 u32 match ip src 192.168.0.13 flowid 1:13
```

```
# Zmieniamy mechanizm kolejkowania w obrębie klas na efektywniej działający
tc qdisc add dev imq0 parent 1:10 handle 10:0 sfq perturb 10
tc qdisc add dev imq0 parent 1:11 handle 11:0 sfq perturb 10
tc qdisc add dev imq0 parent 1:12 handle 12:0 sfq perturb 10
tc qdisc add dev imq0 parent 1:13 handle 13:0 sfq perturb 10
```

```
# Kierujemy ruch przychodzący do interfejsu eth0, czyli ruch
```

```
# przychodzący od komputerów do routera na interfejs imq0
# następującym poleceniem iptables:
iptables -t mangle -A PREROUTING -i eth0 -j IMQ --todev 0
```

W ten sposób obsłużyliśmy najprostszy, sprawiedliwy podział pasma dla czterech komputerów korzystających z symetrycznego łącza 2mbit/sek.

Sprawy o których zawsze musimy pamiętać to:

- identyfikatory kolejek typu: 1:0, 1:13, 10:0 powinny być unikalne w obrębie danego interfejsu. Możemy zatem posiadać niezależne kolejki o identyfikatorze 1:13 na interfejsie eth0, eth1, imq0, imq1.

Do tej pory każdy z przykładów zawierał zestaw klas przepustowości o jednakowym priorytecie. Wszystkie były traktowane na równi i zwolnienie części przepustowości przez jedną z kolejek sprawiało iż uzyskane w ten sposób wolne pasmo było rozprowadzane po wszystkich pozostałych.

Każdej z kolejek możemy nadawać różne wartości priorytetów, tak aby wybrae kolejki mogły mieć zawsze pierwszeństwo w kwestii rywalizacji o zwolnioną przepustowość przed innymi. W konfiguracji zapisujemy to podając dodatkowy parametr 'prio' i liczbę będącą wartością określającą priorytet kolejki. Im niższa wartość tym ważniejsza kolejka. Ostatni z przykładów definicji kolejek po modyfikacji o priorytet będzie wyglądał więc następująco:

```
# Tworzymy klasy dla czterech komputerów - 1:10, 1:11, 1:12, 1:13
tc class add dev imq0 parent 1:1 classid 1:10 htb rate 512kbit ceil 2mbit prio 3
tc class add dev imq0 parent 1:1 classid 1:11 htb rate 512kbit ceil 2mbit prio 2
tc class add dev imq0 parent 1:1 classid 1:12 htb rate 512kbit ceil 2mbit prio 2
tc class add dev imq0 parent 1:1 classid 1:13 htb rate 512kbit ceil 2mbit prio 1
```

W tej sytuacji kolejka posiadająca pierwszeństwo w oczekiwaniu na przydział pasma jest ostatnia kolejka 1:13 posiadająca priorytet 1. Na drugim miejscu traktowane na równi będą kolejki 1:11 i 1:12, a ostatnia wolne zasoby będzie otrzymywała kolejka 1:10.

W ten sposób tworząc listy kolejek o różnych priorytetach i przypisując im różne rodzaje ruchu (np. generowanego przez różnego typu usługi) możemy preferować rodzaj transferu jaki wypełnia nasze łącza do Internetu, np:

```
# Klasa dla ruchu WWW
tc class add dev eth0 parent 1:1 classid 1:20 htb rate 768kbit ceil 2mbit prio 2
# Klasa dla ruchu POP3 (pobieranie poczty z serwerow)
tc class add dev eth0 parent 1:1 classid 1:21 htb rate 1024kbit ceil 2mbit prio 1
# Pozostaly ruch - klasa domyslna
tc class add dev eth0 parent 1:1 classid 1:30 htb rate 512kbit ceil 2mbit prio 2

# Wskazujemy ktory ruch powinien trafiać do wybranej klasy
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip sport 80 0xffff flowid 1:20
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip sport 110 0xffff flowid 1:21
```

Zacytowany powyżej fragment skryptu konfiguracyjnego definiuje trzy klasy przepustowości:

- 1:20 równą 768 kbit/sek (max. 2 Mbit/sek) o priorytecie 2
- 1:21 równą 1024 kbit/sek (max. 2 Mbit/sek) o priorytecie 1, czyli ważniejszą

- 1:30 równą 512 kbit/sek (max. 2 Mb it/sek) o priorytecie 2, podobnie jak kolejka 1:20.

Do klasy 1:20 przypisany jest ruch ze źródłowym portem TCP równym 80, czyli zwykle jest to transfer nadchodzący z serwerów WWW, a do klasy 1:21 ruch nadchodzący z portu 110, czyli transfer poczty POP3. Jak łatwo wywnioskować w sytuacji takiej priorytet nad klasami 1:20 i 1:30 będzie miał ruch w klasie 1:21.

Powyższy przykład przyniósł również kolejną informację - klasyfikacja transmitowanych danych na podstawie numeru portów TCP za pomocą filtra U32 stosowanego we wcześniejszych przykładach do oznaczenia adresów IP źródłowych, bądź docelowych. Wówczas mieliśmy do czynienia z wpisami typu:

```
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.0.10 flowid 1:10
```

W przypadku chęci wskazania ruchu nadchodzącego ze źródłowego portu TCP 80 zapisujemy:

```
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip sport 80 0xffff flowid 1:20
```

czyli: "u32 match ip sport 80 0xffff" wskazuje na port 80. Wartość szesnastkowa znajdująca się po numerze portu równa 0xffff określa maskę bitową która w iloczynie logicznym ze wskazanym numerem portu decyduje o tym czy pakiety zostaną zakwalifikowane do tej reguły czy też nie. Wartość określająca numer portu jest wartością 16-bitową stąd wartość maski o takiej samej długości.

Dla wielu reguły filtrów U32 mogą okazać się zbyt skomplikowane, mało wygodne i niezrozumiałe. Warto pamiętać, że do klasyfikacji ruchu możemy również użyć filtra firewallowego iptables.

Wówczas zapisujemy np:

```
tc filter add dev eth0 protocol ip parent 1:0 handle 7 fw flowid 1:20
```

A za pomocą iptables tworzymy regułę:

```
iptables -t mangle -A PREROUTING -p tcp -sport 80 -j MARK --set-mark 7
```

Według tych reguł wszystkie pakiety TCP z portu 80 otrzymają znacznik '7', a filtr tc skieruje wszystkie pakiety posiadające znacznik 7 do klasy 1:20. Dzięki temu wystarczy nam jedna reguła filtra tc dla każdej kolejki: wskazanie znacznika, a samo klasyfikowanie wybranych rodzajów ruchu będzie się odbywało za pośrednictwem nadawania odpowiednich znaczników przy pomocy iptables.

Metoda ta przyda nam się również w kolejnym kroku, czyli uruchomieniu mechanizmu klasyfikacji ruchu generowanego przez programy wymiany plików w sieciach peer-to-peer.

### **3. Usługi Peer-to-Peer i ich wykrywanie.**

Klasyfikacji P2P dokonamy przy pomocy zainstalowanego wcześniej modułu ipp2p. Chcąc nadać odpowiedni znacznik pakietom należącym do tego typu połączeń wpisujemy do naszego skryptu następujące reguły:

```
iptables -t mangle -A PREROUTING -m ipp2p --ipp2p -j CONNMARK --set-mark 20  
iptables -t mangle -A PREROUTING -m connmark --mark 20 -j CONNMARK --restore-mark
```

Aby wszystkie elementy mogły ze sobą dobrze współpracować zastosowaliśmy dodatkową funkcję filtra pakietów iptables o nazwie CONNMARK, czyli znakowanie połączeń

(należy pamiętać o uaktywnieniu jej w trakcie kompilacji jądra systemu). Moduł ipp2p na podstawie analizy zawartości pakietów przetwarzanych przez jądro odnajduje specyficzne dla każdego z protokołów p2p elementy i na tej podstawie może zaznaczyć wybrany pakiet. Należy jednak pamiętać iż zaznaczenie pojedynczego pakietu inicjującego połączenie pomiędzy dwoma klientami sieci p2p może mieć niewielkie znaczenie ponieważ kolejne pakiety danych tego samego połączenia będą już miały charakter porównywalny np. z transferem FTP. Dlatego też na podstawie tego jednego charakterystycznego pakietu zaznaczamy połączenie w ramach którego pojawiły się dane identyfikujące je jako połączenie peer-to-peer. W ten sposób wszystkie dane wymieniane przez dwa komputery uczestniczące w sieci p2p będą posiadały wybrany przez nas znacznik i będzie można przeznaczyć je do przeznaczonej dla takiego rodzaju ruchu klasy przepustowości.

Instalacja i konfiguracja ipp2p jest więc jak widać sprawą dość prostą, a jej przydatność w zastosowaniu - ogromna. Innymi słowy: mała rzecz, a cieszy.

Warto również wspomnieć w tym momencie o filtrze pakietów działającym w warstwie aplikacji, dostępnym pod adresem: <http://l7-filter.sourceforge.net/>. Filtr ten reprezentuje nieco inne podejście do kwestii rozpoznawania protokołów. Jego zadaniem przede wszystkim jest rozpoznawanie różnego typu protokołów sieciowych warstwy IP i za jego pomocą możemy pokusić się o dodawanie własnych definicji klasyfikujących.

#### **4. Zakończenie**

Zapraszam do eksperymentowania z opisywanymi tu konfiguracjami. Podział przepustowości jest tematem bardzo szerokim, wszelkie sposoby wykorzystania przedstawionych tu narzędzi zależą tylko i wyłącznie od inwencji administratora. Pamiętajmy: każde narzędzie można wykorzystać na dwa sposoby - dobry i zły. Od was zależy, który wybierze.

#### **ZAPAMIĘTAJ:**

- wartości przepustowości, czyli 'rate' poszczególnych kolejek 'dzieci' nie mogą być nigdy większe niż 'rate' kolejki nadrzędnej - rodzica.

- zalecana wartość maksymalnej przepustowości 'ceil' powinna wynosić nieco poniżej górnej wartości maksymalnej przepustowości łącza. Strata kilku procent przepływności może okazać się niską ceną za zawsze krótkie czasy pingów, płynne działanie aplikacji interaktywnych (jak np. ssh), czy gier sieciowych.