# FreeBSD' de PF+ALTQ kullanarak
# Trafik Şekillendirme

/*************************************************************
* **Bâkır EMRE**
* emre [at] enderunix [dot] org
* EnderUNIX Yazılım Geliştirme Takımı
* http://www.enderunix.org
*
* Sürüm : 1.0
* Tarih : 26.04.2006
* Makalenin en yeni versiyonunu
* www.enderunix.org/docs/FreeBSDPfALTQ.pdf
* adresinden elde edilebilir.
*************************************************************/

PF (Packet Filtrer) bilindiği üzere OpenBSD projesiyle özdeşleşmiş paket süzgeci'ne (firewall) verilen isimdir. ALTQ - ALTernate Queueing ise ağ trafiğini meydana getiren paketleri için oluşturulan kuyruk kurallarına verilen isim olarak düşünülebilir.

Trafik şekillendirmeye başlamadan önce bu konu (trafik şekillendirme) hakkında bilinmesi gereken temel bilgileri vermek isterim. Bunlar :

- Packet Queueing (yada queuing) kuyruğa atma yada paket kuyruklama
- Prioritization (önceliklendirme)

# Kuyruğa Atma (queuing)

Bilgisayar ağlarında veri paketleri bir kaynaktan çıktığında işletim sistemi tarafından işlenmek üzere paketler bir kuyruğa girer . İşletim sistemi burada hangi kuyruğun yada hangi paketin işleneceğine karar verir. İşletim sisteminin paketleri işlemek için seçme sırası ağ performansını etkileyecektir. Şöyle düşünelim kullanıcı iki ağ uygulaması çalıştırıyor olsun bunlar ssh ve ftp olsun. Burada ssh paketlerinin, ssh'in zaman-duyarlı yapısından dolayı, ftp paketlerinden önce işlenmesi gerekir. Ssh istemcide tuşlara basıldığı anda acilen karşı tarafın cevap vermesi gerekir. Ftp de ise bir müddet beklenebilir. Şöyle bir varsayım içerisine girersek yönlendiricinin paketleri işlemesi sırasında ssh paketinden evvel büyük miktardaki ftp paketlerini işlesin. Bu durumda ssh bağlantısı kuyrukta kalacaktır kuyruk boyutu bütün paketleri alacak kadar büyük değilse büyük ihtimalle ssh paketleri yönlendirici tarafından atılacaktır yada ssh bağlantısı iyiden iyiye yavaşlayacaktır. Paketleri kuyruğa atma stratejisi bu gibi durumlar için önem kazanıyor.

## A-) Class Based Queueing (Sınıflandırılmış Kuyruklama)

Bu algoritma ile ağ bağlantılarının bağlantı genişliğini birçok sınıfa veya kuyruğa böler. Her bir kuyruk kaynak-hedef ip, port protokol temelli trafiğe atanır. Seçenek olarak bir kuyruk bantgenişliğini kullanmıyorsa onun ebeveyn kuyruğu (kuyruklar içerisinde kuyruklar tanımlanabilir böylece hiyerarşik bir yapı elde edilebilir.) kuyruğu bantgenişliğini ödünç kullanacak şekilde düzenlenebilir. Son olarak bu kuyruklar içersinde öncelik belirtilebilir. Burada dikkat edilmesi gereken şey ise bu önceliklendirmenin her birisi aynı seviyedeki kuyruklar için geçerli olmasıdır.

CBQ kuyruklar yukarıda bahsettiğimiz gibi hiyerarşik bir yapıda düzenlenir. En yukarıdaki kuyruk kök (root) kuyruktur. Kök kuyruğa toplam bantgenişiliğinin tamamı atanır. Alt kuyruklar (child) ise bu kök kuyruğun altında tanımlanır. Alt kuyruklar toplam bantgenişliğinden bir kısmını alacak şekilde atama yapılır

```
Kök kuyruk (2Mbs)
     kuyruk 1(256Kbps)
     kuyruk 2(744Kbps)
     kuyruk 3(1Mbps)
```

burada görüldüğü gibi saniyede 2 megabit 'lik trafik kuyruk 1, kuyruk 2, kuyruk 3 arasında

ihtiyaçlarına binaen paylaştırılmıştır. Bu hiyerarşik yapı biraz daha genişletilebilir.

```
Kök kuyruk (10Mbs)
     kullanıcı A(5Mbps)
          video (4Mbps)
          audio (1Mbps)
     kuyruk 2(3Mbps)
          ftp(2500Kbps)
          http(500Kbps)
     kuyruk 3(1Mbps)
          ssh(50Kbps)
          diger(950Kbps)
```

her bir alt kuyruk kendi kök kuyruklarından daha az bağlantı genişliği kullanacak şekilde ayrılmışlardır.

```
Kök kuyruk (10Mbs)
     kullanıcı A(5Mbps)
          video (4Mbps)
          audio (1Mbps)
     kuyruk 2(3Mbps)
          ftp(2500Kbps,borrow)
          http(500Kbps)
     kuyruk 3(1Mbps)
          ssh(50Kbps)
          diger(950Kbps)
```

yukarıdaki gibi bir yapıda ise ftp için bağlantı genişliği 2500Kbps den fazla olursa kök kuyruktan bantgenişliği ödünç alabilir. Eğer ssh kullanımı artarsa ftp ye verilen ödünç bantgenişliği geri alınır.

## B-) Priority Queuing (Öncelikli kuyruklama)

Öncelikli kuyruklama da ise her biri farklı olan seviyedeki kuyruklara ayrılır. Yüksek önceliğe sahip kuyruklar her zaman düşük öncelikli kuyruklardan önce işlenir.

PRIQ kuyruklama yapısı düzdür yani CBQ de ki gibi hiyerarşik bir yapı gözlenmez. Yine kök kuyruk tanımlanır bu kök kuyruk toplam bant genişliğini tamamını gösterir. Alt kuyruklar yine kök kuyruğun içersimde tanımlanır.

```
Kök kuyruk (2Mbps)
     Kuyruk A(priority 1)
     Kuyruk A(priority 2)
     Kuyruk A(priority 3)
```

Kök kuyruk 2 Mbps lik bant genişliğine sahiptir. Alt kuyruklar her biri farklı bir önceliğe sahip olacak şekilde ayrılmışlardır. Önceliği en yüksek olan kuyruktaki paketleri işlenir daha sonra

işlenmek üzere diğer kuyruklara geçilir. Kuyruk içerisinde paketler işlenirken FIFO mantığı geçerlidir.

## C-) Random Early Detection

RED ise bir tıkanıklık önleyici algoritmadır. Bu algoritmaya göre kuyruk dolmadan ağın tıkanlıklığı önlenmelidir. Bunu yaparken devamlı kuyruğun ortalama boyutu ile iki eşik değerini karşılaştırarak yapar. Bu eşik değerleri ise minimum ve maximum eşik değerleridir. Eğer ortalama kuyruk boyutu minimum eşik değerinden düşük ise Hiçbir paket atılmayacaktır. Ortalama kuyruk boyutu maximum eşik değerinden büyük ise bütün yeni paketler atılacaktır. Son olarak ortalama kuyruk boyutu iki değer arasında ise paketlerin bazıları kuyruğun boyutuna göre rastgele atılacaktır. Diğer deyişle ortalama kuyruk boyutu maksimum eşik değerine yaklaştığında çok daha fazla paket atılacaktır. Paketlerin düşürülmesi sırasında RED paketleri bağlantısını düşürülmesini rastgele seçer. Daha fazla bantgenişliği kullanan paketlerin düşürülme ihtimali daha fazladır

## D-)Explicit Congestion Notification

Explicit Congestion Notification (ECN) ağ üzerinde iki konak haberleşirken haberleşme hattındaki tıkanıklığı bildirmek için RED ile beraber kullanılır. Bunu, paketi düşürmek yerine paket başlığındaki bayrağı ayarlayarak RED i aktif hale getirip yapar. Veri gönderen konağın ECN desteklendiğini varsayıp buradaki bayrağı okuyarak ağ trafiğini yavaşlatabilir.

Bu kadar teorik bilgiden sonra işin pratik kısmına gecelim.
FreeBSD de PF varsayılan olarak çekirdek modulu olarak geliyor. Bu şu demek

```
#kldload pf
```

diyerek pf modulunu sisteme yükleyebilirsiniz. Ve

```
# pfctl –e
```

diyerekte aktif hale getirebilirsiniz.

Ama ALTQ+PF kullanabilmemiz için çekirdek derlemek gerekiyor. Kısaca çekirdek derlemekten bahsetmek gerekirse.

```
# cd /sys/i386/conf/

#cp GENERIC PFALTQ
```

sonra FW dosyasını herhangi bir editör ile açıp içerisine aşağıdaki satırları ekleyelim. Ya da önlerinde # işareti varsa kaldıralım

```
options ALTQ
options ALTQ_CBQ          # Class Bases Queueing
```

```
options ALTQ_RED          # Random Early Drop
options ALTQ_RIO          # RED In/Out
options ALTQ_HFSC         # Hierarchical Packet Scheduler
options ALTQ_CDNR         # Traffic conditioner
options ALTQ_PRIQ         # Priority Queueing
options ALTQ_NOPCC        # Required for SMP build
options ALTQ_DEBUG

device pf
device pflog
device pfsync
```

dosyayı kaydedip çıktıktan sonra,

```
# config PFALTQ
Kernel build directory is ../compile/PFALTQ
Don't forget to do a ``make depend''
# cd ../compile/PFALTQ
# make depend ; make ; make install
```

Sistemi yeniden başlatarak sorunsuz bir şekilde PF+ALTQ sistemini kullanabiliriz

Eğer aşagıdaki satırlarıda `/etc/rc.conf` içersine yazarsanız sistemi bidaha açtığınızda PF ve pf ile alakalı dosyaların yüklenmesi sağlanacaktır.

```
pf_enable="YES"                 # Enable PF (load module if
required)
pf_rules="/etc/pf.conf"         # rules definition file for PF
pf_flags=""                     # additional flags for pfctl
startup
pflog_enable="YES"              # start pflogd(8)
pflog_logfile="/var/log/pflog"  # where pflogd should store the
logfile
pflog_flags=""                  # additional flags for pflogd
startup
```

son olarakta aşağıdaki komutu girerek

```
$ sudo /etc/rc.d/pf start
```

## Kuyruklamanın Yapılandırılması

Alternate Queueing (ALTQ) kuyruklama uygulması OpenBSD 3.0 itibaren ana sistemin parçası olması ile OpenBSD 3.3 ile ALTQ PF içerisine dahil edildi. PF in 2003 temmuzunda FreeBSD'ye port edildi. İlk olarak FreeBSD portlarında yer aldı , kasım 2004'de çıkan FreeBSD 5.3 ile birlikte base sisteme dahil edildi. ALTQ ise haziran 2004 te Freebsd'ye dahiledildi. ALTQ uygulaması Sınıflandırılmış kuyruklamayı ve önceliklendirilmiş kuyruklama yanı sıra rastgele erkenden tespit (RED), açık tıkanıklık duyurusu'nu (ECN) da desteklemektedir. ALTQ PF ile birleştirildiğinden

dolayı kuyruklamanın çalışması için PF muhakkak aktif hale getirilmelidir.

Kuyruklama pf.conf içersinde yapılandırılır. Burada iki tip yönerge vardır. Bunlar:

- `altq on` – kuyruk yapısını belirtilen arayüzde kullanmaya imkân verir. Hangi kuyruk tipinin kullanılacağını ve kök kuyruğun oluşturulması için kullanır.
- `queue` – Alt kuyrukların özelliklerini belirtir.

Bu yapıların sözdizimi aşağıdaki gibidir

1- altq on interface scheduler bandwidth bw qlimit qlim tbrsize size queue \ {queue list}

- altq on :bu satırla bir tane kök kuyruk tanımlamış olduk
- interface : hangi arayüz üzerinde geçerli olacağı
- scheduler :hangi kuyruk tipini kullanacagını (öncelikli yada sınıflandırılmış)
- bandwidth : ne kadar bant genişliği kullanılacak
  - bw : toplam bantgenişliği miktarı yüzde olarakta belritilebilir.
- qlim : kuyrukta tutulacak toplam paket sayısı.
- size : token bucket düzenleyicisi boyutu.
- queue_list : lat kuyrukların listesi

örnek olarak 2 Mbps lik bağlantıyı ssh ftp ve http protokolleri için sınıflandırılmış tipte paylaştırmak istersek.

```
altq  on fxp0 cbq bandwidth 2Mbps queue {http, ftp, ssh}
```

alt kuyruklar için kullanılacak yapıda şu şekildedir.

2- queue name [on interface] bandwidth bw [priority pri] [qlimit qlim] \
scheduler ( sched_options ) { queue_list }

- name – kuyruğun ismi. Bu isim kök kuyruktaki queue_list 'de belirtilen isimlerden biri olmalıdır. Kuyruk ismi 15 karakterden fazla olmaması gerekmektedir.
- interface – kuyruk için kullanılacak tanımlı bir ağ arayüzü. Eğer bir seçenek belirtilmezse bütün arayüzler için geçerli olacaktır.
- bw – kuyruk için kullanılacak toplam bantgenişliği boyutu. Bu parametre cbq kuyrukları için geçerlidir. Eğer tanımlanmaz ise kök kuyruğun toplam bantgenişliğini kullanır.
- pri – kuyruğun önceliği. cbq öncelik aralığı 0 ile 7 arasında iken priq için aralık 0 ile 15 arasıdır. 0 en düşük önceliktir. Tanımlanmadığı zaman değeri 1 dir.
- qlim : kuyrukta tutulacak toplam paket sayısı.Tanımlanmadığı zaman değeri 50 dir.
- scheduler - hangi kuyruk tipini kullanacagını (öncelikli yada sınıflandırılmış)kök kuyruktaki gibi kullanılır.
- sched_options – scheduler için kullanılacak seçenekler.
  - default – varsayılan kuyruğu tanımlar. Buna göre diğer kuyruklar için sıraya girmemiş olan paketler bu kuyruktan geçecektir. Her zaman için bir tane varsayılan kuyruk

gerekmektedir

- red - Random Early Detection (RED)'i kuyruk tarafında kullanılabilir hale getirir.
- rio - RED IN/OUT seçenekleriyle aktif hale getirir. Bu şekliyle RED birkaç tane ortalama kuyruk boyutu ve cok sayıda eşik değerini her bir IP servis kalitesi seviyesi için bir tane sağlar.
- ecn - Explicit Congestion Notification (ECN)i kuyruk tarafında kullanılabilir hale getirir.
- borrow – kök kuyruktan bantgenişliği ödünç almak için kullanılır. cbq scheduler kullanırken kullanılabilir.

- queue_list – alt kuyruklar içerisinde yine alt kuyruk tanımlama için kullanılır. Buda yine cbq scheduler kullanırken kullanılabilir.

Alt kuyruklar için örnek vermek gerekirse :

bant genişliğinin yarısını varsayılan kuyruk olarak kullanacağız. Geri kalan bant genişliğinin yarısını yani toplam bağlantının %25 ini {ssh_login , ssh_data} iki ayrı kuyrukta paylaştırmak isteyelim. Bu listenin dörtte birini ssh_login 'e dörtte üçünü ise ssh_data ya ayıralım. ssh_login önceliği ssh_data dan daha önemli olduğu için onun öncelik numarasını daha büyük gösterelim. Buraki kuyruk tipi olarak sınıflandırılmış kuyruk tipini kullanalım. Birde 500Kb lik bir ftp bağlantısını eğer bant genişiliği müsaitse kök kuyruktan ödünç alacak şekilde ayarlayacağımız kuyruk yapısı aşağıdaki gibidir.

```
queue std bandwidth 50% cbq(default)
queue ssh bandwidth 25% { ssh_login, ssh_bulk }
  queue ssh_login bandwidth 25% priority 4 cbq(ecn)
  queue ssh_bulk bandwidth 75% cbq(ecn)
queue ftp bandwidth 500Kb priority 3 cbq(borrow red)
```

Yukarıda anlatılanları pf.conf uygulayalım şimdi.

fxp0 arayüzünden dışarı çıkacak ssh paketleri için pf kuralına bakacak olursak

```
pass out on fxp0 from any to any port 22
```

şimdi bunu kuyruğa dahil etmek için

```
pass out on fxp0 from any to any port 22 queue ssh
```

dememiz yeterli ama ssh kuyruğunun daha önceden belirtilmiş olması lazım
onuda tanımlarsak

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ssh }
queue std bandwidth 1.5Mb cbq(default)
queue ssh bandwidth 500Kb

pass in on dc0 from any to any port 22 queue ssh
```

burada dikkat edeceğiniz nokta kuyruk `fxp0` üzerinde aktif ediliyor daha sonra paketlerin `dc0` üzerinden geçiyor. Eğer paketler `fxp0` arayüzü üzerinde çıkan pass kuralı ile eşleşirse ssh

kuyruğuna dahil olacaktır.

Eğer pf' ten kuyruk hakkında bilgi almak istersek

```
#pfctl -s queue

queue std bandwidth 1.5Mb cbq(default)
queue ssh bandwidth 500Kb
```

dememiz yeterlidir.

Eğer gerçek zamanlı olarak kuyrukların durumunu görmek istersek:

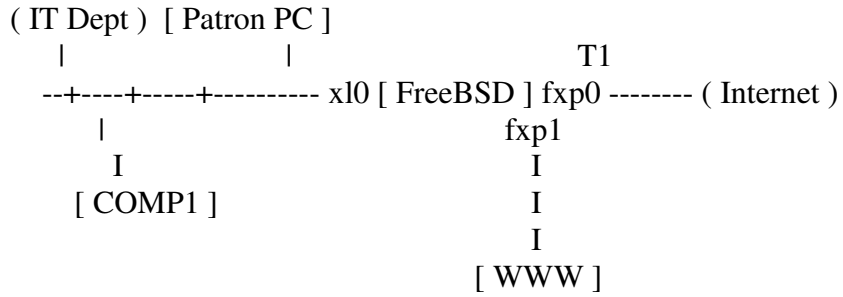
```
#pfctl -vvsq


  [ pkts:       113426   bytes:    10040111   dropped pkts:        0
bytes:      0 ]
  [ qlength:   0/ 50  borrows:      0  suspends:      0 ]
queue  std bandwidth 1.5Mb cbq( default )
  [ pkts:       113426   bytes:    10040111   dropped pkts:   40906
bytes: 3239183 ]
  [ qlength:   0/ 50  borrows:      0  suspends:   4175 ]
queue  ssh bandwidth 500Kb
  [ pkts:            0   bytes:           0   dropped pkts:       0
bytes:      0 ]
  [ qlength:   0/ 50  borrows:      0  suspends:      0 ]
queue  ftp bandwidth 1.02Mb
  [ pkts:            0   bytes:           0   dropped pkts:       0
bytes:      0 ]
  [ qlength:   0/ 50  borrows:      0  suspends:      0 ]
queue  samba bandwidth 100Kb
  [ pkts:            0   bytes:           0   dropped pkts:       0
bytes:      0 ]
  [ qlength:   0/ 50  borrows:      0  suspends:      0 ]
```

şeklinde bir çıktı alabiliriz.

Burada Kb nin Kilobit olduğunu unutmayın.

Son olarak bi kaç örnek verelim.




Bu örnekte FreeBSD üzerindeki pf firewall ile internete çıkacaktır. DMZ de bir adet web server bulunmaktadı. Müşteriler ftp aracılığıyla web sitesine upload yapmaktadır. IT bölümü kendi subnetinden ana ağa bağlanmaktadır. Internet ile 1.5Mbps T1 bağlantısı vardır. Diğer segmentler Fast Ethernet (100Mbps) kullanmakta olduğunu düşünelim. Web sunucusu ile Internet arasındaki trafiği 500Kbps her iki yön için sınırlayalım. HTTP trafiği 250Kbps paylaşsın. Diğer trafikte 250Kbps de kalsın ve herbir kuyruğun 500kpbs nin tamamını ödünç alacak şekilde kullanabilsin internet ile web sunucusu arasındaki HTTP trafiği,yine web sunucusu ile internet arasındaki ftp gibi diğer trafiklerden daha öncelikli olsun. Web sunucusu ile iç ağ arasındaki trafik 100Mbps kullanılabilsin. 500Kbps lik trafiği IT bölümü enson yazılım güncelleştirmeleri için kullansın. Eğer trafik müsaitse daha fazlasınıda kullanabilir. Patron internet trafiği diğer insanların internet kullanımından daha öncelikli olsun. Buna göre kurallarımızı yazarsak.

```
altq on fxp0 cbq bandwidth 1.5Mb queue { std_ext, www_ext,
boss_ext }


# std_ext         - standard kuyruk. Aynı zamanda varsayılan kuyruk
dışarı giden trafik için
#
# www_ext         -  Web server kuyruğu. limiti   500Kbps.
#   www_ext_http - http trafik Web server; yüksek öncelikli
#   www_ext_misc -  WWW serverdan http  dışındaki trafik ftp gibi
# boss_ext        - patronun bilgisayarı için trafik

queue std_ext         bandwidth 500Kb cbq(default borrow)
queue www_ext          bandwidth 500Kb { www_ext_http, www_ext_misc
}
  queue www_ext_http bandwidth 50% priority 3 cbq(red borrow)
  queue www_ext_misc bandwidth 50% priority 1 cbq(borrow)
queue boss_ext        bandwidth 500Kb priority 3 cbq(borrow)


altq on xl0 cbq bandwidth 100% queue { net_int, www_int }


# net_int       - Internet yönünden gelen trafik. Bantgenişiliği
```

```
1.0Mbps.
#     std_int   - standard kuyurk. Aynı zamanda varsayılan   xl0
üzerindeki trafik .
#   it_int    - IT bölümü için  500Kbps.
#   boss_int - patronun bilgisayarı için trafik ,yüksek öncelikli.
# www_int     - WWW server ile  DMZ arasındaki trafik;

queue net_int     bandwidth 1.0Mb { std_int, it_int, boss_int }
  queue std_int  bandwidth 250Kb cbq(default borrow)
  queue it_int   bandwidth 500Kb cbq(borrow)
  queue boss_int bandwidth 250Kb priority 3 cbq(borrow)
queue www_int     bandwidth 99Mb cbq(red borrow)


altq on fxp1 cbq bandwidth 100% queue { internal_dmz, net_dmz }


# internal_dmz    - iç ağdan gelen trafik
# net_dmz         - internetten gelen trafik
#   net_dmz_http - http trafiği; yüksek öncelikli
#    net_dmz_misc - http dışındaki  trafik aynı zamanda varsayılan
kuyruk

queue internal_dmz   bandwidth 99Mb cbq(borrow)
queue net_dmz          bandwidth 500Kb { net_dmz_http, net_dmz_misc
}
  queue net_dmz_http bandwidth 50% priority 3 cbq(red borrow)
  queue net_dmz_misc bandwidth 50% priority 1 cbq(default borrow)



main_net  = "192.168.0.0/24"
it_net    = "192.168.1.0/24"
int_nets  = "{ 192.168.0.0/24, 192.168.1.0/24 }"
dmz_net   = "10.0.0.0/24"

boss      = "192.168.0.200"
wwwserv   = "10.0.0.100"


block on { fxp0, fxp1, xl0 } all


pass in on fxp0 proto tcp from any to $wwwserv port { 21, \
     > 49151 } flags S/SA keep state queue www_ext_misc
pass in on fxp0 proto tcp from any to $wwwserv port 80 \
     flags S/SA keep state queue www_ext_http


pass out on fxp0 from $int_nets to any keep state
pass out on fxp0 from $boss to any keep state queue boss_ext
```

```
pass in on xl0 from $int_nets to any keep state
pass in on xl0 from $it_net to any queue it_int
pass in on xl0 from $boss to any queue boss_int
pass in on xl0 proto tcp from $int_nets to $wwwserv port { 21, 80,
\
     > 49151 } flags S/SA keep state queue www_int

# filter rules for xl0 outbound
pass out on xl0 from xl0 to $int_nets

# filter rules for fxp1 inbound
pass in on fxp1 proto { tcp, udp } from $wwwserv to any port 53 \
     keep state

# filter rules for fxp1 outbound
pass out on fxp1 proto tcp from any to $wwwserv port { 21, \
     > 49151 } flags S/SA keep state queue net_dmz_misc
pass out on fxp1 proto tcp from any to $wwwserv port 80 \
     flags S/SA keep state queue net_dmz_http
pass out on fxp1 proto tcp from $int_nets to $wwwserv port { 80, \
     21, > 49151 } flags S/SA keep state queue internal_dmz
```

Biraz özgün bir örnekte olsada...

ALTQ istenmeyen mail trafiğin bulunması

Son bikaç yıldır maillerle bereber spam veya virus lerin arttığını görmekteyiz. Bunların büyük birçoğunluğunun windows makinelerden geldiği gözlenmekte. Mail server olarak windows makinelerden gelen virus ve spam trafiğinin tüm trafiği engellememesi lazımdır. Bunu için işletim sisteminin ne olduğuda bilinmelidir. PF in ağ bağlantılarından teşhis edebildiği işletim sistemi tiplerini kullanarak bu sorun kısmende olsa çözülebilir.örneği incelersek

```
1- altq on $ext_if cbq queue { q_default q_web q_mail }

2-    queue q_default cbq(default)
3-    queue q_web (...)

       ## all mail limited to 1Mb/sec
4-     queue q_mail bandwidth 1Mb { q_mail_windows }
       ## windows mail limited to 56Kb/sec
5-     queue q_mail_windows bandwidth 56Kb

6-      pass in quick proto tcp from any os "Windows" to $ext_if
port 25 \
            keep state queue q_mail_windows
7-     pass in quick proto tcp from any to $ext_if port 25 label
"smtp" \
            keep state queue q_mail
```

Öncelikle ext_if adındaki arayüze üç adet (`q_default q_web q_mail`) kuyruğa sınıflandırılmış kuyruk (cbq) olarak bölünmüş

Daha sonra bunlar için limitler belirlenmiş. Bütün mail trafiği için 1Mbps
windows makinelerin mail trafiği için ise 56Kbps atanmış

6. kuralda ise

tcp from any os "Windows"

işletim sistemi tipi windows ve port numarası 25 olanların istekleri q_mail_windows adlı kuyruğu atanacağı belirtilmiş. Diğerleri ise q_mail kuyruğunda sıralanacaktır.

## Kaynaklar:


1 - http://www.openbsd.org/faq/pf/
2 - http://pf4freebsd.love2party.net/altq.html
3 - http://solarflux.org/pf
4 - http://www.csl.sony.co.jp/~kjc/software.html
5 - http://www.bgnett.no/~peter/pf