OpenBSD. Tworzenie firewalli za pomocą PF

Autor: Jacek Artymiak
Tłumaczenie: Marek Pętlicki (wstęp, rozdz. 1 – 7), Mateusz Michalski (rozdz. 8 – 16, dod. A – E)
ISBN: 83-7361-680-2
Tytuł oryginału: Building Firewalls with OpenBSD and PF
Format: B5, stron: 272

Tajniki zabezpieczania systemu OpenBSD

- Konfigurowanie OpenBSD
- Projektowanie firewalli
- Definiowanie reguł filtrowania pakietów

OpenBSD cieszy się sławą jednego z najbezpieczniejszych systemów operacyjnych wśród obecnie dostępnych. Twórcy systemu skoncentrowali się głównie na zapewnieniu najwyższego poziomu zabezpieczeń. Działa on na wielu różnych platformach sprzętowych, w tym również na standardowym komputerze Intel PC (i386), komputerach Macintosh (zarówno z procesorami serii MC68xxx, jak i z procesorami Power PC), Sun Sparc, Compaq Alpha i wielu innych. Twórcy systemu OpenBSD przywiązują dużą wagę do poprawności kodu, zakładając, że zmniejsza to prawdopodobieństwo nieprawidłowego działania systemu i podnosi poziom bezpieczeństwa.

Książka „OpenBSD. Tworzenie firewalli za pomocą PF" przedstawia zaimplementowany w OpenBSD system PF — unikatowe narzędzie do samodzielnego definiowania mechanizmów zabezpieczeń filtrujących pakiety. Opisuje metody filtrowania pakietów i sposoby ich konfigurowania. Prezentuje również zaawansowane możliwości systemu PF — translację adresów sieciowych i zarządzanie przepustowością oraz narzędzia pozwalające na kontrolowanie jego pracy.

- Rodzaje firewalli
- Instalacja i konfiguracja OpenBSD
- Struktura pliku konfiguracyjnego pf.conf
- Normalizacja i przeadresowywanie pakietów
- Reguły filtrowania pakietów
- Zarządzanie firewallem
- Wzorce reguł dla typowych konfiguracji firewalli

Jeśli chcesz, aby administrowany przez Ciebie system był nie do zdobycia — przeczytaj tę książkę.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 10.

Zarządzanie obciążeniem pasma i równoważenie obciążenia

Które z pakietów są ważniejsze niż inne? W jaki sposób równoważenie obciążenia może pomóc obleganym serwerom? Jak powstrzymać użytkowników przed zapychaniem łącza?

Równoważenie obciążenia i zarządzanie obciążeniem pasma pozwalają uniknąć przeładowania sieci. *pf(4)* implementuje oba rozwiązania poprzez tzw. opcje puli, które są pewnym ulepszeniem reguł NAT oraz poprzez integrację mechanizmu kolejkowania ALTQ. Równoważenie obciążenia i zarządzanie obciążeniem pasma są do siebie podobne, ale są czymś innym. Pierwsze z nich jest używane w celu równomiernego rozłożenia obciążenia przypadającego na intensywnie wykorzystywane komputery (np. jak serwery HTTP) na wiele maszyn, natomiast drugie jest używane w celu sterowania ruchem pakietów wyjściowych i w pewien sposób ograniczania ruchu pakietów wejściowych.

10.1. Równoważenie obciążenia

Celem równoważenia obciążenia jest mniej więcej równomierna dystrybucja zadań pomiędzy dwoma lub większą liczbą maszyn lub połączeń. Równoważenie obciążenia jest nowym dodatkiem do *pf(4)*, ale sama koncepcja nie jest nowa, a na pewno nie dla tych Czytelników, którzy używali dystrybucji obciążenia typu *round robin* oferowanej przez serwer DNS *named(8)*:

```
www.example.com.    60    IN    A    a.a.a.a
www.example.com.    60    IN    A    a.a.a.b
www.example.com.    60    IN    A    a.a.a.c
www.example.com.    60    IN    A    a.a.a.d
```

W chwili gdy *BIND* otrzyma zapytanie o *www.example.com*, zwróci następujące zbiory adresów:

```
a.a.a.a a.a.a.b a.a.a.c a.a.a.d
```

następnie:

```
a.a.a.b a.a.a.c a.a.a.d a.a.a.a
```

następnie:

```
a.a.a.c a.a.a.d a.a.a.a a.a.a.b
```

następnie:

```
a.a.a.d a.a.a.a a.a.a.b a.a.a.c
```

następnie:

```
a.a.a.a a.a.a.b a.a.a.c a.a.a.d
```

następnie:

```
a.a.a.b a.a.a.c a.a.a.d a.a.a.a
```

i tak dalej.

Program *named* równomiernie rozkłada obciążenie pomiędzy cztery oddzielne serwery HTTP udostępniające te same treści, odsyłając klienty pytające o *www.example.com* do każdego z tych serwerów po kolei. W wyniku tego poszczególne serwery powinny otrzymać tylko jedną czwartą zapytań kierowanych do *www.example.com*.

Równoważenie obciążenia dokonywane przez *pf(4)* (patrz rysunek 10.1) działa według podobnych zasad, jakie przyjęto dla metody *round-robin* stosowanej w DNS. Pomiędzy nimi istnieją cztery zasadnicze różnice:

- *Nie ma potrzeby dokonywania edycji rekordów DNS*. Równoważenie obciążenia dokonywane przez *pf(4)* jest przezroczyste.
- *Większa elastyczność i lepsza kontrola*. *pf(4)* jest w stanie stosować różne algorytmy równoważenia obciążenia w zależności od klasy adresowej, protokołu lub portu.
- *Prędkość*. Algorytm *round-robin* stosowany w DNS jest wolniejszy od metody użytej w *pf(4)*.
- *Oszczędzanie przestrzeni adresów IP*. Aby korzystając z metody *round-robin* w DNS uruchomić publicznie dostępny serwer sieciowy, należy przydzielić mu publiczny adres IP. Nie jest to konieczne w przypadku *pf(4)*. Można uruchomić dowolną liczbę serwerów sieciowych w DMZ (zobacz rozdział 2., „Projekty zapór sieciowych"), które zostaną udostępnione pod jednym, tym samym publicznym adresem IP.

Równoważenie obciążenia oferowane przez pf(4) może działać jako uzupełnienie równoważenia DNS. Oba rozwiązania mogą być wykorzystywane jednocześnie i działać na różnym poziomie redundancji. Równoważenie z pf(4) nie zadziała jeśli zapora nie będzie w ogóle dostępna ze świata zewnętrznego. W takim przypadku sprawdzi się jednak równoważenie z DNS (pewien procent żądań będzie jednak nadal kierowany do nieosiągalnych serwerów).

Rysunek 10.1. *Rozkład obciążenia zaimplementowany za pomocą pf(4)*

10.1.1. Implementacja równoważenia obciążenia

Równoważenie obciążenia może być zaimplementowane jedynie dla reguł typu `nat` lub `rdr` (zobacz rozdział 7. „Przeadresowanie pakietów"). Reguły typu `binat` nie mogą być do tego użyte, ponieważ przypisane są dokładnie dwóm adresom.

Wyboru metody najlepiej pasującej do potrzeb dokonuje się poprzez użycie następujących słów kluczowych:

- `round-robin` — implementuje algorytm o tej samej nazwie. W przypadku użycia więcej niż jednego adresu docelowego jest to jedyny dozwolony algorytm:

```
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> { 10.1.1.1/24 } round-robin
oraz:
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> { 10.1.1.1/24, 192.168.22.5/8, 10.34.2.76 } round-robin
a także:
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> { 10.1.1.45, 192.168.22.5, 10.34.2.76 } round-robin
```

- `random` — algorytm wybiera w sposób losowy adresy maszyn z podanej podsieci zamiast listy adresów (używanej w przypadku `round-robin`). Podsieć jest adresem sieciowym podanym w notacji adres/maska opisanej w rozdziale 5. „Plik /etc/pf.conf". Na przykład następująca reguła losowo przekierowuje pakiety na osiem adresów w segmencie 10.4.3.6/29:

```
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> 10.4.3.6/29 random
```

A jeśli nie posiada się ośmiu maszyn w swojej sieci? Można wtedy przypisać więcej niż jeden adres do poszczególnych interfejsów, zwracając uwagę, by nie przypisać tego samego adresu do dwóch interfejsów. Tę czynność wykonuje się za pomocą opcji `alias` w programie *ifconfig(8)*. Następnie należy skonfigurować serwery tak, aby nasłuchiwały także na dodatkowych adresach. Ten sposób łamie nieco losową naturę samej metody, dlatego też serwer posiadający dodatkowe adresy musi charakteryzować się najlepszą wydajnością.

- `source-hash` — przy stosowaniu `round-robin` lub `random` może się zdarzyć, że *pf(4)* za każdym razem kierować będzie połączenia z jednego komputera źródłowego do różnych komputerów docelowych, co nie zawsze jest pożądanym

zjawiskiem. Aby zapewnić stałe połączenie pomiędzy określoną maszyną źródłową a zawsze tym samym komputerem docelowym, należy użyć opcji `source-hash`, która za pierwszym razem wybiera adres docelowy w sposób losowy i wiąże go z adresem źródłowym, a wszystkie kolejne połączenia z tego samego adresu źródłowego kierować będzie do tego samego adresu docelowego:

```
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> 10.4.3.6/29 source-hash
```

Początkowe przypisanie jest losowe, chyba że poda się wartość mieszającą:

```
rdr on ne0 proto tcp from any to $ext_ad port 80 \
    -> 10.4.3.6/29 source-hash lancuchmieszajacymozebycdowolny
```

- `bitmask` — metoda ta sama w sobie nie jest równoważeniem obciążenia. Wiąże adresy sieciowe w sposób „jeden-do-jednego" w `nat`, lecz nie wykonuje dwukierunkowej translacji. Oba segmenty sieciowe muszą mieć równą wielkość:

```
# przekierowanie połączeń:
# od 192.168.1.1 do 10.4.3.1
# od 192.168.1.2 do 10.4.3.2
# od 192.168.1.3 do 10.4.3.3
# i tak dalej...
nat on ne0 proto tcp from 192.168.1/24 to any \
    -> 10.4.3/24 bitmask
```

Niestety adresowanie dokonywane przez `round-robin` oraz `random` zakłóca niektóre protokoły, jak np. SSL, co może być nie do przyjęcia. W takich wypadkach należy użyć translacji `source-hash` lub `bitmask`.

Reguły `rdr` dostosowane są do połączeń wejściowych lub przekierowania portów dla wyjściowych połączeń z wieloma maszynami proxy. Czy można posiadać dwa łącza z internetem i spowodować, aby były równomiernie obciążone przez maszyny z sieci wewnętrznej? Tak, jeśli użyje się opcji `source-hash` w regule typu `nat`. W tym przypadku adresy zewnętrznych interfejsów muszą należeć do tego samego segmentu, tj.:

```
nat on $ext_if from 10.3.3.1/24 to any \
    -> 192.168.23.34/31 source-hash
```

Innym rozwiązaniem mogłoby być zastosowanie opcji trasowania `reply-to` (patrz rozdział 8., „Filtrowanie pakietów"), powodującej wysyłanie odpowiedzi poprzez inny interfejs.

Gdy używa się opcji puli w regułach `nat`, opcja `static-port` pozwala wyłączyć zmiany przypisań portów dokonywane przez te reguły.

10.2. Zarządzanie obciążeniem pasma

Zarządzanie obciążeniem pasma wykonuje się w systemach OpenBSD za pomocą ALTQ, który jest częścią projektu KAME.

www.kame.net — KAME, siedziba ALTQ.

ALTQ jest w szczególny sposób efektywny, jeśli chce się, aby pewne pakiety miały większą wagę niż pozostałe i były przetwarzane wcześniej. Rozwiązaniem przyjętym w ALTQ jest zarządzanie pasmem w oparciu o zmodyfikowany mechanizm typu „pierwszy na wejściu, pierwszy na wyjściu" (FIFO) zaimplementowany domyślnie do przetwarzania pakietów w stosie TCP/IP systemu OpenBSD. Domyślny algorytm przetwarza pakiety w takim porządku, w jakim one przychodzą. ALTQ przydziela poszczególne pakiety do oddzielnych kolejek (list pakietów) o różnych priorytetach. Pakiety umieszczone w kolejkach o wyższym priorytecie przetwarzane są przed pakietami z kolejek o niższym priorytecie. Te ostatnie utrzymywane są w pamięci aż do chwili, gdy wszystkie pakiety z kolejek o wyższym priorytecie zostaną przetworzone.

Zarządzanie kolejkami opiera się na kilku algorytmach. OpenBSD 3.4 wspiera trzy typy algorytmów: *kolejkowanie priorytetowe* (ang. *Priority Queuing*) PRIQ, *kolejkowanie klasowe* (ang. *Class-Based Queuing*) CBQ oraz HFSC (ang. *Hierarchical Fair Service Curve*). Główną różnicą pomiędzy nimi jest sposób traktowania pakietów oraz szerokości pasma. PRIQ zarządza szerokością pasma przez przetwarzanie pakietów zgodnie z ich poziomami priorytetów. Im wyższy poziom, tym wcześniej pakiet zostanie przetworzony. CBQ pozwala na zorganizowanie kolejki w postaci złożonego drzewa różnorodnych priorytetów i szerokości pasma, podobnie jak HFSC, który daje administratorowi daleko większy stopień kontroli nad drzewami kolejek. W obu przypadkach definicja kolejki rozpoczyna się pojedynczą regułą algorytmu, po którym następuje lista reguł definiujących kolejki.

10.2.1. Anatomia reguły nadrzędnej

Wszystkie definicje kolejek rozpoczynają się pojedynczą regułą, która definiuje kolejkę *nadrzędną*:

- *Słowo kluczowe* `altq`. Oznacza rozpoczęcie definicji kolejki nadrzędnej. Do jednego interfejsu można przypisać tylko jedną kolejkę nadrzędną. Ta część jest wymagana.
- *Specyfikacja interfejsu*. Słowo kluczowe `on`, za nim występuje nazwa interfejsu, któremu przypisuje się kolejkę nadrzędną. Część wymagana.
- *Algorytm*. Należy podać: `priq` (PRIQ), `cbq` (CBQ) lub `hfsc` (HFSC). Wybiera typ algorytmu kolejek użytego dla kolejki nadrzędnej. Ta część jest obowiązkowa.
- *Maksymalna dostępna szerokość pasma*. Słowo kluczowe `bandwidth`, za którym następuje liczba zakończona jednym z następujących przyrostków: `b` (bitów na sekundę), `Kb` (kilobitów na sekundę), `Mb` (megabitów na sekundę) lub `Gb` (gigabitów na sekundę). Jest to wartość oznaczająca maksymalną szerokość pasma interfejsu, któremu przypisywana jest kolejka nadrzędna. Podanie niższych wartości nie przeszkadza (powiedzmy 10 Mb zamiast 100 Mb), ale przypisanie wyższej od rzeczywistej nie pomoże (nie można zmusić interfejsu, aby pracował szybciej, niż pozwala na to sprzęt). Ta część nie jest wymagana. Jeśli zostanie pominięta, *pfctl(8)* spróbuje automatycznie użyć prędkości oferowanej przez interfejs, jeśli zdoła ją określić, w przeciwnym przypadku zgłosi błąd.

- *Ograniczenie długości kolejki*. Słowo kluczowe `limit`, po którym następuje liczba całkowita. Wartość tego parametru oznacza dla ALTQ liczbę pakietów utrzymywanych w kolejce. Wartością domyślną jest 50. Można zmienić długość kolejki, jeśli połączenia są często zrywane (zobacz też rozdział 14. „Optymalizacja zbioru reguł", gdzie zamieszczono informacje o wartościach stanowych czasów oczekiwania). Ta część nie jest wymagana.
- *Regulator koszyka żetonów*. Słowo kluczowe `tbrsize`, po którym następuje liczba bajtów określająca szybkość, z jaką ALTQ powinien wysyłać pakiety z kolejki. Ta część nie jest obowiązkowa i jeśli jest pominięta, ALTQ automatycznie dopasuje się do optymalnego poziomu.
- *Lista kolejek*. Słowo kluczowe `queue`, po którym następuje lista kolejek podrzędnych umieszczona w nawiasach klamrowych. Nazwa każdej kolejki musi być unikalna, lecz nie trzeba umieszczać nazwy domyślnej kolejki w tym miejscu (którą tak czy inaczej należy zdefiniować). Nazwy kolejek mogą być dowolne, nie mogą być tylko zarezerwowanymi przez *pf(4)* słowami kluczowymi. Jest to część obowiązkowa.

Poniżej zamieszczono przykłady definicji kolejki nadrzędnej:

```
# definiuje kolejkę nadrzędną i daje jej pod zarządzanie
# 45Mb z całego pasma; definiuje cztery kolejki podrzędne: ssh,
# www, other (domyślna), ctrl (sterująca); zarządzana jest za
# pomocą PRIQ
altq on $ext_if priq bandwidth 45Mb queue {ssh, www, other, ctrl}

# definiuje kolejkę nadrzędną z pasmem 45 Mb i sześcioma
# kolejkami podrzędnymi: accounting, developers, managers,
# users, other (domyślna), ctrl (sterująca); zarządzana jest za
# pomocą CBQ
altq on $ext_if cbq bandwidth 45Mb \
    queue {accounting, developers, managers, users, other, ctrl}

# definiuje kolejkę nadrzędną z pasmem 45 Mb i sześcioma
# kolejkami podrzędnymi: accounting, developers, managers,
# users, other (domyślna), ctrl (sterująca); zarządzana jest za
# pomocą HFSC
altq on $ext_if hfsc bandwidth 45Mb \
    queue {accounting, developers, managers, users, other, ctrl}
```

10.2.2. Anatomia reguły kolejkowej

Po zdefiniowaniu kolejek nadrzędnych należy przejść do definicji kolejek podrzędnych przypisanych do każdej z kolejek nadrzędnych:

- *Słowo kluczowe* `queue`. Rozpoczyna definicję kolejki podrzędnej. Część wymagana.
- *Szerokość pasma kolejki*. Słowo kluczowe `bandwidth`, po którym następuje wartość maksymalnej szerokości pasma dostępnego dla danej kolejki podrzędnej. Wartość ta może być określona w bitach (`b`), kilobitach (`Kb`), megabitach (`Mb`), gigabitach (`Gb`) lub w procentach (`%`) szerokości pasma kolejki bezpośrednio nadrzędnej. Ta część jest wymagana, ale nie jest dozwolona dla PRIQ.

- *Priorytet kolejki*. Słowo kluczowe `priority`, za którym występuje liczba całkowita (0 – 15 dla kolejek PRIQ, 0 – 7 dla kolejek CBQ). Im wyższa wartość tego argumentu, tym wyższy priorytet kolejki (15 oznacza najwyższy priorytet w PRIQ, 7 dla CBQ, 0 jest najniższym priorytetem dla obu przypadków). Ta część jest obowiązkowa.
- *Opcje algorytmu*. Nazwa algorytmu, za którą następuje lista opcji ujęta w nawiasy okrągłe:
  - `borrow` — (tylko CBQ i HFSC) bieżąca kolejka może pożyczać pasmo ze swojej kolejki nadrzędnej w momencie, gdy pasmo nie jest przez kolejkę nadrzędną w pełni wykorzystane.
  - `default` — każda kolejka nadrzędna określa jedną kolejkę podrzędną, która domyślnie zarządza pakietami nienależącymi do innych kolejek podrzędnych.
  - `red` — pakiety szeregowane są za pomocą RED (ang. *Random Early Detection*). Pakiety będą odrzucane w sposób proporcjonalny do długości kolejki. Pakiety w dłuższych kolejkach odrzucane są wcześniej niż pakiety z kolejek krótszych. W praktyce oznacza to, że komunikatory internetowe, SSH lub telnet będą działały szybciej, podczas gdy długotrwałe transfery w rodzaju FTP staną się wolniejsze.
  - `ecn` — pakiety w takiej kolejce są szeregowane z użyciem ECN (ang. *Excplicit Congestion Notification*), co zostało opisane w [RFC 3168]. Można w skrócie powiedzieć, że ECN jest rozszerzeniem RED, umożliwiającym routerom powiadamianie klientów i serwerów o konieczności spowolnienia ruchu w sieci z powodu przeciążenia.
  - `rio` — pakiet szeregowane są z użyciem RED IN/OUT. Użycie tej opcji wymaga uaktywnienia RIO w jądrze systemowym i przebudowania go. W tym celu należy dodać następujący wiersz w pliku konfiguracyjnym jądra:

```
option ALTQ_RIO
```

Więcej informacji na temat przebudowywania jądra można znaleźć w *Upgrade-MiniFAQ*: *http://www.openbsd.org/faq/upgrade-minifaq.html*.

  Ta część nie jest obowiązkowa.
- *Lista kolejek podrzędnych*. Lista nazw kolejek podrzędnych danej kolejki umieszczona w nawiasach klamrowych. Każda nazwa musi być unikalna. Ta część nie jest wymagana dla CBQ i HFSC, a niedozwolona dla PRIQ (dla których można określić tylko jeden poziom kolejek podrzędnych).

10.2.3. Przypisywanie kolejek do reguł filtrowania pakietów

Po zdefiniowaniu kolejki nadrzędnej i jej kolejek podrzędnych należy przypisać do nich pakiety dopasowywane przez różne reguły typu `pass`. Uzyskuje się to przez użycie słowa kluczowego `queue` umieszczonego na samym końcu reguły, na przykład:

```
pass out quick on $ext_if from any to any queue users
```

Dozwolone jest umieszczenie nazw dwóch kolejek:

```
pass out quick on $ext_if from any to any queue {users, admins}
```

Przy podaniu dwóch kolejek druga z nich zostanie użyta, jeżeli:

- Pole `TOS` dopasowanych pakietów ustawiono na `lowdelay`.
- Dopasowane zostały pakiety `TCP ACK`, które nie posiadają żadnych danych.

Pakiety nieprzypisane do innych kolejek zostaną automatycznie przypisane kolejce domyślnej.

10.2.4. Kolejkowanie priorytetowe (PRIQ)

Algorytm PRIQ używa prostego modelu płaskiego pasma, które podzielone jest na mniejsze składowe posiadające różne priorytety. Jest to efektywny sposób implementacji prostej polityki kolejkowania w rodzaju „połączenia SSH są ważniejsze niż HTTP i NNTP" lub „połączenia do działu badań są ważniejsze od połączeń z biblioteką, lecz oba są mniej ważne niż połączenia z administratorami sieci".

Zobaczmy, w jaki sposób implementuje się następującą politykę:

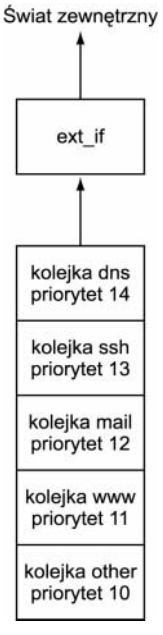
- Zapytania DNS mają najwyższy priorytet.
- Połączenia typu SSH oraz TELNET mają niższe priorytety niż DNS.
- Połączenia do serwerów pocztowych (SMTP, POP2, POP3, IMAP, IMAP3, POP3S) mają niższy priorytet niż połączenia SSH i TELNET.
- Połączenia z serwerami WWW (HTTP, HTTPS) mają niższy priorytet niż połączenia z serwerami pocztowymi.
- Wszystkie inne typy połączeń mają najniższy, domyślny priorytet.

Przykładowy zbiór reguł oparty na algorytmie PRIQ umożliwiający nadanie różnych priorytetów różnym typom usług (patrz rysunek 10.2) pokazany został poniżej:

```
# MAKRODEFINICJE
# zewnętrzne interfejsy
ext_if = "ne1"

# DEFINICJA KOLEJKI NADRZĘDNEJ
# definiuje kolejkę nadrzędną PRIQ: pasmo 45 Mb i pięć kolejek
# podrzędnych: dns, ssh, www, mail, other (domyślna).
# Kolejność kolejek podrzędnych jest nieistotna
altq on $ext_if priq bandwidth 45Mb \
    queue {dns, ssh, www, mail, other}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# Zapytania DNS posiadają wysoki priorytet, ponieważ muszą
# być wykonane tak szybko, jak to możliwe
queue dns priority 14 priq(red)
# Połączenia SSH mają jeden z wysokich priorytetów, ponieważ
# często używane są w celach administracyjnych
```



Rysunek 10.2.
Układ kolejkowania typu PRIQ dla ALTQ ustalony dla różnych zewnętrznych usług

```
queue ssh priority 13 priq(red)
# Połączeniom pocztowym nadaje się niższy priorytet niż SSH,
# lecz wyższy niż HTTP/HTTPS, ponieważ chcemy wysyłać/odbierać
# nasze listy tak szybko, jak to możliwe
queue mail priority 12 priq(red)
# Połączenia HTTP/HTTPS mają niższy priorytet, ponieważ nie są
# tak bardzo czułe na opóźnienia jak inne
queue www priority 11 priq(red)
# Inne połączenia przypisane są domyślnej kolejce
queue other priority 10 priq(default)

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM
# Pakiety wysyłane na port 53 (DNS) zostaną przypisane kolejce
# dns (należy zauważyć użycie keep state zamiast synproxy state
# lub modulate state, ponieważ pakiety UDP mogą być poddane
# filtrowania tylko z użyciem keep state).
pass out quick on $ext_if inet proto udp \
    from any to any port 53 keep state queue dns
pass out quick on $ext_if inet proto tcp \
    from any to any port 53 synproxy state queue dns
# pakiety wysłane na port 22 (SSH), 23 (TELNET) zostaną przypisane
# do kolejki ssh
pass out quick on $ext_if inet proto tcp \
    from any to any port (22, 23) synproxy state queue ssh
# pakiety wysłane na port 25 (SMTP), 109 (POP2), 110 (POP3),
# 143 (IMAP), 220 (IMAP3), 995 (POP3S) zostaną przypisane do
# kolejki mail
pass out quick on $ext_if inet proto tcp \
    from any to any port {25, 109, 110, 143, 220, 995} \
    synproxy state queue mail
# pakiety wysłane na port 80 (HTTP), 443 (HTTPS) zostaną
# przypisane kolejce www
pass out quick on $ext_if inet proto tcp \
    from any to any port {80, 443) synproxy state queue www
```

Inny przypadek: nadawanie różnych priorytetów połączeniom inicjowanym przez różne zewnętrzne maszyny.

- Pakiety wysłane z maszyn używanych przez administratorów mają najwyższy priorytet.
- Pakiety wysłane z komputerów z działu finansów mają niższe priorytety niż pakiety wysyłane z maszyn administratorów.
- Pakiety wysyłane z komputerów programistów mają niższy priorytet niż pakiety z komputerów działu finansów.
- Pakiety z komputerów zwykłych użytkowników mają niższy priorytet niż pakiety wysyłane z maszyn programistów.
- Wszystkie inne połączenia maja najniższy, domyślny priorytet.

Przykładowy zbiór reguł opartych na algorytmie PRIQ, nadający różne priorytety połączeniom wyjściowym z różnych komputerów pokazany został poniżej:

```
# MAKRODEFINICJE
# zewnętrzny interfejs
ext_if = "ne1"
# maszyny administratorów
admins_ad = "{a.a.a.a, a.a.a.b}"
# maszyny księgowych
accounts_ad = "{a.a.a.c, a.a.a.d, a.a.a.e}"
# maszyny programistów
coders_ad = "{a.a.a.f, a.a.a.g, a.a.a.h}"
# maszyny użytkowników
users_ad = "{a.a.a.i, a.a.a.j, a.a.a.k}"

# DEFINICJA KOLEJKI NADRZĘDNEJ
# definiuje kolejkę nadrzędną PRIQ: pasmo 45 Mb i pięć kolejek
# podrzędnych: admins, accounts, coders, users, other (domyślna)
altq on $ext_if priq bandwidth 45Mb \
    queue {admins, accounts, coders, users, other}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# administratorzy otrzymują najwyższy priorytet
queue admins priority 14 priq(red)
# dział finansowy
queue accounts priority 13 priq(red)
# programiści
queue coders priority 12 priq(red)
# zwykli użytkownicy
queue users priority 11 priq(red)
# inne
queue other priority 10 priq(default)

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM
# administratorzy
pass out quick on $ext_if inet proto tcp \
    from $admins_ad to any synproxy state queue admins
pass out quick on $ext_if inet proto udp \
    from $admins_ad to any keep state queue admins
# księgowi
pass out quick on $ext_if inet proto tcp \
    from $accounts_ad to any synproxy state queue accounts
```

```
pass out quick on $ext_if inet proto udp \
    from $accounts_ad to any keep state queue accounts
# programiści
pass out quick on $ext_if inet proto tcp \
    from $coders_ad to any synproxy state queue coders
pass out quick on $ext_if inet proto udp \
    from $coders_ad to any keep state queue coders
# użytkownicy
pass out quick on $ext_if inet proto tcp \
    from $users_ad to any synproxy state queue users
pass out quick on $ext_if inet proto udp \
    from $users_ad to any keep state queue users
```

Zaprezentowany powyżej zbiór reguł zakłada, że wewnętrzne maszyny mają prawidłowe adresy publiczne. Co zrobić z NAT, który ukrywa wszystkie maszyny za pojedynczym interfejsem i wysyła wszystkie pakiety na zewnątrz z adresem źródłowym równym adresowi zewnętrznego interfejsu zapory? W dalszym ciągu można rozróżnić komputery, jeśli zdefiniuje się, które porty mogą być użyte przez poszczególne komputery:

```
# MAKRODEFINICJE
# zewnętrzny interfejs
ext_if = "ne1"
# maszyny administratorów
admins_ad = "{a.a.a.a, a.a.a.b}"
# maszyny księgowych
accounts_ad = "{a.a.a.c, a.a.a.d, a.a.a.e}"
# maszyny programistów
coders_ad = "{a.a.a.f, a.a.a.g, a.a.a.h}"
# maszyny użytkowników
users_ad = "{a.a.a.i, a.a.a.j, a.a.a.k}"

# DEFINICJA KOLEJKI NADRZĘDNEJ
# definiuje kolejkę nadrzędną PRIQ: pasmo 45 Mb i pięć kolejek
# podrzędnych: admins, accounts, coders, users, other (domyślna)
altq on $ext_if priq bandwidth 45Mb \
    queue {admins, accounts, coders, users, other}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# administratorzy otrzymują najwyższy priorytet
queue admins priority 14 priq(red)
# dział finansów
queue accounts priority 13 priq(red)
# programiści
queue coders priority 12 priq(red)
# zwykli użytkownicy
queue users priority 11 priq(red)
# inne
queue other priority 10 priq(default)

# REGUŁY NAT
# administratorzy
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.a to any -> ($ext_if) port 1024:6888
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.b to any -> ($ext_if) port 6889:12753
# księgowi
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.c to any -> ($ext_if) port 12754:18618
```

```
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.d to any -> ($ext_if) port 18619:24483
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.e to any -> ($ext_if) port 24484:30348
# programiści
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.f to any -> ($ext_if) port 30349:36213
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.g to any -> ($ext_if) port 36214:42078
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.h to any -> ($ext_if) port 42079:47943
# użytkownicy
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.i to any -> ($ext_if) port 47944:53808
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.j to any -> ($ext_if) port 53809:59673
nat on $ext_if inet proto {tcp, udp} \
    from a.a.a.k to any -> ($ext_if) port 59674:65535

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM
# administratorzy
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 1023><6889 to any queue admins
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 6888><12754 to any queue admins
# księgowi
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 12753><18619 to any queue accounts
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 18618><24484 to any queue accounts
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 24483><30349 to any queue accounts
# programiści
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 30348><36214 to any queue coders
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 36213><42079 to any queue coders
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 42078><47944 to any queue coders
# użytkownicy
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 47943><53809 to any queue users
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 53808><59674 to any queue users
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 59673><65536 to any queue users
```

Jak widać, reguły PRIQ nie pozwalają definiować wielkości pasma przypisanej każdej kolejce podrzędnej, tylko kontrolują jej priorytet. Jeśli chce się mieć lepszą kontrolę nad szerokością pasma dla poszczególnych kolejek, należy użyć algorytmu CBQ lub HFSC.

10.2.5. Kolejkowanie oparte na klasach (CBQ)

Algorytm CBQ pozwala na lepszą kontrolę szerokości pasma. Można decydować nie tylko o priorytecie każdej kolejki, ale również o szerokości przypisanego jej pasma. Kolejki można zorganizować na kilku poziomach. Możliwe jest pożyczanie pasma od

kolejek nadrzędnych. Ta cecha umożliwia implementację takich polityk jak: „finanse muszą mieć co najmniej pasmo 1 Mb, programiści nie mogą używać większego pasma niż 2 Mb, a menedżerowie nie mogą używać więcej niż 1 Mb, z tym że szef musi mieć co najmniej 200 Kb".

Najlepszym sposobem nauczenia się CBQ jest napisanie prostej konfiguracji. Załóżmy, że chcemy podzielić pasmo pomiędzy dwa segmenty sieci połączonej do zapory, która dokonuje translacji typu NAT (patrz rysunek 10.3):

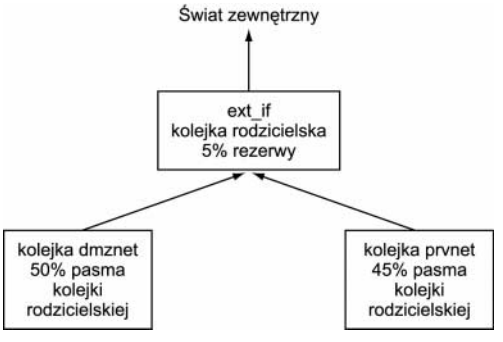
```
# MAKRODEFINICJE
# zewnętrzny interfejs
ext_if = "ne1"
# interfejs DMZ
dmz_if = "ne2"
# interfejs prywatny
prv_if = "ne3"

# DEFINICJA KOLEJKI NADRZĘDNEJ
# definiuje kolejkę nadrzędną CBQ: pasmo 45 Mb z trzema kolejkami
# podrzędnymi: dmznet (komputery w DMZ),
# prvnet (komputery w prywatnym segmencie),
# other (domyślna, połączenia z samej zapory)
altq on $ext_if cbq bandwidth 45Mb \
    queue {dmznet, prvnet, other}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# nadaje wyjściu z DMZ 50% szerokości pasma kolejki
# nadrzędnej
queue dmznet bandwidth 50% priority 6 cbq(red)
# nadaje wyjściu z prywatnej sieci pasmo 45% szerokości pasma kolejki
# nadrzędnej
queue prvnet bandwidth 45% priority 6 cbq(red)
# przypisuje pasmo wyjściu z zapory
queue others bandwidth 5% priority 5 cbq(default)

# TRANSLACJA PAKIETÓW
# dodaje reguły NAT ze zdefiniowanymi zakresami portów, tak abyśmy
# wiedzieli, skąd przychodzą wejściowe pakiety (NAT zmieni ich adresy
# źródłowe i numerami portów trzeba posłużyć się do rozpoznania, kto jest kim).
# Wyłącz NAT dla połączeń pomiędzy prywatną siecią a segmentem DMZ
no nat on $ext_if inet proto {tcp, udp} \
    from $dmz_if:network to $prv_if:network
no nat on $ext_if inet proto {tcp, udp} \
    from $prv_if:network to $dmz_if:network
# dokonuj translacji NAT pomiędzy segmentem DMZ a światem zewnętrznym
nat on $ext_if inet proto {tcp, udp} \
    from $dmz_if:network to any -> $ext_if port 1024:32255
# dokonaj translacji NAT pomiędzy segmentem prywatnym a światem
# zewnętrznym
nat on $ext_if inet proto {tcp, udp} \
    from $prv_if:network to any -> $ext_if port 32256:65535

# REGUŁY FILTROWANIA PAKIETÓW PRZYPISANE KOLEJKOM
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 1023><32256 to any queue dmznet
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 32255><65536 to any queue prvnet
```



Rysunek 10.3. *Podział pasma pomiędzy dwa wewnętrzne segmenty sieci za pomocą CBQ*

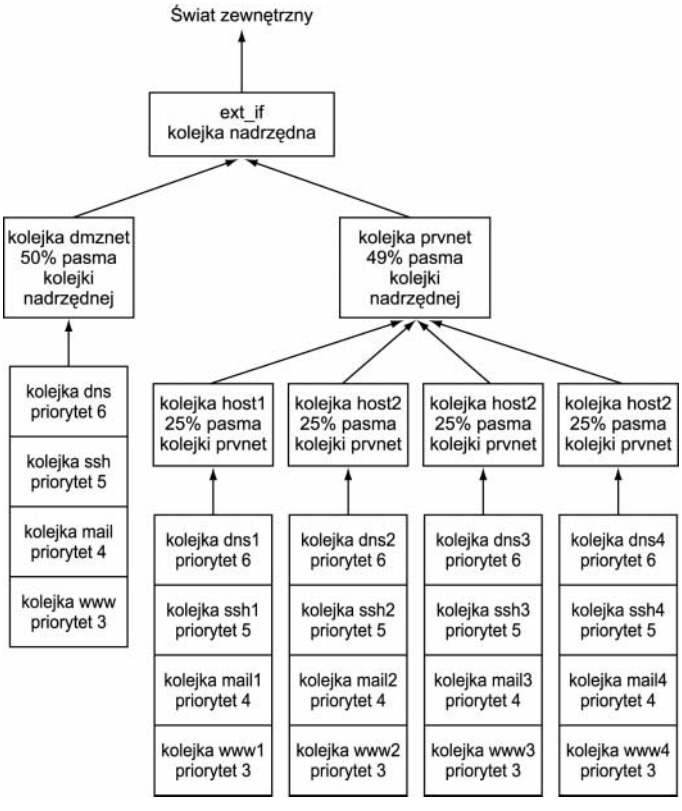
Przedstawiony powyżej zbiór reguł pokazuje podstawową implementację kolejkowania pakietów, która nakłada ograniczenie na szerokość pasma pakietów wyjściowych wysłanych z segmentu DMZ i prywatnego. Pakiety te nie są poddawane żadnemu zaawansowanemu kolejkowaniu. Należy zauważyć, że kolejka nadrzędna używa 50% pasma swojego interfejsu. Zrobiono tak, aby pozostawić trochę miejsca dla połączeń wejściowych, inaczej użytkownicy komputerów podłączonych do segmentu prywatnego mogliby zużyć całe dostępne pasmo.

Zobaczmy, w jaki sposób zaprezentowany zbiór reguł może zostać rozszerzony do postaci bardziej złożonego drzewa kolejek, które implementują następującą politykę kolejkowania pakietów (patrz rysunek 10.4):

- Kolejka nadrzędna używa 50% całej szerokości pasma.
- Segment DMZ używa 50% pasma kolejki nadrzędnej. Pakiety kolejkowane są z użyciem algorytmu CBQ bez wyszczególnienia pasma, ale z podanym priorytetem (tak jak w przypadku PRIQ, ale nie można użyć więcej niż jednego algorytmu, a zakres priorytetów jest mniejszy: od 0 do 7).
- Prywatny segment używa 49% pasma swej kolejki nadrzędnej.
- Każdy z *n* prywatnych komputerów dostaje 1/*n*-tą szerokości pasma swej kolejki nadrzędnej (w tym przypadku 25% dla każdej maszyny z tego segmentu).
- Kolejkowanie pakietów dla każdej prywatnej maszyny jest wykonywane w sposób podobny do polityki kolejkowania zastosowanej dla maszyn z DMZ.

Poniżej zamieszczono przykład zbioru reguł dla takiego podejścia:

```
# MAKRODEFINICJE
# zewnętrzny interfejs
ext_if = "ne1"
# interfejs DMZ
dmz_if = "ne2"
# interfejs prywatny
prv_if = "ne3"
host1_ad = "192.168.5.10/32"
host2_ad = "192.168.5.11/32"
host3_ad = "192.168.5.12/32"
host4_ad = "192.168.5.13/32"
# DEFINICJA KOLEJKI NADRZĘDNEJ
# definiuje kolejkę nadrzędną CBQ: pasmo 45 Mb z trzema kolejkami
# podrzędnymi: dmznet (komputery w DMZ),
# prvnet (komputery w prywatnym segmencie),
# other (domyślna)
```



Rysunek 10.4. *Bardziej złożony układ kolejek*

```
altq on $ext_if cbq bandwidth 45Mb \
    queue {dmznet, prvnet, others}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# nadaje wyjściu z DMZ 50% szerokości pasma kolejki
# nadrzędnej
queue dmznet bandwidth 50% priority 6 cbq(red) \
    queue{dns, ssh, www, mail}
# nadaje wyjściu z prywatnej sieci pasmo 49% szerokości pasma kolejki
# nadrzędnej
queue prvnet bandwidth 49% priority 5 cbq(red) \
    queue{host1, host2, host3, host4}
# przypisuje pasmo do wyjścia z zapory
queue others bandwidth 1% priority 4 cbq(default)

# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla dmznet)
queue dns priority 6 cbq(red, borrow)
queue ssh priority 5 cbq(red, borrow)
queue mail priority 4 cbq(red, borrow)
queue www priority 3 cbq(red, borrow)

# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla prvnet)
queue host1 bandwidth 25% cbq(red) {dns1, ssh1, mail1, www1}
queue host2 bandwidth 25% cbq(red) {dns2, ssh2, mail2, www2}
queue host3 bandwidth 25% cbq(red) {dns3, ssh3, mail3, www3}
queue host4 bandwidth 25% cbq(red) {dns4, ssh4, mail4, www4}
```

```
# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla host1)
queue dns1 priority 6 cbq(red, borrow)
queue ssh1 priority 5 cbq(red, borrow)
queue mail1 priority 4 cbq(red, borrow)
queue www1 priority 3 cbq(red, borrow)

# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla host2)
queue dns2 priority 6 cbq(red, borrow)
queue ssh2 priority 5 cbq(red, borrow)
queue mail2 priority 4 cbq(red, borrow)
queue www2 priority 3 cbq(red, borrow)

# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla host3)
queue dns3 priority 6 cbq(red, borrow)
queue ssh3 priority 5 cbq(red, borrow)
queue mail3 priority 4 cbq(red, borrow)
queue www3 priority 3 cbq(red, borrow)

# DEFINICJE KOLEJEK PODRZĘDNYCH: (dla host4)
queue dns4 priority 6 cbq(red, borrow)
queue ssh4 priority 5 cbq(red, borrow)
queue mail4 priority 4 cbq(red, borrow)
queue www4 priority 3 cbq(red, borrow)

# REGUŁY NAT
no nat on $ext_if from $dmz_if:network to $prv_if:network
no nat on $ext_if from $prv_if:network to $dmz_if:network
nat on $ext_if from $dmz_if:network to any \
    -> ($ext_if) port 1024:32255
nat on $ext_if from $host1_ad to any \
    -> ($ext_if) port 32256:40574
nat on $ext_if from $host2_ad to any \
    -> ($ext_if) port 40575:48893
nat on $ext_if from $host3_ad to any \
    -> ($ext_if) port 48894:57212
nat on $ext_if from $host4_ad to any \
    -> ($ext_if) port 57213:65535

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM: (dla dmznet)
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 1023 >< 32256 to any port 53 queue dns
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 1023 >< 32256 to any port {22, 23} \
    queue ssh
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 1023 >< 32256 to any \
    port {25, 109, 110, 143, 220, 995} queue mail
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 1023 >< 32256 to any port {80, 443} \
    queue www

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM: (dla host1)
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 32255 >< 40575 to any port 53 queue dns1
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 32255 >< 40575 to any port {22, 23} \
    queue ssh1
```

```
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 32255 >< 40575 to any \
    port {25, 109, 110, 143, 220, 995} queue mail1
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 32255 >< 40575 to any port {80, 443} \
    queue www1

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM: (dla host2)
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 40574 >< 48894 to any port 53 queue dns2
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 40574 >< 48894 to any port {22, 23} \
    queue ssh2
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 40574 >< 48894 to any \
    port {25, 109, 110, 143, 220, 995} queue mail2
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 40574 >< 48894 to any port {80, 443} \
    queue www2

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM: (dla host3)
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 48893 >< 57213 to any port 53 queue dns3
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 48893 >< 57213 to any port {22, 23} \
    queue ssh3
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 48893 >< 57213 to any \
    port {25, 109, 110, 143, 220, 995} queue mail3
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 48893 >< 57213 to any port {80, 443} \
    queue www3

# REGUŁY FILTROWANIA PRZYPISANE KOLEJKOM: (dla host4)
pass out quick on $ext_if inet proto {tcp, udp} \
    from ($ext_if) port 57212 >< 65536 to any port 53 queue dns4
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 57212 >< 65536 to any port {22, 23} \
    queue ssh4
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 57212 >< 65536 to any \
    port {25, 109, 110, 143, 220, 995} queue mail4
pass out quick on $ext_if inet proto tcp \
    from ($ext_if) port 57212 >< 65536 to any port {80, 443} \
    queue www4
```

10.2.6. Hierarchical Fair Service Curve (HFSC)

Algorytm HFSC oferuje możliwości podobne do CBQ, z tym że wzbogaca zestaw narzędzi administratora o możliwość definiowania dwóch typów algorytmów szeregowania pakietów: *czasu rzeczywistego* (ang. *real-time*) oraz *dzielenia łącza* (ang. *link-share*). Jeśli nie zostanie użyty ten pierwszy lub żadne pakiety nie mogą zostać poddane pod działanie tego algorytmu, HFSC użyje algorytmu typu *link-share*. Jeżeli dostępne jest większe pasmo, kolejki go użyją, chyba że nałoży się odgórne ograniczenie. Zachowanie HFSC kontrolowane jest przez następujące parametry:

- `realtime` — steruje minimalną szerokością pasma wymaganą dla kolejki. Gdy dostępne jest większe pasmo, zostanie ono użyte, chyba że wystąpi parametr `upperlimit`. Pakiety szeregowane są najpierw za pomocą algorytmu typu *real-time*. Gdy nie ma pakietów, które mogą być w ten sposób zaszeregowane, lub gdy nie używa się szeregowania czasu rzeczywistego, HFSC użyje szeregowania typu *link-share*. Ten parametr jest opcjonalny.
- `linkshare` — ustawia minimalny „udział" pasma kolejki nadrzędnej dla bieżącej kolejki. Jeżeli jest dostępne większe pasmo, zostanie ono użyte, chyba że wyspecyfikowano parametr `upperlimit`. Jeżeli używa się także parametru `realtime`, ma on pierwszeństwo przed `linkshare`. Parametr ten musi być użyty w definicji kolejki typu HFSC albo trzeba użyć słowa kluczowego `bandwidth` przed polem `hfsc()`, w innym przypadku *pfctl(8)* zgłosi błąd. Najprostsze ustawienia HFSC zostaną pokazane w dalszej części tego podrozdziału.
- `upperlimit` — ustawia maksymalną dozwoloną szerokość pasma dla danej kolejki. HFSC zastosuje to ograniczenie automatycznie dla każdego rodzaju algorytmu, jaki został użyty. Jeśli ten parametr jest użyty w definicji kolejki, musi być większy lub równy ograniczeniu zdefiniowanemu dla `realtime` oraz `linkshare`. Ten parametr nie jest wymagany.

Po każdym z opisanych parametrów może wystąpić pojedyncza wartość liczbowa określająca wielkość pasma, którego kolejka używa, lub trójka parametrów opisująca *krzywą usługi*, na którą składa się: (a) początkowy poziom pasma (*m1*), (b) opóźnienie (*d*, mierzone w milisekundach) oraz (c) poziom, do którego pasmo zostanie dopasowane po upływie czasu *d* (*m2*). Parametry te definiują tak zwaną *krzywą usługi*, która może być krzywą wypukłą lub wklęsłą, z wyjątkiem parametrów `realtime`, które muszą definiować krzywą wypukłą. Krzywa wypukła jest tworzona wtedy, gdy parametr *m1* jest większy niż *m2*. Możliwa jest także definicja płaskiej krzywej, gdy *m1* jest równe *m2*. Taki efekt uzyskuje się, pomijając określenia *m1* i *m2*.

HFSC nie robi dużego użytku z priorytetów określonych dla kolejek, mimo że można je określić za pomocą słowa kluczowego `priority` (zobacz podrozdziały opisujące PRIQ oraz CBQ), tak jak dla CBQ. Wpływ tego parametru na końcowy rezultat szeregowania nie jest duży i może zostać pominięty w celu uproszczenia zbioru reguł.

Inne słowo kluczowe, które w dużej mierze jest zbyteczne w kolejkach typu HFSC, to `bandwidth`. Istnieje jeden przypadek, kiedy jest ono pomocne, mianowicie prosta kolejka typu HFSC z jednym poziomem kolejek, w której wartości `bandwidth` są użyte jako pojedynczy parametr ustawień `linkshare`:

```
altq on $ext_if hfsc bandwidth 45Mb \
    queue {dns, ssh, www, mail, other}

queue dns bandwidth 20%
queue ssh bandwidth 20%
queue mail bandwidth 20%
queue www bandwidth 20%
queue other hfsc(default)
```

co odpowiada:

```
altq on $ext_if hfsc bandwidth 45Mb \
    queue {dns, ssh, www, mail, other}
queue dns hfsc(linkshare 20%)
queue ssh hfsc(linkshare 20%)
queue mail hfsc(linkshare 20%)
queue www hfsc(linkshare 20%)
queue other hfsc(default)
```

To był prosty przykład. Zobaczmy, co się stanie, jeśli utworzymy coś bardziej złożonego:

```
# DEFINICJA KOLEJKI NADRZĘDNEJ
altq on $ext_if hfsc bandwidth 45Mb \
    queue {dmznet, prvnet, others}

# DEFINICJE KOLEJEK PODRZĘDNYCH
# jeżeli zaległości trwają krócej niż 10 sekund, kolejka dostaje 50% lub więcej
# całego dostępnego pasma; dla dłuższych okresów ograniczenie zmniejsza się
# do 65% całego dostępnego pasma
queue dmznet hfsc(linkshare (50% 10000 65%))
# jeżeli zaległości trwają krócej niż 5 sekund, kolejka dostaje 40% lub więcej
# całego dostępnego pasma, po ich upływie ograniczenie wzrasta
# do 25% całego dostępnego pasma
queue prvnet hfsc(linkshare (40% 5000 25%))
queue others hfsc(default)
```

Pierwszą rzeczą wartą zapamiętania jest to, że suma wartości parametrów tego samego typu (np. początkowy poziom pasma ograniczenia `linkshare`) nie może przekroczyć ogólnej wartości szerokości pasma dostępnej dla bieżącego poziomu kolejki. Kolejka domyślna nie bierze udziału w tej sumie. Ograniczenie to może być wyrażone za pomocą tych samych jednostek, których używa się przy słowie kluczowym `bandwidth` (`%`, `b`, `Kb`, `Mb`, `Gb`).

W poprzednim przykładzie 45Mb pasma przypisanych kolejce nadrzędnej reprezentuje 100% dostępnego pasma. Kolejka nadrzędna podzielona została na trzy kolejki podrzędne: `dmznet` (która zabiera 50% całości), `prvnet` (która korzysta przynajmniej z 40% pasma) oraz `others` (która korzysta z tego, co pozostało). Jeżeli zaległości w kolejce `dmznet` będą trwały dłużej niż 10 sekund, algorytm HFSC rozpocznie zwiększanie ilości dostępnego pasma aż do 65% dostępnej szerokości. Jeżeli dostępne jest więcej, zostanie użyte w całości, lecz 65% jest gwarantowane (podobnie jak początkowe 50%). Gdy zaległości znikną, algorytm powoli uwolni dodatkowe pasmo i powróci do 50%. Jaki jest powód wzrostu szerokości pasma dla tej kolejki? Uniknięcie zaległości i zaspokojenie zwiększonych potrzeb co do szerokości pasma. Załóżmy, że pewien serwer pocztowy lub FTP, podłączony do DMZ, charakteryzuje się zwiększonym obciążeniem pomiędzy 10:00 a 16:00. Reguła z określonymi dwoma pasmami da sobie radę, bez ograniczania pasma na resztę dnia. Przyglądając się bliżej poprzednim regułom, można zauważyć, że szerokość pasma przydzielonego kolejce `prvnet` będzie spadać, jeżeli zaległości na łączu będą utrzymywały się przez pięć sekund. Dlaczego? Aby zrobić miejsce dla zwiększających się wymagań co do przepustowości dla kolejki `dmznet` i zapobiec wykorzystaniu całego pasma przez użytkowników `prvnet`.

Jak wspomniano wcześniej, użycie parametru `linkshare` wraz z `realtime` w tej samej kolejce jest prawidłowe — HFSC automatycznie wybierze ten parametr, który działa sprawniej. Można użyć tych samych parametrów dla obu typu algorytmów, lecz niektórzy administratorzy nieznacznie je różnicują. Należy używać tego, który najlepiej odpowiada danemu przypadkowi, trzeba jednak pamiętać, że krzywa dla `realtime` musi być wypukła. Innymi słowy, użycie tego parametru powoduje, że można zdefiniować tylko ilość spadku dostępnego pasma w przedziale czasu. Jeśli chce się zdefiniować wzrost, należy użyć parametru `linkshare`.

```
# DEFINICJA KOLEJKI NADRZĘDNEJ
altq on $ext_if hfsc bandwidth 45Mb \
     queue {dmznet, prvnet, others}
# DEFINICJE KOLEJEK PODRZĘDNYCH
# jeżeli zaległości trwają krócej niż 10 sekund, kolejka dostaje 50% lub więcej
# całego dostępnego pasma, po ich upływie ograniczenie zmniejsza się
# do 65% całego dostępnego pasma
queue dmznet hfsc(linkshare (50% 10000 65%))
# jeżeli zaległości trwają krócej niż 5 sekund, kolejka dostaje 40% lub więcej
# całego dostępnego pasma, po ich upływie ograniczenie wzrasta
# do 25% całego dostępnego pasma
queue prvnet \
    hfsc(realtime (40% 5000 25%) linkshare (40% 5000 25%))
queue others hfsc(default)
```

Jak wspomniano wyżej, można zdefiniować wklęsłą krzywą usługi dla parametru `linkshare`, aby zmniejszyć wielkość dostępnego pasma dla pewnych użytkowników i uniknąć wykorzystania przez nich całego łącza. Lepszym rozwiązaniem jest umieszczenie twardych ograniczeń rozmiaru pasma używanego przez każdą z kolejek poprzez zastosowanie parametru `upperlimit`:

```
# DEFINICJA KOLEJKI NADRZĘDNEJ
altq on $ext_if hfsc bandwidth 45Mb \
     queue {dmznet, prvnet, others}
# DEFINICJE KOLEJEK PODRZĘDNYCH
# jeżeli zaległości trwają krócej niż 10 sekund, kolejka dostaje przynajmniej 50% (maksymalnie 60%)
# całego dostępnego pasma, po ich upływie ograniczenie zmniejsza się
# do 65% (maksymalnie 75%) całego dostępnego pasma
queue dmznet hfsc(linkshare (50% 10000 65%)) \
    upperlimit (60% 10000 75%))
# jeżeli zaległości trwają krócej niż 5 sekund, kolejka dostaje przynajmniej 35% (maksymalnie 40%)
# całego dostępnego pasma, po ich upływie ograniczenie wzrasta
# do 20% (maksymalnie 25%) całego dostępnego pasma
queue prvnet hfsc(realtime (35% 5000 25%) \
    linkshare (35% 5000 20%)) upperlimit (40% 5000 25%))
queue others hfsc(default)
```

Kolejki podrzędne można definiować podobnie jak kolejki CBQ, istnieje jednak pewna znacząca różnica pomiędzy nimi, dotycząca sposobu określania procentu pasma dostępnego dla kolejki nadrzędnej, które może być wykorzystane przez podrzędną. CBQ stosuje metodę „proporcjonalną”, podczas gdy HFSC używa metody „substraktywnej”. Aby przekonać się, w jaki sposób działa to w praktyce, należy porównać podane poniżej reguły, które dzielą pasmo w ten sam sposób, lecz z innym zapisem podziału:

```
# CBQ
altq on $ext_if cbq bandwith 20Mb \
    queue {dmznet, prvnet, others}
```

```
# prvnet otrzymuje 8 Mb
queue prvnet bandwidth 40% queue {host1, host2}
# host1 otrzymuje 4 Mb
queue host1 bandwidth 50%
# host2 otrzymuje 4 Mb
queue host2 bandwidht 50%

# HFSC
altq on $ext_if hfsc bandwidth 20Mb \
    queue {dmznet, prvnet, others}

# prvnet otrzymuje 8 Mb
queue prvnet hfsc(linkshare 40%) queue {host1, host2}
# host1 otrzymuje 4 Mb
queue host1 hfsc(linkshare 20%)
# host2 otrzymuje 4 Mb
queue host2 hfsc(linkshare 20%)
```

10.2.7. Kolejkowanie pakietów wejściowych

ALTQ działa bardzo dobrze dla pakietów wyjściowych. Pozostaje pytanie: co z pakietami wejściowymi? Chociaż możliwe jest utworzenie kolejek dla połączeń wejściowych, okaże się to pomocne tylko wtedy, gdy będą one dopasowane do ograniczeń nałożonych na pasmo, zanim pakiety dotrą do naszej sieci. Można to uzyskać korzystając z pomocy lokalnego ISP, który musi tak skonfigurować swoje routery, aby ograniczyć ruch pakietów wejściowych do ustalonego poziomu. W innym przypadku ALTQ nie sprawdzi się, ponieważ po pojawieniu się pakietów na zewnętrznym interfejsie jest już za późno, aby je zatrzymać — już wykorzystały dane pasmo.

Można dodać kolejki ALTQ oraz reguły `pass out` dla interfejsów zapory do segmentów wewnętrznej sieci. Należy zauważyć, że te ograniczenia będą zastosowane tylko dla połączeń zainicjowanych przez zewnętrzne komputery. Połączenia rozpoczęte przez maszyny wewnętrzne oraz zza NAT-u nie będą dopasowane do reguł `pass out`, ponieważ zostaną przepuszczone dzięki mechanizmom filtrowania stanowego wywołanym przez reguły NAT (`nat`, `binat` lub `rdr`). Dlatego właśnie niektórzy użytkownicy będą mogli użyć całego pasma, nawet jeśli ustawi się twarde ograniczenia — ograniczenia stosowane dla pakietów wyjściowych, nie wejściowych. Rozwiązaniem jest umieszczenie routera z systemem OpenBSD przed zaporą, skonfigurowanego jako most (zobacz też rozdział 4. „Konfiguracja OpenBSD") oraz zaimplementowanie reguł filtrowania ALTQ na interfejsie łączącym most z zaporą.

10.2.8. Który algorytm jest najlepszy?

Trudno odpowiedzieć na to pytanie, ale zamieszczony dalej przewodnik pomoże zdecydować, który z algorytmów ALTQ bardziej odpowiada wymaganiom w danym przypadku:

- Gdy trzeba ustalić priorytety kolejek (dla pewnych usług, maszyn użytkowników, którzy są ważniejsi niż inni), należy użyć PRIQ.
- CBQ używa się, gdy istnieje potrzeba podziału pasma na małe składowe, które gwarantują minimalne poziomy przepustowości.

- Jeżeli chce się używać CBQ z równoczesną możliwością dopasowywania się do zmian w ruchu pakietów, używa się HFSC.
- Jeśli istnieje potrzeba nałożenia górnych ograniczeń na użyte pasmo, używa się HFSC.

Nie należy spodziewać się, że ALTQ będzie w 100% dokładny. Doskonale wykonuje kolejkowanie pakietów, lecz trzeba pamiętać, że ruch TCP nie przypomina lejącej się wody z kranu ani nie poddaje się łatwo podziałowi. Ponadto ALTQ pracuje najlepiej wtedy, gdy występują zaległości; w sytuacjach, gdy obciążenie interfejsu jest małe, ALTQ nie ma zbyt wiele pracy. Gdy napotka się problemy, należy uprościć zestaw reguł. Zbyt duża złożoność zawsze odbija się w jakiś sposób na wydajności. Czasami rozwiązaniem będzie zmiana sprzętu (płyty głównej, procesora lub karty sieciowej), czasami wystarczy dopasować parametry `tbrsize`. Należy eksperymentować, ale nie za bardzo, pamiętając, że ALTQ nie jest lekiem na zatkaną sieć, której w wyraźny sposób nie starcza pasma.