

HOWTO

—

Bandwidth Management for ADSL with OpenBSD

*OpenBSD 4.3 based Router with Bandwidth Management
for PPPOE-based Internet Connections like ADSL*

1.	Introduction.....	1
2.	Install your OpenBSD System.....	2
3.	Configure basic router services.....	4
4.	Firewall configuration with bandwidth management included.....	6
5.	Configure automated updates.....	11
6.	Finishing up.....	12
7.	Adding additional tools.....	12
8.	Monitoring PF.....	13
9.	FAQ.....	13
10.	Possibilities for improvements of this Howto.....	13

1. Introduction

Disclaimer

Following this Howto, you will loose all data on the target systems harddisk drive. For this and all other things that may be going wrong, the author is not responsible. The only one responsible for what you are doing, are you.

This Howto is about...

More common

Building an OpenBSD based router for a small flat-sharing community with four residents. The router should manage the available bandwidth fairly between the paying users and their guests. For the residents it should be possible to use incoming ports for services like Skype or Emule.

The Howto basically addresses IT-Professionals with a rough idea what OpenBSD is about.

More technical

Building an OpenBSD based router for PPPOE-based internet connections including bandwidth management for four users and additional guests. The router will be prepared with static incoming port forwarding for all four users to a fixed range from the IP addresses 192.168.0.10 to 192.168.0.13. Also users, especially guests, can receive dynamic IP addresses through DHCP without port forwarding out of the range 192.168.0.100 to 192.168.0.200. The routers internal interface will be determined to 192.168.0.1.

This Howto assumes an i386 system with two network cards and one harddisk installed, which will be totally used for the new OpenBSD system. For the Howto to completely fit with the described examples OpenBSD 4.3 is needed, but it will work with some small adaptations with OpenBSD 4.2 to 3.7 and hopefully future releases of OpenBSD.

What this Howto is not about...

The Howto is no OpenBSD beginners guide to a highly secured system. And it will not explain the world of OpenBSD.

At second it will not disclaim to guide you to the most stable and hardened OpenBSD system ever build. In fact this Howto does not refer to any OpenBSD hardening possibilities at all (like removing unused parts from the kernel, unnecessarily running daemons, ...).

2. Install your OpenBSD System

Preparations: Hardware and Install-CD

Prepare your systems hardware as you need. At least you should have:

- Two Network Cards (One for the Internet Connection, the other for your Local Area Network)
- One Hard Disk Drive with at least 5 GB capacity (recommended, to be able to update your installation)
- Bootable CD-ROM drive
- The other stuff (like a CPU, memory and ...; for further details see <http://www.openbsd.org/faq/faq1.html#Platforms>)

Afterwards install your OpenBSD depending on your needs. Recommended install sets for this Howto:

- base4x.tgz
- bsd
- comp4x.tgz
- etc4x.tgz
- man4x.tgz

Also it is recommended to provide the content of the packages for future OpenBSD Updates:

- src.tar.gz
- sys.tar.gz

Also suggested: create a bootable CD using the "floppy43.fs" as your 1,44 MB floppy image and store all the above mentioned files onto.

See <http://www.openbsd.org/faq/faq4.html>, if you need some advice to prepare your OpenBSD installation media.

Installing the basic OpenBSD System

Initializing installation

Boot your system from the prepared bootable CD with your preferred install sets.

- Select "i" for install
- Select default "vt220" as your terminal type
- As your "kbd(8) mapping" you may want to select your keyboard layout type, e.g. for a German layout "de"
- Select "y" to proceed with the install
- Select your harddisk drive, by choosing the default ("wd0" or "sd0", which depends on your type of hardware)
- Select "y" to use all of the harddisk for OpenBSD

Disk setup

- To configure the different mount points enter, always accept the default offset and filesystem type:
 - “a a <ENTER> [default offset] <ENTER> 150M <ENTER> 4.2BSD <ENTER> / <ENTER>”
 - “a b <ENTER> [default offset] <ENTER> 300M <ENTER> swap <ENTER>”
 - “a d <ENTER> [default offset] <ENTER> 500M <ENTER> 4.2BSD <ENTER> /tmp <ENTER>”
 - “a e <ENTER> [default offset] <ENTER> 150M <ENTER> 4.2BSD <ENTER> /var <ENTER>”
 - “a f <ENTER> [default offset] <ENTER> 3750M <ENTER> 4.2BSD <ENTER> /usr <ENTER>”
 - “a g <ENTER> [default offset] <ENTER> 150M <ENTER> 4.2BSD <ENTER> /home <ENTER>”
 - ”q <ENTER>”
- Write the labels by selecting default “y”
- Enter “done” as you are asked if the first mount point is correctly set
- Select “y” to proceed with destroying the current data on your harddisk

Network setup

- Name your system
- Select default “y” to configure the network (the names of your network adapters depend on the type of your hardware)
 - You want to initialize the first provided default adapter “<dev1>”
 - Accept the default name
 - Select “n” and do not change the media options
 - As IPv4 address enter “192.168.0.1”
 - As netmask enter the default “255.255.255.0”
 - Next you do not want to configure the second provided default adapter “<dev2>” by entering “done”
- Enter your domain name
- Select “127.0.0.1” as your dns server
- Select default “y” to use the dns server now
- Default IPv4 route is “none”
- Select default “n” as you don’t want to edit hosts
- Select default “n” for no further network configuration

Basic installation

- Enter your root account password twice
- Select “cd” as the location of your OpenBSD sets
- Accept default “cd0” as the CD-ROM location (or in your case any other drive)
- Select “/” as pathname to your OpenBSD sets (or in your case any other path on your CD)
- “y” ignore the missing INSTALL.i386 textfile and use sets anyway
- As all sets are selected by default, enter “done” to accept
- Select default “y” to install the sets
- Ask for other set locations, enter “done”
- Select default “y” to start the SSH daemon

- Select default “none” not to start the NTP daemon
- You do not expect to run a X window system, so select “n”
- Select default “n” not to change the default console
- Choose your timezone, e.g. for a German system “Europe/Berlin”
- Enter “halt” to shutdown your new OpenBSD system

See <http://www.openbsd.org/faq/faq4.html>, if you need some more advice for the OpenBSD installation.

Provide OpenBSD sources

After booting your new OpenBSD system, log in, mount your CD-ROM and extract the OpenBSD sources for future updates in “/usr/src”. This will take **at least 2 GB disk space in “/usr/src”**, so be prepared.

Shell

```
mount_cd9660 /dev/cd0c /mnt
cd /usr/src
tar xvzf /mnt/src.tar.gz
tar xvzf /mnt/sys.tar.gz
```

3. Configure basic router services

Automatic service startup preparations

After the basic OpenBSD system is set up, configure the basic router services. First activate IP Forwarding in the kernel. While editing “/etc/sysctl.conf” also activate the routing of GRE packages for PPTP VPN tunnels.

/etc/sysctl.conf

```
...
net.inet.gre.allow=1
net.inet.ip.forwarding=1
...
```

Then allow the firewall to start in “/etc/rc.conf.local”. Also activate the nameserver, timeserver, ftp proxy and DHCP server. And as we are already there: deactivate quotas, inetd and sendmail, as we will not need them.

/etc/rc.conf.local

```
named_flags=""
ntpd_flags=""
dhcpcd_flags=""
ftpproxy_flags=""
sendmail_flags=NO
pf=YES
inetd=NO
check_quotas=NO
```

Edit your crontab to deactivate the sendmail cronjob.

Shell

```
crontab -e
```

crontab

```
...
#*/30 * * * * /usr/sbin/sendmail -L sm-msp-queue -Ac -q
...
```

Network service configurations

All services are prepared for their autostart. Now each of them needs to get configured before starting it. First prepare the nameserver by creating the file "/var/named/named.boot".

```
/var/named/named.boot
options forward-only
forwarders <ip1> <ip2> <...>
```

Where <ip1>, <ip2> and so on has to be replaced by at least one of your providers official name servers. For the German provider Arcor you could use these four IPs "145.253.2.11 145.253.2.75 195.50.140.114 195.50.140.252", for example.

Also check the configuration in "/etc/resolv.conf".

```
/etc/resolv.conf
lookup file bind
nameserver 127.0.0.1
```

Then prepare the timeserver to get the accurate time. This is useful, if you wish to use automated tasks, e.g. for updating your OpenBSD and also if you provide the accurate time to your network clients. Edit the file "/etc/ntp.conf" and initially set the accurate time.

```
/etc/ntp.conf
...
listen on *
servers pool.ntp.org
...
```

```
shell
rdate -ncv pool.ntp.org
```

Now be ready to edit "/etc/dhcpd.conf" to configure the DHCP server. Here we assume the internal interface of this router got the IP address 192.168.0.1 and the DCHP clients should get IP addresses from the range 192.168.0.100 to 192.168.0.200.

```
/etc/dhcpd.conf
...
shared-network LOCAL-NET {
    option domain-name "wg.local";
    option domain-name-servers 192.168.0.1;

    subnet 192.168.0.0 netmask 255.255.255.0 {
        option routers 192.168.0.1;
        option time-servers 192.168.0.1;

        range 192.168.0.100 192.168.0.200;
    }
}
...
```

After all the PPPOE connection to your internet service provider has to be established. This is done via creating the file "/etc/hostname.pppoe0" and "/etc/hostname.<dev2>".

```
/etc/hostname.pppoe0
inet 0.0.0.0 255.255.255.255 0.0.0.1 pppoe0dev <dev2> \
    authproto pap authname <user> authkey <password> up
!/sbin/route add default 0.0.0.1
```

```
/etc/hostname.<dev2>
up
```

Just replace the string <dev2> with your network cards device name for one of your cards, also replace <user> and <password> with your account data from your provider.

4. Firewall configuration with bandwidth management included

Now to get to the core of the matter: the firewall ruleset with its bandwidth management rules. To implement some rules for your routed traffic edit the file `"/etc/pf.conf"`.

Part 1: Default settings for PF

First define some basics, like the interfaces as macros, known user hosts, reserved user port ranges, bandwidth, some network address sets and some commonly used ports you want to use in your ruleset. Then define the default block policy and your logging interface. After all define to reassemble your traffic travelling through the router.

Don't forget to insert your network cards device name from the currently unused card instead of the string `<dev1>` and your upstream and downstream bandwidth. Fill in your bandwidth size for `<upstream>` and `<downstream>` in kilobits (**! bits, not bytes !**). For example for a standard DSL2000 line in Germany you would use: 192Kb for upstream and 2048Kb for downstream.

```
/etc/pf.conf → PART 1
### MACROS & TABLES ###
#
#Define all interfaces
#
ext_if="pppoe0"
int_if="<dev1>"

#
#Define favoured client hosts
#
host_usr1="192.168.0.10"
host_usr2="192.168.0.11"
host_usr3="192.168.0.12"
host_usr4="192.168.0.13"

#
#Define reserved client port ranges
#
ports_usr1="1000:1099"
ports_usr2="1100:1199"
ports_usr3="1200:1299"
ports_usr4="1300:1399"

#
#Define available bandwidth
#
bnd_upstream="<upstream>"
bnd_downstream="<downstream>"

#
#Define privileged network address sets
#
nets_priv = "{ 127.0.0.0/8 192.168.0.0/16 172.16.0.0/12 10.0.0.0/8 }"

#
#Define ports for common internet services
#
ports_web = "{ 80 8080 443 25 110 143 993 }"

### OPTIONS ###
#
#Default behaviour
#
##Define default response for block filters
set block-policy drop
##Define statistics logging on
set loginterface $ext_if

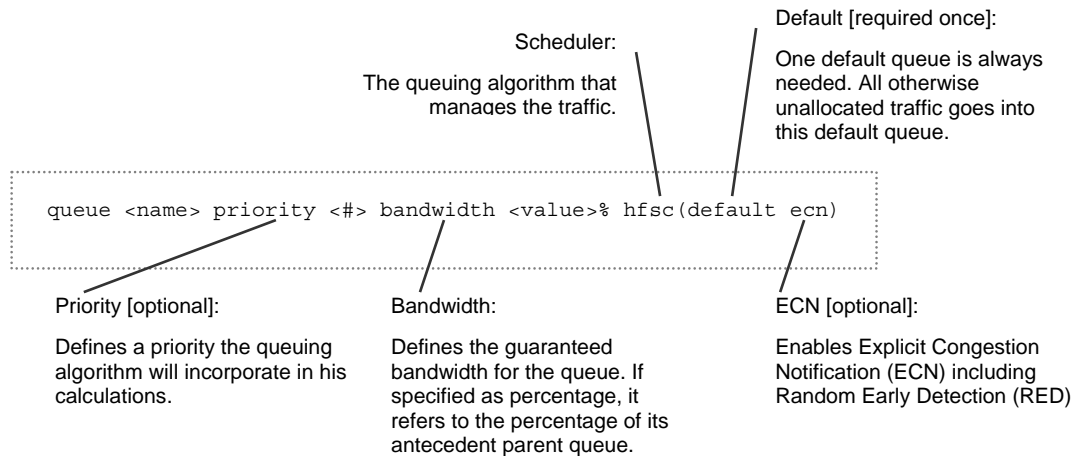
### TRAFFIC NORMALIZATION ###
#
#Filter traffic for unusual packets
#
scrub in all
```

Part 2: The queuing phase I

Bandwidth management consists of two phases: In phase one all queues will be defined, so we have something to sort your traffic in.

Queuing basics

Let's take a short look at how queues are defined:



Note: Unassigned or unused bandwidth is always split up fairly between all currently active queues, e.g. if just two users are active and they both demand full bandwidth, each of them will get 50% of the whole bandwidth. So at least each user gets what is defined in his queues plus additionally his fair part of all the unused bandwidth currently available from other queues.

If you are more deeply interested in how queuing works, take a look at <http://www.openbsd.org/faq/pf/queueing.html> and the pf.conf manual pages under <http://www.openbsd.org/cgi-bin/man.cgi>.

Parent queues

Let's start with the outgoing and incoming parent queues. Remember: It is technically only possible to manage traffic on its way out of the router. So we have to manage the outgoing traffic on the external interface (because here this traffic leaves towards the internet). And according to this, we have to manage the incoming traffic on the internal interface (because here this traffic leaves towards your local area network).

Define the queues using the hfsc scheduler. Additionally declare your first line of child queues. In our case we define one default queue and four user queues, each for incoming and outgoing traffic.

Child queues

After the parent queues are ready, the individual child queues are planned. Basically each child queue consists of three subqueues. One subqueue for the child default traffic, one for the prioritized traffic to often used ports and another subqueue for highly prioritized traffic like TCP acknowledges.

`/etc/pf.conf` → PART 2

```
### QUEUEING ###
#
#Define parent queues
#
##Define upstream parent queue
altq on $ext_if hfsc bandwidth $bnd_upstream queue { up_def up_usr1 up_usr2 up_usr3 up_usr4 }
##Define downstream parent queue
altq on $int_if hfsc bandwidth $bnd_downstream queue { dn_def dn_usr1 dn_usr2 dn_usr3 \
    dn_usr4 }
```

```

#
#Define upstream child queues
#
##Define default upstream queue
queue up_def bandwidth 20% { up_def_def up_def_web up_def_quick }
  queue up_def_def priority 1 bandwidth 50% hfsc(default ecn)
  queue up_def_web priority 3 bandwidth 25% hfsc(ecn)
  queue up_def_quick priority 6 bandwidth 25% hfsc(ecn)
##Define User1 upstream queue
queue up_usr1 bandwidth 20% { up_usr1_def up_usr1_web up_usr1_quick }
  queue up_usr1_def priority 2 bandwidth 50% hfsc(ecn)
  queue up_usr1_web priority 4 bandwidth 25% hfsc(ecn)
  queue up_usr1_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User2 upstream queue
queue up_usr2 bandwidth 20% { up_usr2_def up_usr2_web up_usr2_quick }
  queue up_usr2_def priority 2 bandwidth 50% hfsc(ecn)
  queue up_usr2_web priority 4 bandwidth 25% hfsc(ecn)
  queue up_usr2_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User3 upstream queue
queue up_usr3 bandwidth 20% { up_usr3_def up_usr3_web up_usr3_quick }
  queue up_usr3_def priority 2 bandwidth 50% hfsc(ecn)
  queue up_usr3_web priority 4 bandwidth 25% hfsc(ecn)
  queue up_usr3_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User4 upstream queue
queue up_usr4 bandwidth 20% { up_usr4_def up_usr4_web up_usr4_quick }
  queue up_usr4_def priority 2 bandwidth 50% hfsc(ecn)
  queue up_usr4_web priority 4 bandwidth 25% hfsc(ecn)
  queue up_usr4_quick priority 7 bandwidth 25% hfsc(ecn)

#
#Define downstream child queues
#
##Define default downstream queue
queue dn_def bandwidth 20% { dn_def_def dn_def_web dn_def_quick }
  queue dn_def_def priority 1 bandwidth 50% hfsc(default ecn)
  queue dn_def_web priority 3 bandwidth 25% hfsc(ecn)
  queue dn_def_quick priority 6 bandwidth 25% hfsc(ecn)
##Define User1 downstream queue
queue dn_usr1 bandwidth 20% { dn_usr1_def dn_usr1_web dn_usr1_quick }
  queue dn_usr1_def priority 2 bandwidth 50% hfsc(ecn)
  queue dn_usr1_web priority 4 bandwidth 25% hfsc(ecn)
  queue dn_usr1_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User2 downstream queue
queue dn_usr2 bandwidth 20% { dn_usr2_def dn_usr2_web dn_usr2_quick }
  queue dn_usr2_def priority 2 bandwidth 50% hfsc(ecn)
  queue dn_usr2_web priority 4 bandwidth 25% hfsc(ecn)
  queue dn_usr2_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User3 downstream queue
queue dn_usr3 bandwidth 20% { dn_usr3_def dn_usr3_web dn_usr3_quick }
  queue dn_usr3_def priority 2 bandwidth 50% hfsc(ecn)
  queue dn_usr3_web priority 4 bandwidth 25% hfsc(ecn)
  queue dn_usr3_quick priority 7 bandwidth 25% hfsc(ecn)
##Define User4 downstream queue
queue dn_usr4 bandwidth 20% { dn_usr4_def dn_usr4_web dn_usr4_quick }
  queue dn_usr4_def priority 2 bandwidth 50% hfsc(ecn)
  queue dn_usr4_web priority 4 bandwidth 25% hfsc(ecn)
  queue dn_usr4_quick priority 7 bandwidth 25% hfsc(ecn)

```

Each user child queue is designed identically, only the default child queue uses less priority for its subqueues.

If you need to take an in-depth look to bandwidth management, try your luck with

<http://www.openbsd.org/faq/pf/queueing.html> or <http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf> under chapter 'QUEUEING'.

Part 3: Necessary stuff as filling

Before we proceed with bandwidth management, we need some other stuff coming next. Define your network address translation (**don't forget the brackets around \$ext_if**). Also define redirections, if you need them. In our case, each client gets a fixed range of ports for less administrative overhead.

Afterwards create the basic blocking rule for your firewall and pass all local traffic around.

/etc/pf.conf → PART 3

```
### TRANSLATION ###
#
#NAT for the external traffic
#
##Mask internal ip addresses with actual external ip address
nat on $ext_if from $int_if:network to any -> ($ext_if)
##Anchor for FTP proxy
nat-anchor "ftp-proxy/*"

#
#Redirections
#
##Redirect FTP clients to FTP proxy
rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 port 8021
##Anchor for FTP proxy
rdr-anchor "ftp-proxy/*"
##Redirect client ports
rdr on $ext_if proto { tcp udp } from any to ($ext_if) port $ports_usr1 -> $host_usr1
rdr on $ext_if proto { tcp udp } from any to ($ext_if) port $ports_usr2 -> $host_usr2
rdr on $ext_if proto { tcp udp } from any to ($ext_if) port $ports_usr3 -> $host_usr3
rdr on $ext_if proto { tcp udp } from any to ($ext_if) port $ports_usr4 -> $host_usr4

### PACKET FILTERING ###
#
#Default filter
#
block log all

#
#Loopback interface traffic
#
pass quick on lo0 all
```

Part 4: The queuing phase II

Remember: Bandwidth management consists of two phases. After we defined our queues in phase one, here in phase two we use them to fill in the appropriate traffic.

We start with the rules for the outgoing traffic on the external interface (*Upstream*). Before really starting, we're blocking traffic with addresses from privileged networks. Also we're allowing traffic to the ftp proxy server and 'ping' requests to our router from the internet.

Now to the queuing one the external interface itself: It works quite simple on this interface, as we assume some of the packets we need to queue got tagged in a mysterious way.

So first we sort out traffic to DNS servers and pipe them into the 'up_def_quick' queue. Afterwards we take care off all the tagged packets from our users and pipe them into the appropriate 'up_usr...' queues. Out of the traffic left we separate the packets to the often used ports for 'up_def_web' from the remainder 'up_def_def' and pipe them in their primed queues. Additionally we configure PF to pipe highly prioritized packets into 'up_def_quick'.

/etc/pf.conf → PART 4a

```
#
#Filter and queue external interface traffic
#
##Deny incoming or outgoing privileged network address sets
block in quick on $ext_if from $nets_priv to any
block out quick on $ext_if from any to $nets_priv
##Allow incoming ping request to router
pass in quick on $ext_if inet proto icmp from any to ($ext_if) icmp-type 8 code 0
##Allow incoming traffic to privileged port ranges for special users
pass in on $ext_if inet proto { tcp udp } from any to port \
    { $ports_usr1 $ports_usr2 $ports_usr3 $ports_usr4 }
##Assign upstream traffic to queues
pass out on $ext_if queue (up_def_def up_def_quick)
pass out on $ext_if proto { tcp udp } from any to any port $ports_web \
    queue (up_def_web up_def_quick)
```

```

pass out on $ext_if proto { tcp udp } from any to any port domain \
    queue up_def quick
pass out on $ext_if tagged UP_USR1_DEF queue up_usr1_def
pass out on $ext_if tagged UP_USR1_WEB queue up_usr1_web
pass out on $ext_if tagged UP_USR2_DEF queue up_usr2_def
pass out on $ext_if tagged UP_USR2_WEB queue up_usr2_web
pass out on $ext_if tagged UP_USR3_DEF queue up_usr3_def
pass out on $ext_if tagged UP_USR3_WEB queue up_usr3_web
pass out on $ext_if tagged UP_USR4_DEF queue up_usr4_def
pass out on $ext_if tagged UP_USR4_WEB queue up_usr4_web

```

We continue with the outgoing traffic, but this time on the internal interface (still **Upstream**). Here we start tagging the traffic from our users with different tags according to their later queue affiliation. Note: This is necessary as we're losing the original address through the network address translation on the external interface. Also allow all other incoming traffic from the internal network.

Afterwards care about the incoming traffic on the internal interface (**Downstream**). As before we first sort out traffic to DNS servers and pipe them into the 'dn_def_quick' queue. Afterwards we take care off packets addressed to our users and pipe them into the appropriate 'dn_usr...' queues. Out of the traffic left we separate the packets to the often used ports for 'dn_def_web' from the remainder 'dn_def_def' and pipe them in their primed queues. Additionally we configure PF to pipe highly prioritized packets into 'dn_def_quick'.

/etc/pf.conf → PART 4b

```

#
#Filter and queue internal interface traffic
#
##Allow other incoming traffic from internal network
pass in on $int_if from $int_if:network to any no state
##Allow incoming traffic from special users and add tags for some upstream queues
pass in on $int_if from $host_usr1 to any no state tag UP_USR1_DEF
pass in on $int_if proto { tcp udp } from $host_usr1 to any port $ports_web no state \
    tag UP_USR1_WEB
pass in on $int_if from $host_usr2 to any no state tag UP_USR2_DEF
pass in on $int_if proto { tcp udp } from $host_usr2 to any port $ports_web no state \
    tag UP_USR2_WEB
pass in on $int_if from $host_usr3 to any no state tag UP_USR3_DEF
pass in on $int_if proto { tcp udp } from $host_usr3 to any port $ports_web no state \
    tag UP_USR3_WEB
pass in on $int_if from $host_usr4 to any no state tag UP_USR4_DEF
pass in on $int_if proto { tcp udp } from $host_usr4 to any port $ports_web no state \
    tag UP_USR4_WEB
##Assign outgoing traffic from other interfaces to queues for downstream
pass out on $int_if no state queue (dn_def_def dn_def_quick)
pass out on $int_if proto { tcp udp } from any port $ports_web to any no state \
    queue (dn_def_web dn_def_quick)
pass out on $int_if proto { tcp udp } from any port domain to any no state queue dn_quick
pass out on $int_if from any to $host_usr1 no state queue dn_usr1_def
pass out on $int_if proto { tcp udp } from any port $ports_web to $host_usr1 no state \
    queue dn_usr1_web
pass out on $int_if from any to $host_usr2 no state queue dn_usr2_def
pass out on $int_if proto { tcp udp } from any port $ports_web to $host_usr2 no state \
    queue dn_usr2_web
pass out on $int_if from any to $host_usr3 no state queue dn_usr3_def
pass out on $int_if proto { tcp udp } from any port $ports_web to $host_usr3 no state \
    queue dn_usr3_web
pass out on $int_if from any to $host_usr4 no state queue dn_usr4_def
pass out on $int_if proto { tcp udp } from any port $ports_web to $host_usr4 no state \
    queue dn_usr4_web

```

Part 5: Antispoofing rules

To prevent spoofing create the required rules in your ruleset.

/etc/pf.conf → PART 5

```

#
#Anchor for FTP proxy
#
anchor "ftp-proxy/*"

#
#Deny spoofing
#
antispoof for $ext_if
antispoof for $int_if

```

5. Configure automated updates

Create scripts for automation

The CVS synchronization script

First create a script “/root/update_part1.sh” to synchronize your OpenBSD sources with the actual source tree. Change to your source tree directory, set the CVS configuration parameters, start CVS update process and log scripts invocation. Ensure that the CVS invocation parameter ‘-r’ matches your OpenBSD version.

```
/root/update_part1.sh
#!/bin/csh
cd /usr/src
setenv CVS_CLIENT_PORT -1
setenv CVSRROOT anoncvs@anoncvs.openbsd.org:/cvs
cvs -d $CVSRROOT -q up -rOPENBSD_4_3 -Pd
date > /root/update_part1.log
```

Note: You will have to authenticate the CVS server upon first execution of the script. So run the script once by hand, before you use it within a cronjob.

The kernel compilation script

Then create a script “/root/update_part2.sh” to compile a new kernel from your updated OpenBSD sources. Change to your source tree directory, prepare and start the compilation, exchange the current kernel, log scripts invocation and reboot OpenBSD.

```
/root/update_part2.sh
#!/bin/csh
cd /usr/src/sys/arch/i386/conf
/usr/sbin/config GENERIC
cd /usr/src/sys/arch/i386/compile/GENERIC
make clean && make depend && make
cp /bsd/bsd.old
cp bsd /bsd
date > /root/update_part2.log
reboot
```

Note: If you do not use an i386 platform, you should change the according directory paths.

All the rest compilation script

Create a script “/root/update_part3.sh” to compile all other system parts from your updated OpenBSD sources. Change to your source tree directory, prepare and start the compilation, log scripts invocation and reboot OpenBSD.

```
/root/update_part3.sh
#!/bin/csh
cd /usr/src
rm -r /usr/obj/*
make obj && make build
date > /root/update_part3.log
reboot
```

Update application packages script

At least create a script “/root/update_part4.sh” to update installed OpenBSD application packages. Set the path to your preferred mirror server and execute the update command for the application packages.

```
/root/update_part4.sh
#!/bin/csh
setenv PKG_PATH ftp://ftp.openbsd.org/pub/OpenBSD/4.3/packages/i386/
pkg_add -ui -F update -F updatedependends
date > /root/update_part4.log
reboot
```

Configure automated script execution

Before we can execute the script, we have to set proper file permissions for all scripts.

```
Shell
chmod ug+x update_part*.sh
```

Use the cron daemon to automate the script invocation. Edit your crontab to configure script automation.

```
Shell
crontab -e
```

Invoke the “update_part1.sh” each first of a month at three o’clock a.m.. Accordingly invoke “update_part2.sh” each third of a month and on the fifth “update_part3.sh”.

```
crontab
...
0 3 1 * * /root/update_part1.sh
0 3 3 * * /root/update_part2.sh
0 3 5 * * /root/update_part3.sh
0 3 7 * * /root/update_part4.sh
...
```

6. Finishing up

Reboot your system to start all newly configured daemons.

7. Adding additional tools

Note: For this action your internet connection should be ready for usage.

If you like to improve the monitoring possibilities for PF, add pfTop to your installation. pfTop is a curses-based utility for real-time display of active states and rules for PF. Therefore install the current release of the pfTop packet form an OpenBSD mirror server.

```
Shell
export PKG_PATH=ftp://ftp.openbsd.org/pub/OpenBSD/4.3/packages/i386/
pkg_add pftop-0.7.tgz
```

See <http://www.openbsd.org/ftp.html>, if you need an overview on current mirror servers for OpenBSD. And take a look at http://www.openbsd.org/4.3_packages/ for the current version of pfTop for your platform.

Note: If you do not use an i386 platform, you should change the according directory paths.

If you like to install more than one package, think about adding the PKG_PATH variable to your profile. So you will not need to specify this every time you are working with packages.

```
/root/.profile
...
PKG_PATH=ftp://ftp.openbsd.org/pub/OpenBSD/4.3/packages/i386/
export PKG_PATH
...
```

8. Monitoring PF

PFTOP

Show current usage of all queues	pftop -v queue
----------------------------------	----------------

PFCTL

List usage of all queues	pfctl -vsq
--------------------------	------------

TCPDUMP

Show current PF logging	tcpdump -n -e -ttt -i pflog0
-------------------------	------------------------------

Show PF logfile	tcpdump -n -r /var/log/pflog
-----------------	------------------------------

Show current traffic on PPPOE-Interface	tcpdump -n -i pppoe0
---	----------------------

9. FAQ

I have got less users/ computers as the tutorial describes?

Than you are perfectly ok with the defaults in the tutorial. Why? Because the queueing algorithm fairly spreads unused bandwidth between the existing users.

I have got more users/ computers as the tutorial describes?

Well, than hopefully you are an experienced computer administrator. Start to understand how the PF configuration in this tutorial works and rewrite it by adding more users and according queues and rules.

Do not ask me to do this for you, except your willing to pay for? ☺

What are those reserved client port ranges good for?

The reserved port ranges are for incoming ports as they are needed by applications like Emule or Skype. If you're running an application that needs to receive data from other applications directly, they often demand you to specify an incoming port they could use for this communication. For this, just take on of the preassigned ports from the reserved clients port range.

If you can't think of applications using incoming ports, you probably don't need this feature.

10. Possibilities for improvements of this Howto

In general: If you email your improvements, they can get part of the next version of this Howto.

PF rules

Feel free to add some more macros to the PF ruleset.

CVS update mechanism

Feel free to describe a more efficient way for your OpenBSD update, especially to avoid compiling the sources, if there was no source update released. Or only compile this part of the sources that actually changed.

More useful tools

If someone misses an often used tool for this use case of OpenBSD, please feel free to give me some hint.