# Kolejkowanie pasma w Linuxie

Autor: Piecuch Tomasz  IVFDS

# STRESZCZENIE

Kolejkowanie procesów ,które wysyłamy do sieci jest bardzo istotne w momencie gdy nasze łącze jest obciążone. Celem tego projektu jest przedstawienie sposobów zoptymalizowania przepływu danych w węźle obsługiwanym przez komputer z systemem operacyjnym Linux. 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. Filtrowanie pakietów w jądrze Linuxa ma długą historie począwszy od wersji 1.1 istnieje tam kod, który potrafi śledzić i kontrolować pakiety jakie pojawiają się w jego zasięgu. Jądra z serii 2.4 posiadają zupełnie przepisany kod dotyczący TCP/IP w stosunku do 2.2. Został też przepisany od nowa kod odpowiedzialny za filtrowanie pakietów. W tej chwili możliwościami filtracji pakietów nie ustępuje swoim dotychczas bardziej zaawansowanym konkurentom takim jak np. pakiet ipfilter, wywodzący się od systemów BSD.W opracowaniu tym zostaną tu przedstawione różne algorytmy przydziału pasma. Przedstawię sposób instalacji i zasadę działania algorytmu **CBQ** oraz jego następcy **HTB** – ta wersja jest bardziej zalecana działa szybciej i bardziej wydajnie. W tym doku męcie przedstawiam sposób instalacji programów niedokładnie zakładam że osoba czytająca to opracowanie zna podstawy Linuxa na tyle , że potrafi zainstalować program czy skompilować jądro systemu czy tworzenia maskarady przy użyciu kolędy `iptables` (są to osobne zagadnienia , które należy opanować przed przystąpieniem do instalowania algorytmów przydziału pasma).

# SPIS TREŚCI

# 1 PRZEGLĄD NAJPOPULARNIEJSZYCH DYDSCYPLIN KOLEJKOWANIA DOSTĘPU DO PASMA W LINUXIE

## 1.1 Bezklasowe dyscypliny (sposoby) kolejkowania

- **PFIFO_FAST** (najczęściej używana kolejka – domyślna) tradycyjna kolejka FIFO
- **Token Bucket Filter (TBF)** prosta kolejka z dyscyplin ą , która przepuszcza tylko dane przychodzące z pewną częstotliwością nie przekraczającą nałożonych ograniczeń
- **Sprawiedliwe Kolejkowanie Stochastyczne (SFQ)** to prosta implementacja rodziny algorytmów ze sprawiedliwym podziałem pasma. Jest mniej dokładna niż inne, ale wymaga również mniejszej ilości wyliczeń. Jest również samo konfigurowalna.

## 1.2 Dyscypliny kolejkowania z klasami

- **Class Based Queue(CBQ)** jest najbardziej skomplikowaną ponieważ algorytm CBQ nie jest zbyt precyzyjny i niezbyt pasuje do sposobu w jaki działa Linux. Równocześnie jest jednym z najpopularniejszych algorytmów przydziału pasma więc jest najczęściej używany. Niestety ten algorytm nie sprawdza się najlepiej przy dużym łączu , zaleca się jego używanie w przypadku sieci z kilkoma komputerami i wolnym łączem intrnetowym
- **Hierarchical Token Bucket (HTB)** została stworzona jako bardziej zrozumiała i intuicyjna alternatywa dla kolejki CBQ. Pozwala na kontrolę ruchu wychodzącego z serwera, symulacji kilku wolniejszych połączeń , jak również wysyłanie różnego rodzaju ruchu na różnych symulowanych po łączeniach. Doskonale sprawdza się w sieciach z większą ilością komputerów i stosunkowo szybkim łączem .
- **Kolejka PRIO -** nie zajmuje się tak naprawdę kształtowaniem ruchu, dzieli jedynie ruch na podstawie tego, jak zostały skonfigurowane filtry. Można ją traktować jak rozszerzoną kolejkę `pfifo_fast` (zamiast pasma jest osobna klasa zamiast prostej kolejki FIFO).

# 2 ALGORYTM HTB

## 2.1 WPROWADZENIE

Kolejka HTB pozwala na kontrolę ruchu wychodzącego z serwera, symulacji kilku wolniejszych połączeń, jak również wysyłanie różnego rodzaju ruchu na różnych symulowanych połączeniach. HTB określa sposób odwzorowania fizycznego połączenia w symulowane połączenia i decyduje, które z nich powinno być użyte dla określonego połączenia.

## 2.2 INSTALACJA OPROGRAMOWANIA

- W jądrach serii 2.4.x kolejka HTB nie jest jego integralną częścią, należy doinstalować jej obsługę własnoręcznie (dodać patcha na jądro)

  Należy znaleźć patcha – `htbx.x.x.x.diff` na sieci (http://luxik.cdi.cz/~devik/qos/htb/), przekopiować go do katalogu `/usr/src/` i wykonąć polecenia :

```
#mv linux linux-2.4      -zmieniamy nazwę katalogu ze źródłami Linuxa
#patch -p0 <htb3.5_2.4.17.diff   -dodajemy patcha do jądra
#rm -f htb3.5_2.4.17.diff    -usuwamy już niepotrzebnego patcha, który
                                 został już dodany do jądra i jest zbędny
```

Należy również uaktualnić skróty do plików nagłówkowych jądra w katalogu `/usr/include/`.Robimy to tak:

```
#cd /usr/include/          -ptrzechodzimy do katalogu include
#rm -rf asm linux scsi     -usuwamy stare linki do (asm , linux i scsi)
```

Teraz tworzymy nowe linki :

```
#ln -s /usr/src/linux-2.4/include/asm-i386 asm
#ln -s /usr/src/linux-2.4/include/linux linux
#ln -s /usr/src/linux-2.4/include/scsi scsi
```

Na koniec „sprzątamy" pliki pozostawione przez programistów jądra i uruchamiamy program konfiguracyjny.

```
#cd /usr/src/linux-2.4/
#make mrproper
#make menuconfig
```

- Teraz w menuconfig wybieramy opcje niezbędne dla działania HTB:

Najpierw wchodzimy do zakładki
Networking options ---> IP: Netfilter Configuration --->
A tu wybieramy wszystkie opcje

Teraz przechodzimy do zakładki :
Networking options ---> QoS and/or fair queueing --->
A tu wybieramy opcje zaznaczone poniżej [*] :

**[*] QoS and/or fair queueing**
[*] CBQ packet scheduler
**[*] HTB packet scheduler**
[ ] CSZ packet scheduler
[*] The simplest PRIO pseudoscheduler
[ ] RED queue
[*] SFQ queue
[ ] TEQL queue
**[*] TBF queue**
[ ] GRED queue
[*] Diffserv field marker
[*] Ingress Qdisc
**[*] QoS support**
**[*] Rate estimator**
[*] Packet classifier API
**[*] TC index classifier**
[*] Routing table based classifier
[*] **Firewall based classifier**
**[*] U32 classifier**
[*] Special RSVP classifier

- Teraz po wybraniu odpowiednich opcji kompilujemy jądro i ustawiamy je jako jądro domyślne dla systemu jak robiliśmy to dla algorytmu CBQ (opis poniżej)
- Do naszego systemu musi być zainstalowany pakiet **iproute2** (w Debianie instalujemy go poleceniem `apt-get install iproute` ). Jeśli nie używamy Debiana należy dodać patcha na jądro ,który obsługuje to polecenie. Patcha możemy znaleźć pod adresem : ftp://sunsite.icm.edu.pl/pub/linux/iproute/. Po rozpakowaniu źródeł przechodzimy do tego katalogu i wydajemy polecenie:
  patch –p1 < htb..._tc.diff
  Teraz mamy obsługę **iproute2** zaimplemętowaną w naszym jądrze

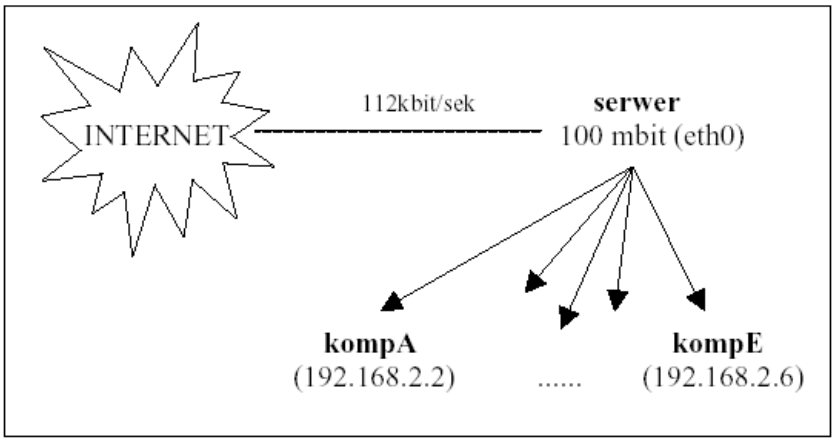
- Teraz pozostało nam zainstalować bardzo przydatne narzędzie do kontrolowania szybkości sieci (nie jest ono konieczne ale bardzo przydatne), jego obsługa jest bardzo prosta. Można je znaleźć pod adres: ftp://ftp.arl.mil/pub/ttcp/.

  Instalacja :

  - należy rozpakować plik ttcp.tar.gz i wydać polecenie: `gcc ttcp.c -o ttcp`
  - teraz wystarczy w tym katalogu znaleźć plik binarny `ttcp` i skopiować ją do katalogu np. /usr/local/bin/

## 2.3 Konfiguracja

Ponieważ nie jestem w stanie przedstawić wszystkich kombinacji sieci , zakładam że sieć wygląda tak:




Serwer połączony jest poprzez SDI do Internetu (112kbit/sek) poprzez interfejs ppp0 oraz do sieci lokalnej (100mbit/sek) poprzez interfejs eth0. Dodatkowo serwer pełni funkcję serwera plików dla LAN.Sieć lokalna lokalna składa się z 5 komputerów.

```
#!/bin/sh
```
**-standardowe wywołanie powłok sh (tak w linuxie musi zaczynać się każdy skrypt)**

```
tc qdisc del root dev eth0
```
**- kasowanie poprzednich ustawień kolejki**

```
tc qdisc add dev eth0 root handle 1:0 htb
```
**- założenie głównej kolejki na interfejs** `eth0`

**Utworzona zostaje klasa 1:1, która przejmuje całe pasmo interfejsu eth0. Przypominam, że 100mbit = 100000kbit.(nie wpisujemy tu całej wartości bo rzeczywiście nasza sieć nie pracuje z tą prędkością)**

```
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 99000kbit ceil 99000kbit
```

Teraz należy wyjaśnić składnie poleceń :

- tc: program pochodzi z pakietu iproute2. Jest narzędziem do konfiguracji kontroli przepływu danych (**T**raffic **C**ontrol).
- qdisc: (ang. q*ueueing* ***disc****ipline*) można przetłumaczyć jako sposób kolejkowania, określona dyscyplina której podlega kolejka.
- del: usuń
- add: dodaj
- dev: (ang. *device*) urządzenie, tutaj: interfejs, karta sieciowa
- root: tutaj: główna, „kolejka – rodzic”
- handle: uchwyt, tutaj: identyfikator kolejki
- 1:0: w tym przypadku identyfikatorem głównej kolejki jest 1:0
- htb: to nasz algorytm przydziału pasma

**Przepustowość interfejsu eth0 została podzielona na dwie klasy. Pierwsza z nich (1:2) przeznaczona jest na SDI czyli połączenie z Internetem. Druga natomiast (1:3) przeznaczona jest na przerzucanie plików po sieci lokalnej i inne usługi serwera dla LAN.**

```
tc class add dev eth0 parent 1:1 classid 1:2 htb rate 110kbit ceil 110kbit
tc class add dev eth0 parent 1:1 classid 1:3 htb rate 98000kbit ceil 98000kbit
```

**Teraz przydzielamy każdemu komputerowi pasmo gwarantowane – 20kbit ,które może być powiększone do 100kbit jeśli inni użytkownicy nie wykorzystują swojego łącza całkowicie**

```
tc class add dev eth0 parent 1:2 classid 1:4 htb rate 20kbit ceil 100kbit prio 2
tc class add dev eth0 parent 1:2 classid 1:5 htb rate 20kbit ceil 100kbit prio 1
tc class add dev eth0 parent 1:2 classid 1:6 htb rate 20kbit ceil 100kbit prio 2
tc class add dev eth0 parent 1:2 classid 1:7 htb rate 20kbit ceil 100kbit prio 2
tc class add dev eth0 parent 1:2 classid 1:8 htb rate 20kbit ceil 100kbit prio 2
```

Teraz należy wyjaśnić składnie poleceń :

- class: klasa, wykorzystywana przy podziale pasma, głównej kolejki na mniejsze. Ponieważ na razie nie mamy żadnej klasy, w tym miejscu tworzona jest główna klasa, która otrzymuje pasmo swojego rodzica.
- parent: wskazuje rodzica klasy. Argumentem jest identyfikator, uchwyt nadrzędnej kolejki lub klasy.
- 1:0: w tym przypadku rodzicem dla klasy głównej jest kolejka o identyfikatorze 1:0. Czyli tworzymy klasę, która będzie dzielić, zarządzać pasmem swojego rodzica.
- classid: identyfikator klasy. Przez niego będziemy odwoływać się do danej klasy.
- 1:1: to jest właśnie identyfikator aktualnie dodawanej klasy.
- rate: jego argumentem jest minimalna, gwarantowana szybkość klasy.
- 10mbit: w naszym przypadku oznacza to, że klasa 1:1 ma zagwarantowaną, minimalną szybkość 10 megabitów.
- ceil: maksymalna szybkość, jaka dostępna będzie dla danej klasy.
- 10mbit: w naszym przypadku jest to maksymalna przepustowość dla kolejki 1:1. Domyślnie i tak wartość rate = ceil, jednak chcę uniknąć nieporozumień, w ten sposób łatwiej zapanować nad rozrastającym się drzewem klas (o tym później).
- prio możemy zdefiniować , który komputer ma pierwszeństwo do łącza jeśli istnieje jeszcze coś do przydziału ,jeśli któryś z komputerów nie wykorzystuje całego przydzielonego dla siebie łącza. Im mniejsza liczba tym wyższy priorytet

**Teraz należy filtrować pakiety przychodzące do nas. Jeśli nadawcą pakietów jest serwer (192.168.2.1) , to dane trafiają do części przeznaczonej na ruch w sieci lokalnej, czyli 98mbit (1:3).Pozostałe filtry pakietów, które wychodzą przez eth0 serwera, ale ich nadawcą nie jest serwer (czyli pakietach z Internetu) trafiają do odpowiednich klas o ograniczonej już przepustowości. W przypadku filtrów jest podobnie jak w przypadku** `ipchains / iptables` **– wygrywa pierwszy pasujący filtr. Czyli: jeśli coś pochodzi od serwera, idzie na 98mbit. Reszta musi pochodzić z Internetu i trafia do odpowiednich klas. Teoretycznie. Aby mieć całkowitą pewność, że tak będzie, należy dopisać między** `protocol ip` **a** `parent 1:0` **w filtrach .Przy pierwszym należy dopisać** `preference 1`**, przy pozostałych** `preference 2`**. Bez** `preference` **też powinno to tak działać, „wygrywa pierwsza pasująca reguła", ale lepiej mieć pewność.**

```
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip src 192.168.2.1 flowid 1:3
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.2.2 flowid 1:4
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.2.3 flowid 1:5
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.2.4 flowid 1:6
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.2.5 flowid 1:7
tc filter add dev eth0 protocol ip parent 1:0 u32 match ip dst 192.168.2.6 flowid 1:8
```

**Kilka informacji o kolejkowaniu w filtrach. Jak możesz zauważyć, filtry decydują o tym, które pakiety trafią do której klasy. Obsługiwane są na zasadzie FIFO. Jeśli zależy Tobie na tym, aby dodatkowo pakiety trafiające do filtra były obsługiwane na zasadzie „wszystkim po równo", należy jeszcze dodać do każdej klasy dodatkowy filtr: SFQ. Dokładnie mówiąc HTB zapewnia podział pasma: nikt nie dostanie więcej, niż jest to zdefiniowane. W naszym przypadku użytkownik kompA może się jednocześnie np. używać FTP i WWW .Jeśli ma dobre połączenie z FTP'em, sam sobie może uniemożliwić używanie WWW. Po stronie HTB jest wszystko porządku (szybkość będzie ograniczona), ale możemy i tak nie dopuścić do takiej sytuacji korzystając dodatkowo z filtra SFQ. Obsługuje on różne typy połączeń wrzucone do jednej docelowej klasy (tu 1:2) po kolei, co uniemożliwi w naszym przypadku przyblokowanie WWW przez ftp. Czyli SFQ sprawiedliwie obsługuje wszystkie połączenia wrzucone do jednego wora (czyli klasy).**

```
# sprawiedliwy dostęp do kabla wielu jednoczesnych połączeń
tc qdisc add dev eth0 parent 1:3 handle 3:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:4 handle 4:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:5 handle 5:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:6 handle 6:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:7 handle 7:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:8 handle 8:0 sfq perturb 10
```

## 2.4 USTARIENIA KOŃCOWE SERWERA

Teraz wystarcz tylko stworzyć skrypt startowy dla HTB przykładowo `rc.htb` , nadać mu atrybuty pliku wykonywalnego i dopisać jedną linie w pliku startowym `rc.init1` w katalogu `/etc/rc.d` tz. Należy dopisać :

```
/etc/rc.htb
```

W pliku `rc.htb` należy wpisać wszystkie komendy wymienione powyżej w punkcie **2.3Konfiguracja** w tej samej kolejności

# 3 ALGORYTM CBQ

## 3.1 WPROWADZENIE

Sam algorytm CBQ jest jednym ze starszych , które były używane do dzielenia łącza. Jednocześnie jest skomplikowany i niezbyt prosty do zaimplementowania w Linxie .Wymaga ustawienia firewalla i szeregu opcji konfiguracyjnych .Jest on wciąż bardzo popularny i często używany na wielu serwerach .Zadanie instalacji możemy bardzo uprościć używając oprogramowania już istniejącego – np.SHERPER.

Po włączeniu się w sieć nowego użytkownika i rozpoczęciu transmisji z internetem (lokalne połączenia nie rezerwują łącza), algorytm testuje czy użytkownik nie łączy się przypadkiem z jednym z adresów ip lub portów wpisanych w liście wyjątków (plik /etc/shaper/ignore). Jeśli nie znajdzie takiego wyjątku to przydziela część łącza wynikającą z prostego podziału szybkości maksymalnej łącza przez ilość pracujących w sieci komputerów.

Po określonym czasie (zależnie od ustawienia w cronie) - np. po 1 minucie – program (tu użyjemy shaper) sprawdza, czy użytkownik wykorzystał co najmniej 50% z przydzielonego łącza. Jeśli nie wykorzystał to zostaje mu przydział zmniejszony o 25%. Jeśli użytkownik wykorzystuje przydział w granicach 50%-75% to skrypt niczego nie zmienia a jeśli wykorzystuje więcej od 75% to shaper zwiększa mu transfer o minimalny transfer (np o 8000 czyli 1KBajt) - pod warunkiem, że coś jeszcze zostało do przydzielenia. Zasada jest taka, że w pierwszej części skrypt sprawdza, komu można obciąć a dopiero na samym końcu z tego co zostanie z przydzielania szybkości dodaje tym co dużo ciągną.

Dzięki temu CBQ zapobiega przejęciu całkowitej kontroli nad łączem przez jeden komputer.

## 3.2 INSTALACJA OPROGRAMOWANIA

### 3.2.1 Oprogramowanie

- Do naszego systemu musi być zainstalowany pakiet **iproute2** (w Debianie instalujemy go poleceniem `apt-get install iproute` ).Jeśli nie używamy Debiana należy dodać patcha na jądro ,który obsługuje to polecenie.
  Patcha możemy znaleźć pod adresem : ftp://sunsite.icm.edu.pl/pub/linux/iproute/.
  Po rozpakowaniu źródeł przechodzimy do tego katalogu i wydajemy polecenie:
  patch –p1 < htb..._tc.diff
  Teraz mamy obsługę **iproute2** w naszym jądrze.
- Jeszcze jedna , ważna rzecz na naszym serwerze musimy mieć zainstalowanego firewala wraz z maskaradą najlepiej używając `iptables` .

- Teraz przystępujemy do instalacji programu (znajdziemy go pod adresem : http://www.klicek.one.pl/includes/download/down.php?id=linux/programy/shaper_cbq_iptables.tar.gz)
- Mając spełnione wszystkie warunki przystępujemy do instalacji programu , najpierw musimy rozpakować program, następnie otrzymamy kilka katalogów, których zawartość kopiujemy do głównych katalogów w naszym systemie operacyjnym zachowując ścieżki dostępu tak jak w archiwum.

  Aby rozpakować plik wydajemy polecenie

```
tar -zxvf shaper_cbq_iptables.tar.gz
```

### 3.2.2 Jądro systemu

- Aby algorytm zadziałał musimy mieć wkompilowane do naszego jądra (serii **2.4.x**) z obsługą CBQ (możemy to sprawdzić poleceniem `tc -d qdisc`, jeśli polecenie wyświetli jakieś błędy to należy przekompilować jądro).
- Krótki opis kompilacji jądra jeśli dostaniemy błąd (nie podaje szczegółów ,jak kompiluje się jądro i co należy zaznaczyć w opcjach, nie jest to temat tego dokumentu)

```
#cd /usr/src/linux
#make menuconfig
```

  Teraz w zakładce **Networking Options** wybieramy zakładkę **QoS and/or fair queuing**, a tam zaznaczamy wszystkie możliwości (zalecane jest wkompilowanie wszystkiego do jądro a nie jako moduły) .Pozostałe elementy pozostawiamy tak, jak były lub zmieniamy wzależności od potrzeb i sprzętu.

```
#make dep
#make clean
#make bzimage
#make modules
#make modules_install
```

  Teraz tylko musimy ustawić nasz system operacyjny tak aby uruchamiał się z naszego nowego jądra.

## 3.3 KONFIGURACJA PROGRAMU

### 3.3.1 Ustawienia skryptów firewalla

- W pliku, gdzie mamy wpisane reguły maskarady dopisujemy regułki , które rejestrują ilość danych pobraną z internetu przez poszczególnych użytkowników z interfejsu ppp

```
iptables -N ppp-in
iptables -A INPUT -i ppp -j ppp0-in

iptables -A FORWARD -i ppp -s ! 192.168.11.1 -d 192.168.11.2 -j RETURN
iptables -A FORWARD -i ppp -s ! 192.168.11.1 -d 192.168.11.3 -j RETURN
iptables -A FORWARD -i ppp -s ! 192.168.11.1 -d 192.168.11.4 -j RETURN
```

ppp – to nazwa nasze połączenia z internetem

192.168.11.1 to IP serwera a pozostałe IP to adresy komputerów w naszej sieci

Teraz możemy sprawdzić czy nasze wpisy są prawidłowe :

```
iptables -vxL FORWARD -n | awk '{print $9,$2}'
```

Efektem tej komendy powinna być informacja o tym ile każdy użytkownik sieci przesyła danych jeśli wartości są różne od zera to znaczy że wszystko działa .

### 3.3.2 Konfigurujemy parametry łącza lokalnego

pliku **/etc/shaper/shaper.cfg** wpisujemy

```
mainip=192.168.14.99
high_start_speed=1
local_int=eth0;10485760;192.168.0.0/16;192.168.11.0/24;10
```

**mainip** numer ip zewnętrzny serwera

**local_int** parametry lokalnych interfejsów

**eth0** - nazwa interfejsu. Nie wpisuj tutaj interfejsu łączącego z internetem!

**10485760** - szybkośc interfejsu w bitach/sek (10MBit=10*1024*1024 bit)

**192.168.0.0/16** - numer ip interfejsu (w tym przypadku jest to maska sieci klasy C co powoduje, że wszystkie transfery od numerów ip pasujących do tej maski nie będą miały ograniczeń. Chodzi o to aby transfer lokalny z serwera bądź rutowany przez interfejsy od ludzi lokalnych w sieci nie był przycinany).

**192.168.11.0/24** - numer ip docelowy z maską dla którego stosowany jest shaper (czyli maska podsieci stosowana na danym interfejsie - np eth0 może mieć 192.168.1.0/24 a eth1 192.168.2.0/24 - nuery ip w podsieci to 192.168.2.0-255)

**10** - identyfikator klasy (starszy bajt) (dla każdego interfejsu musi być inny - najlepiej kolejny po 10).

**high_start_speed** - sposób przydzielania łącza przy pojawieniu się nowego numeru ip. Dla 1 początkowa szybkość jest równą częścią szybkości łącza wynikłą z podziału szybkości przez ilość userów aktualnie z niego korzystających (np. dla SDI z 5 pracującymi userami = 100000/5 czyli 20000 (kbitów)). Dla 0 przydzielona szybkość jest minimalna (taka jak podana w parametrze przy uruchomieniu shapera)

W przypadku , gdy w sieci mamy kilka interfejsów (każda mała podsieć jest wpięta do serwera na innej karcie sieciowej – ma inny adres) sherper.cfg ma postać(dla 3 interfejsów):

```
mainip=212.17.14.118
high_start_speed=1
local_int=eth0;10485760;192.168.0.0/16;192.168.1.0/24;10
local_int=eth1;104857600;192.168.0.0/16;192.168.2.0/24;11
local_int=eth1;104857600;192.168.0.0/16;192.168.3.0/24;12
```

Jeżeli mamy w skrypcie z maskaradą rozpoznawanie użytkowników i zezwalanie na połączenie to możemy ten podpunkt pominąć. Jeśli nie to musimy wpisać wszystkie lokalne numery IP do pliku **/etc/shaper/iplist** np.

```
192.168.11.2=eth0
192.168.11.3=eth0
192.168.11.4=eth0
```

### 3.3.3 Obsługa aplikacji ,które nie obciążają łącza

Istnieje możliwość przyśpieszenia łącza dla aplikasji , które praktycznie nie zajmują łącza jak GaduGadu , Telnet czy SSH(bardzo przydatne przy administracji go nie musimy czekać na odpowiedz serwera zbyt długo). Określamy je w /ets/sherper/ w sposób : 22$ - dla portu 22 (SSH) niema ograniczeń.

### 3.3.4 Uruchamianie programu

Teraz mamy już wszystko gotowe jeszcze trzeba uruchamiać nasz sherper cyklicznie co minutę (oczywiście nie jest to dokładnie określony czas sam administrator powinien go określić , jednak zaleca się właśnie taki okres czasu).Najlepiej dopisać do crona (w katalogu /etc/ znajduje się albo plik **crontab** )taką linijkę:

```
*/1 * * * * root /sbin/shaper 115200 8000
```

UWAGA jeśli na naszym serwerze mamy zainstalowanego serwer Achache i obsługuje on skrypty PHP ,to uruchamiam go tak:

```
*/1 * * * * root /usr/bin/php -q /sbin/shaper.php 100000 8000 > /dev/null
```

### 3.3.5 Sprawdzenie

Możemy kontrolować stan pracy naszego serwera za pomocą polecenia

```
tc -s qdisc
```

Jego wyniki powinny wyglądać mniej więcej tak:

```
qdisc tbf d09f: dev eth0 rate 5430bps burst 10Kb lat 1.2s
 Sent 108930 bytes 73 pkts (dropped 0, overlimits 0)

 qdisc tbf d09e: dev eth0 rate 5430bps burst 10Kb lat 1.2s
 Sent 126618 bytes 89 pkts (dropped 0, overlimits 0)
```

# MATERIAŁY POMOCNICZE

[1] Kształtowanie Ruchu i Zaawansowany Ruting HOWTO
[2] CBQ micro-HOWTO
[3] „Filtrowanie stateful–inspection w Linuksie i BSD" Paweł Krawczyk – dokument PDF
[4] Internet