INASP: Effective Network Management Workshops

Unit 11: Technical Measures

About these workshops

Authors:

- Dick Elleray, AfriConnect
    - delleray@africonnect.com
- Chris Wilson, Aptivate
    - chris+inaspbmo2013@aptivate.org

Date: 2013-04-29

Table of Contents

About these workshops 1

Web Proxies and Caches 4

Objectives 4

License 4

Introduction 4

What is a web proxy? 4

Forward and Reverse Proxies 5

Why use web proxies? 5

Benefits of using a web proxy 5

What is a web cache? 6

Why use web caches? 6

Why not to use web caches? 6

Not transparent 6

Effectiveness is falling 6

Hardware requirements 6

Single point of failure 7

Getting started with Squid 7

Basic installation 7

Configuring your browser 7

Testing the installation 8

Access control by IP address 8

Why do you deny me? 9

Reading the logs 9

Don't deny me! 10

Reloading and restarting Squid 11

Reverse proxies and open proxies 11

Cache Size 12

Disk cache size 12

Memory usage 12

Squid Access Control 13

Access control elements 13

ACE types 13

The `srcdomain` ACE: a special case 13

ACEs with multiple values 14

Access control rules 14

Rules with multiple ACEs 14

- Rule processing examples 15
- Access control practice 15
- Solutions 15

Web Proxies and SSL **17**

- What can we do about it? 17
- HTTP and CONNECT requests 17
- Results of blocking SSL requests 17

Forcing people to use the proxy **18**

- Configure pfSense as your router 18
- Proxy auto configuration 19
- Creating a PAC file 20
- DHCP server settings in pfSense 20
- Testing Proxy Auto Configuration 21

Proxy Authentication **21**

- About RADIUS 21
- Setting up a RADIUS Server 21
- Installing FreeRADIUS on pfSense 22
- Configuring FreeRADIUS 22
- Adding Users 22
- Testing RADIUS Authentication 23
- Squid RADIUS Authentication 23

Squid Delay Pools **24**

- Classes of delay pools 25
- Request routing 25
- Limitations of pools 26
- Simple example 26
- More advanced configuration 27

FIN **27**

- Objectives 27
- License 27

Introduction **27**

- What is bandwidth management? 27
- Wired Magazine’s take 28
- What are the limitations? 28
- What can we do with pfSense? 29
- Why use pfSense? 29
- How do we start? 29
- Kilobits and kilobytes 29

Example configuration 30
Bandwidth Allocation 31
How HFSC Works 31
Configure pfSense as your router 32
Configure the Interfaces 32
Testing 35
Traffic and Ping times 35
Adding more queues 35
Filtering traffic into queues 36
Adding filtering rules 36
Testing 38
Classifying inbound connections 39
FIN **39**

Web Proxies and Caches

Objectives

On completion of this session, we hope you will be able to:

- Install and configure a Squid web cache

If you are the facilitator, please tell the group:

> At the end of session I will ask if we have met the objectives – if not, we will discuss again.

License

Some materials reused under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 license:

- the Web Caching manual, by Richard Stubbs of TENET;
- the BMO Book, by various authors;
- the Squid Cache Wiki, by Amos Jeffries and other.

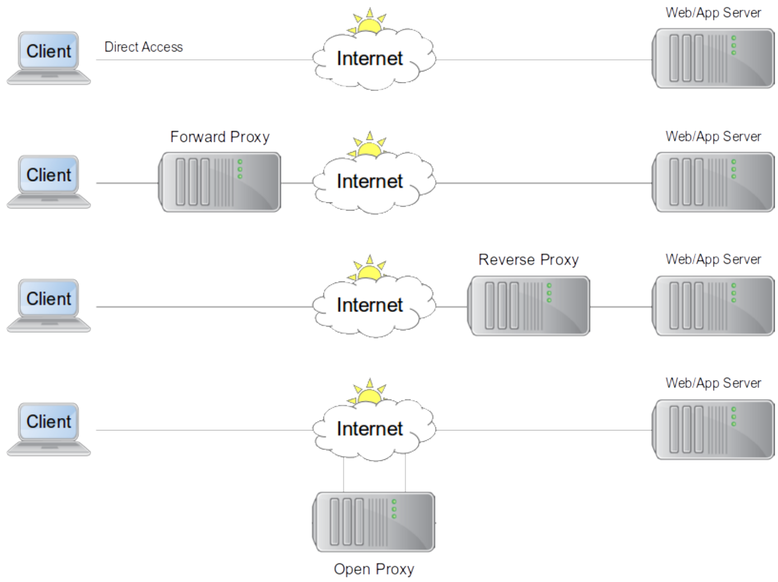
Introduction

What is a web proxy?

A proxy is a person or thing that acts on behalf of another person or thing.

A web proxy fetches web pages “on your behalf”. So when you want to access a particular page, instead of requesting it directly, you ask the proxy to request it for you.

Forward and Reverse Proxies



Forward proxy
: Operated by the client's organisation, used by specific clients to connect to (usually) all web sites (servers).

Reverse proxy
: Operated by the server's organisation, used by (usually) all clients to connect to specific web sites (servers).

Open proxy
: Usually operated by a third party, used by any client to connect to any web server, potentially dangerous/exploitable.

Why use web proxies?

Web proxies can:

- Require you to log in and authenticate yourself to the proxy.
- Log the web page that you requested.
- Block access to the web page.
- Scan the content for viruses.
- Scan the content for obscenities or banned content.
- Serve a local cached copy of the content.

All of these things can be desirable in an institutional environment, depending on how strict you want to be in denying or logging web accesses.

Benefits of using a web proxy

For users
: They can filter out viruses and other dangeous content. Users may also be forbidden from directly accessing the Internet by site policy, and must therefore use a proxy for all web requests.

For administrators
: They allow authenticating users, logging and inspecting the content of requests, associating a user account with a request, and filtering out dangerous or banned content. As reverse proxies, they can share public IP addresses between multiple

independent applications.

Some reverse proxies are more efficient at serving static content than most web servers and application servers. You might need to run Apache to host your application, but Squid, Nginx or Lighttpd would intercept requests for static content, reducing the load on the application server.

What is a web cache?

The term cache literally means to store. In computing terms caching is the act of storing information on a local system, where the act of retrieving the information from the local cache is less than the cost of retrieving the information from the original source.

A web cache is a proxy that can cache copies of downloaded pages and files, and serve them automatically, following the rules for caching HTTP requests. This is very important because it ensures that the cache doesn't serve stale content, which could break web applications.

Why use web caches?

For users

They can return returns faster than accessing the Internet, if the requested document is already cached.

For administrators

Web caches can reduce your inbound bandwidth needs by up to 40% of your web traffic.

Why not to use web caches?

AKA: limitations of web caches

Not transparent

Each computer needs to be configured to use the proxy (or you need to use network tricks such as PAC or interception, described later).

Effectiveness is falling

More and more content is dynamic (not cacheable) and/or served over SSL. Proxies add overhead to requests for dynamic content, and usually can't intercept SSL connections as that would invalidate the security certificate on the connection.

Hardware requirements

A web cache requires a fairly fast server with a lot of disk space to be effective:

- Limit the number of simultaneous web requests from all users to the capacity of the proxy/cache (usually 50-100 for Squid).
- Slower CPUs will add more overhead to each request.
- Need enough disk space to be effective, otherwise the cache hit rate will fall, so more requests are slowed down and fewer are accelerated.
- Need enough RAM for OS to cache commonly used cache objects and directories, otherwise the disk accesses will add overhead to every request.

Note: reverse proxies for static content perform much better than Squid, if the static files are accessible to the proxy via a shared filesystem.

Single point of failure

If all web requests pass through a single server, then if that server fails, all web requests will fail.

Proxies are good candidates for replication and load balancing, as they are usually stateless (apart from the cache, but that only affects performance).

However, Squid's high hardware requirements make it expensive to replicate if you have a busy network.

Getting started with Squid

Basic installation

To install Squid on an Ubuntu or Debian system:

```
$ sudo apt-get install squid3
$ service squid3 status
```

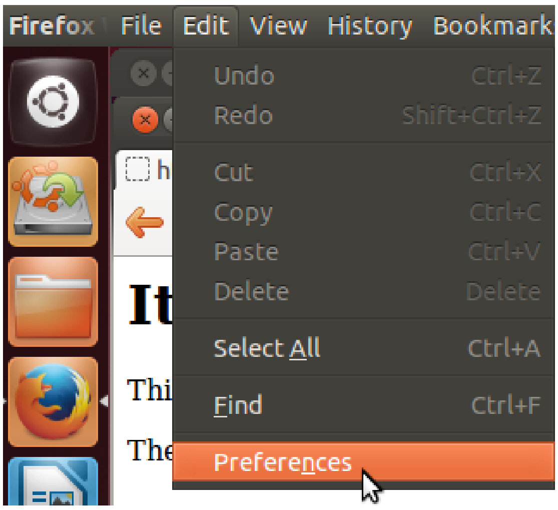
If you're using the Ubuntu 12.04 Live CD, it may fail to start due to a bug in the Ubuntu 12.04.3 Live CD. Then you need to run these commands:

```
$ sudo initctl reload-configuration
$ sudo start squid3
```

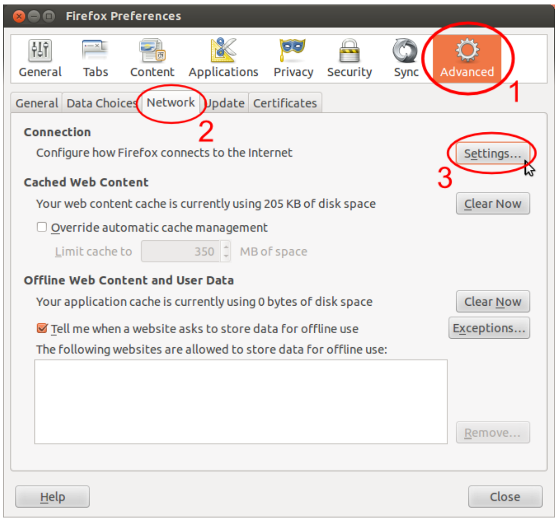
Configuring your browser

Reconfigure your web browser to use the proxy.

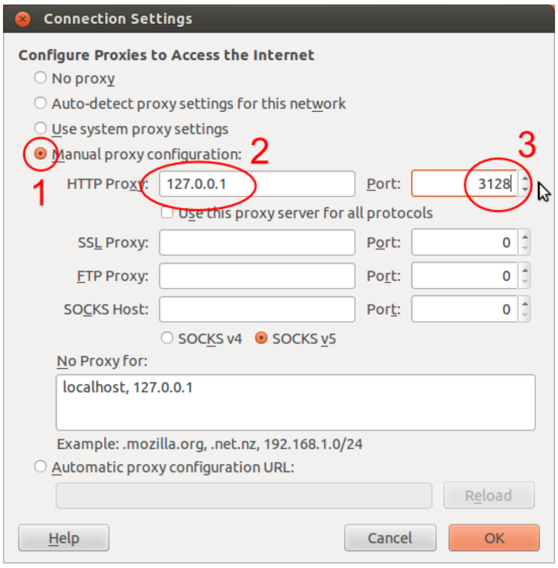
In Firefox for example, go to Edit/Preferences:



Then go to the Advanced tab, under that choose Network, and click on the Connection/Settings button:



- Choose *Manual proxy configuration*;
- For *HTTP proxy* enter `127.0.0.1`, assuming that you want to connect to Squid running on the same host;
- For *Port* enter 3128, the default port for Squid.



Testing the installation

Now try to access a website in the browser. What happens?

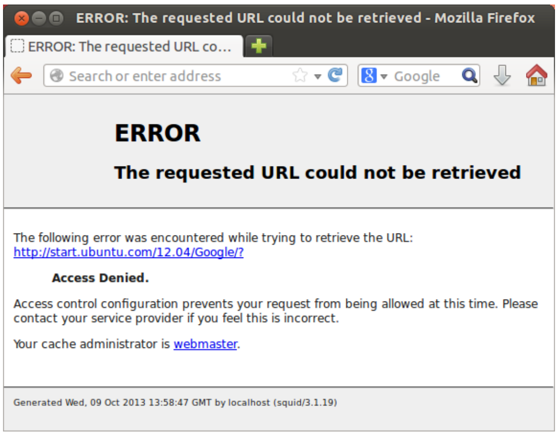
How can you tell if you're using the proxy? Look at the logs:

```
$ sudo tail /var/log/squid3/access.logging
```

You should see your IP address, the URL accessed, page size, etc. You'll also see a separate request line for any image included by the page.

Access control by IP address

Try to configure a different computer to access your proxy server, for example your laptop. What happens?



What caused the *Access Denied* error? We need to find out how Squid access control works. It's defined in the Squid configuration file, `/etc/squid3/squid.conf`, by the following lines:

```
acl localhost src 127.0.0.1/32 ::1
# acl localnet src 10.0.0.0/8    # RFC1918 possible internal network
# http_access allow localnet
http_access allow localhost
http_access deny all
```

What does this mean?

acl localhost src 127.0.0.1/32 ::1

This ACL condition is true if the request's source (i.e. the client's IP address) is either `127.0.0.1` or `::1`.

 # acl localnet src 10.0.0.0/8

This is a commented-out example of an ACL condition called `localnet` (**local network**), which would be true if the client's IP address was in the subnet 10.0.0.0/8.

 # http_access allow localnet

This is a commented-out example of an ACL rule that allows HTTP access (clients connecting to port 3128) to any host where the `localnet` ACL condition is true.

http_access allow localhost

This is a real ACL rule that allows HTTP access to any host where the `localhost` ACL condition is true.

http_access deny all

This ACL rule denies HTTP access to anyone else. It always matches, but rules are applied in order, so the `http_access allow localhost` rule applies first.

Why do you deny me?

Questions:

- What IP address did we try to access the cache from?
- If you don't know, how would you find out?
- Is it allowed or denied by the rules? Which rule in particular?
- How would you change it? What would you have to add?

If you don't know the client's IP address, have a look at the logs.

Reading the logs

Here is an example line from the Squid log file:

1381327552.088 0 **10.0.156.126** TCP_DENIED/403 4425 POST http://safebrowsing.clients.google.com/safebrowsing/downloads? - NONE/- text/html

The IP address is the third field on the line, **10.0.156.126** in this case.

What are the other fields?

1381327552.088

This is the time of the log entry, in Unix timestamp format. Unambiguous, but hard to read. You can convert it on the command line:

```
date --date '@1381327552'
Wed Oct  9 14:05:52 UTC 2013
```

0

Duration, or elapsed time. How long it took to process the request, and return a response, in milliseconds.

10.0.156.126

The IP address of the requesting instance, the client IP address. The client_netmask configuration option can distort the clients for data protection reasons, but it makes analysis more difficult.

TCP_DENIED/403

This column is made up of two entries separated by a slash: the cache result (`TCP_DENIED`) and the HTTP status code returned to the client (`403`).

4425

The length of the response sent to the client, in bytes.

POST

The HTTP *method* requested by the client. Usually this is `GET` to retrieve a web page or image, and `POST` when submitting a form. See the HTTP standard (RFC 2616) for more details.

http://safebrowsing.clients.google.com/safebrowsing/downloads?

The URL requested by the client.

-

The *ident lookup* result. Usually this is useless and turned off.

NONE/-

The *hierarchy code*, which consists of three items: the optional prefix `TIMEOUT`; A code that explains how the request was handled, e.g. by forwarding it to a peer, or going straight to the source; and the IP address or hostname where the request (if a miss) was forwarded to, which might be the origin server, or a neighbor cache.

text/html

The *MIME type* of the response, which usually indicates whether it is a web page, an image, a downloadable executable file, etc. This is sent by the origin server, not determined by Squid, and is not guaranteed to be correct.

Thanks to Amos Jeffries for writing the Squid Wiki LogFormat page where this information was found.

Don’t deny me!

How do we change the access control configuration, to allow connections from a different IP address?

Add the following lines to the Squid configuration file:

```
acl localnet1 src 10.0.156.0/24
http_access allow localnet1
```

Note that:

- The ACL name must be unique. It should also be descriptive. Don't call all your local networks `localnet` or `localnet1`.
- These lines **must** appear before `http_access deny all`. (Why?)
- It's probably safest, and easier to read the configuration file, if you keep all of your own ACL configuration lines between `http_access allow localhost` and `http_access deny all`.

Reloading and restarting Squid

What happens when you change the configuration? Does it automatically take effect?

No. Squid doesn't reload its configuration file automatically. You need to restart it:

```
$ sudo restart squid3
```

Or tell it to reload its configuration:

```
$ sudo /etc/init.d/squid3 reload
or
$ sudo squid3 -k reconfigure
```

Restarting is slow, because it waits for open connections to finish. No requests are serviced during this time, so web access is impossible. The `reload` and `reconfigure` commands (which do the same thing) don't cause any downtime for the service, and don't clear the in-memory caches (`cache_mem` and the DNS cache), so they are usually a better choice.

However, if you enable `cache_dir` then Squid needs to shut down and restart in order to initialize it. Just a `reload` isn't enough, and it won't cache anything on disk until you `restart` it.

Reverse proxies and open proxies

Why not just allow everyone? Like this:

```
acl everyone src 0.0.0.0/24
http_access allow everyone
http_access allow all
```

Because this would create an open proxy, which is bad because:

- People outside the organisation can waste your bandwidth.
- They can also conduct illegal activities using your proxy, and the police will come knocking on your door instead of theirs.
- Spammers often use open proxies to send spam.
- As a result, some realtime blacklists (RBLs) scan for open proxies and when they find one, they add its IP address to their blacklist.

So every proxy should do one of the following:

Forward proxy

Restrict access to certain source IP addresses

Reverse proxy

Restrict access to certain destination domains (with the `acl dstdomain`).

Further configuration of reverse proxies is out of scope of this tutorial, but you can find more details on the Squid Cache wiki.

Cache Size

The cache size determines the hit rate (bandwidth and time saving) of the Squid proxy server, trading off against disk space and memory usage.

Making the caches too large for the system can result in complete failure of the proxy server, starvation of resources from other applications on the same server, and eventually swap death of the server.

Disk cache size

The default configuration on Ubuntu contains the following:

```
#cache_dir ufs /var/spool/squid3 100 16 256
```

How big is the default disk cache size? Do we want to change it?

There is NO uncommented `cache_dir` by default, so there is **no disk cache**. There is however a memory cache of 256 MB:

```
cache_mem 256 MB
```

(This is the default unless an uncommented `cache_mem` line is found in the file, which there isn't in the default Ubuntu configuration.)

The `100` in the above configuration means that the cache would be 100 MB, if it was enabled. A more useful cache size would be 10-100 GB, so you could uncomment this line and change it to:

```
cache_dir ufs /var/spool/squid3 10000 16 256
```

Note that this will place the cache in the directory `/var/spool/squid3`. This filesystem must not fill up, otherwise the cache will stop working, and nobody will be able to browse the web! Make sure that you don't allow the cache to grow larger than the free space on the filesystem, which you can tell with the `df` command.

Also, leave enough space for anything else using the same filesystem, so that it doesn't fill up. Log files, mailboxes and SQL databases usually live under `/var`, and if you don't have a separate filesystem for them, `/home` and `/tmp` will also take space away from the Squid cache.

Memory usage

Memory that will be used by Squid:

- about 10 MB of RAM per GB of cache specified by your `cache_dir` directive;
- plus the amount specified by the `cache_mem` directive;
- plus another 20 MB for additional overhead.

You need to ensure that there's enough memory left for the OS and its block cache.

For example, if you set `cache_dir` to 10000 (10 GB) and leave `cache_mem` set to the default 256 MB, then Squid will use approximately 100 + 256 + 20 = 376 MB.

If this is more than half the RAM in your cache server, then reduce either the `cache_dir` or `cache_mem`, or add more memory to the cache server.

Squid Access Control

Access control determines which requests are allowed or denied by the Squid proxy server. It also determines which requests are routed into which delay pools (bandwidth limits).

Access control elements

Every line in the configuration file that starts with `acl` is an Access Control Element (ACE). These are reusable sets of conditions:

- You can use them in as many rules as you like,
- and combine them with each other in rules.

Every ACE must be *defined*, which gives it a unique *name*. The definition looks like this:

```
acl <name> <type> <values>
```

For example, the ACL we created earlier:

```
acl localnet1 src 10.0.156.0/24
```

Has the name **localnet1**, type **src** (source IP address) and value (which is specific to the *src* type) **10.0.156.0/24**.

ACE types

The *type* determines what kinds of *values* are appropriate:

ACL type	Values	Example
src	source (client) IP addresses or CIDR ranges	10.0.156.1, 10.0.156.0/24, 2001::dead:beef
dst	destination (server) IP addresses or CIDR ranges	10.0.156.1, 10.0.156.0/24, 2001::dead:beef
dstdomain	destination (server) domain name, exact/prefix	`www.facebook.com`, `.facebook.com`
dstdom_regex	destination (server) regular expression pattern	`.facebook..*`
maxconn <N>	client IP address has more than N TCP connections	`10`
proto	the protocol part of the requested URL	HTTP, FTP
time	days (SMTWHFA) and time range (h1:m1-h2:m2)	`19:00-23:59`, `MTWHF 08:00-18:00`
url_regex	regular expression match on requested URL	`sex`, `iso`, `mp3`
browser [-i]	pattern match on User-Agent header	`-i MSIE 6.1`

The `srcdomain` ACE: a special case

If you block `.microsoft.com`, does it block `microsoft.com` as well as `www.microsoft.com`? Why?

Answer: Yes it does, because of a specific exception in the Squid source code. Many websites are accessible with and without the `www` subdomain, by convention, and it would be annoying to have to specify every domain twice, with and without the initial dot `.`, to match both of them.

ACEs with multiple values

The values are combined using `OR` logic. If any value matches, the whole ACE matches. So it's valid to include mutually exclusive values on the same ACE:

```
acl mynetworks src 192.168.1.0/24 192.168.3.0/24
acl updates dstdomain .microsoft.com .adobe.com
```

What happens if you specify overlapping domains? For example:

```
acl updates dstdomain .microsoft.com .download.microsoft.com
```

The Squid FAQ says:

> You can't have one entry that is a subdomain of another. Squid will warn you if it detects this condition.

Access control rules

Rules look like this:

```
http_access         allow <ace name> <ace name>
http_access         deny  <ace name> <ace name>
delay_access <pool> allow <ace name> <ace-name>
```

There are several different types of rules, all ending with `_access`:

http_access
: Control whether a client is allowed to make a particular request through the HTTP port (3128)

icp_access
: Control whether a cache peer is allowed to make a particular request through the ICP port. Could this be abused, and how?

cache_peer_access
: Control which requests will be sent to a particular cache peer. This type of rule needs a parameter; why?

delay_access
: Control which requests will be sent to a particular delay pool. This type of rule needs a parameter; why?

snmp_access
: Control access to the built-in SNMP server (need to recompile Squid on Debian and Ubuntu to use this).

Rules with multiple ACEs

The ACEs on an access control rule are combined using `AND` logic. All the ACEs must be true, otherwise the rule will be ignored for that request.

Rules are processed in order, and the first matching rule (where all the ACEs are true) of a particular type determines what happens for that rule type.

Examples:

- The first matching `http_access` rule determines whether an HTTP request is allowed or denied.
- The first matching `cache_peer_access` rule determines whether the request is sent to a peer cache, and which one.
- The first matching `delay_access` rule determines whether the request is sent to a delay pool, and which one.

Rule processing examples

Which hosts and domains are allowed, which are denied, and which are sent to a peer cache in the following configuration?

> acl microsoft dstdomain .microsoft.com acl wireless src 10.0.158.0/24 http_access allow all http_access deny wireless cache_peer_access updates allow microsoft cache_peer_access updates deny all

Access control practice

Try blocking the following, and get someone else to check your work:

- a particular client IP address
- the subnet that your client is on
- a subnet that your client is NOT on
- www.facebook.com
    - except for one client IP address
    - and try to evade the ban
    - did you just block `http://www.bing.com/search?q=facebook` as well?
    - how would you do that?
- any website with `sex` in the URL
- did you just block `http://www.essex.ac.uk/`?
- more than 2 connections per client IP address (how would you test it?)
- FTP downloads from `ftp://www.mirrorservice.org/`

Remember to follow a good, thorough process for each exercise:

- decide beforehand how you will test for success;
- check that your request is not already blocked;
- make the change to implement the block;
- check that it behaves as you expected;
- undo the change before moving on to the next;
- check that the request is allowed again.

Otherwise you might think that you succeeded, when actually the request was blocked by some previous configuration that you didn't undo successfully.

Be careful if you test using a site that automatically redirects you to SSL, such as `www.google.com` or `www.duckduckgo.com`, as this will bypass the cache without you realising! You can test with `www.bing.com` as it doesn't do that at the time of writing (2013-10-09).

Solutions

Block a particular client IP address:

```
acl bad_boy src 10.0.156.126
http_access deny bad_boy
```

Block the subnet that your client is on:

```
acl bad_boys src 10.0.156.0/24
http_access deny bad_boys
```

Block a subnet that your client is NOT on:

```
acl bad_boys src 10.0.157.0/24
http_access deny bad_boys
```

Block `www.facebook.com`:

```
acl facebook dstdomain www.facebook.com
http_access deny facebook
```

Allow Facebook only for a single client IP address:

```
acl facebook dstdomain www.facebook.com
acl good_boy src 10.0.156.126
http_access allow good_boy
http_access deny facebook
```

Try to evade the ban:

- go to `http://m.facebook.com` instead
- go to `https://www.facebook.com` instead

Did you just block `http://www.bing.com/search?q=facebook` as well?

Block any website with `sex` in the URL:

```
acl sex url_regex sex
http_access deny sex
```

Prevent more than 2 connections per client IP address:

```
acl too_many_connections maxconn 2
http_access deny too_many_connections
```

Testing that it worked:

- `ab -X localhost:3128 -n 10 -c 2 http://www.mirrorservice.org/` (2 concurrent requests) should show no errors: `Non-2xx responses: 0`
- `ab -X localhost:3128 -n 10 -c 3 http://www.mirrorservice.org/` (3 concurrent requests) should show some errors, e.g. `Non-2xx responses: 8`

Block all FTP downloads:

```
acl ftp proto ftp
http_access deny ftp
```

Note: you will need to configure your browser to use the proxy for FTP as well as HTTP requests.

Web Proxies and SSL

Web proxies can't intercept SSL connections, because:

- they would have to sign the response pages (to be SSL compliant)
- and nobody except Facebook has the keys to sign responses as www.facebook.com (we hope!)
- so the proxy could not create a valid signature
- and the browser would complain about an invalid signature
- this is exactly what SSL security is supposed to do!

What can we do about it?

- Put a fake Certificate Authority (CA) in all the browsers and have the proxy sign responses with that certificate (hard to reach all devices and browsers!)
- Or use browser support for the CONNECT method.

HTTP and CONNECT requests

An HTTP request looks like:

```
> GET http://www.google.com/ HTTP/1.0
> Headers...

< Response...
```

A CONNECT request looks like this:

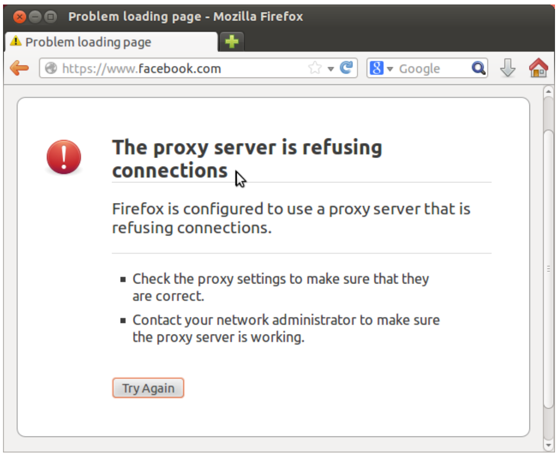
```
> CONNECT www.google.com:80
> Encrypted traffic
< Encrypted traffic
```

With CONNECT, the proxy only sees the hostname connected to, not the page requested or any other details about the connection. We can filter on hostname, and that's about it. For example, if the browser is configured to use our proxy for all requests, then this ACL blocks Facebook SSL as well:

```
acl facebook dstdomain .facebook.com
http_access deny facebook
```

Results of blocking SSL requests

What happens in the browser?



This is a lie! The proxy didn't refuse the connection at all. It did however refuse to service the request. It returned an error page, but Firefox won't display it for you because it's not encrypted.

How can you tell? Look at the logs:

> 1381400327.288 0 10.0.156.121 TCP_DENIED/403 3631 CONNECT www.facebook.com:443 - NONE/- text/html

This is just a limitation of SSL filtering that we have to live with.

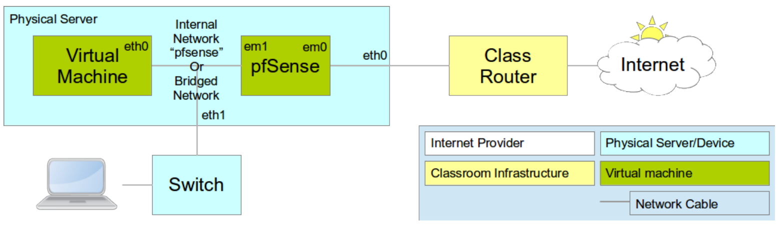
Forcing people to use the proxy

People can just disable their proxy configuration to work around blocks. What can you do about it?

First, we need to block direct access to HTTP and HTTPS ports (80 and 443) for all clients **except the proxy server**.

Configure pfSense as your router

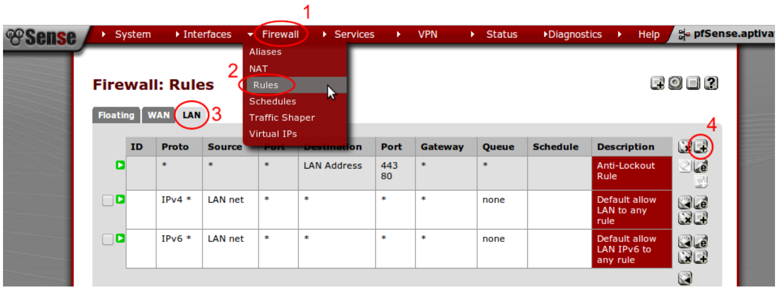
To do these exercises using pfSense, configure your virtual network as follows:



In other words:

- The **external** interface of the pfSense virtual machine (*Network Adapter 1*) is Bridged with the external interface of your server (probably *eth0*).
- If your server has two network interfaces, then the **internal** interface of the pfSense virtual machine (*Network Adapter 2*) is Bridged with the internal interface of your server (probably *eth1*), and so is the only network interface (*Network Adapter 1*) of your client Virtual Machine. This allows you to connect laptops to *eth1* and use them to test your connection, as well as the client Virtual Machine.
- If your server has only one network interface, then the **internal** interface of the pfSense virtual machine (*Network Adapter 2*) is connected to the *Internal Network pfsense*, and so is the only network interface (*Network Adapter 1*) of your client Virtual Machine. This only allows you to test your connection from the client Virtual Machine.

Then configure pfSense to block ports 80 and 443 outbound from LAN:



- Open the pfSense webConfigurator and log in
    - This is probably at http://192.168.1.1/ from your laptop or VM, connected to the internal interface *em1* of the pfSense VM, unless you've reconfigured pfSense to change the LAN subnet.
- From the menu choose Firewall/Rules
- Click on the LAN tab
- Click on the pfSense "add rule" button
- Add a rule to **reject** TCP traffic on the LAN interface to destination port HTTP (80).
- Add another rule before this one, to **pass** TCP traffic on the LAN interface to destination port 80 **from the proxy server VM** (Under *Source*, choose *Single host or alias*, and enter the IP address of the proxy server VM)
- Repeat the same rules for HTTPS (port 443).

Your rules should now look like this:

Floating | WAN | LAN

ID	Proto	Source	Port	Destination	Port	Gateway	Queue	Schedule	Description
	*	*	*	LAN Address	443 80	*	*		Anti-Lockout Rule
	IPv4 TCP	192.168.1.100	*	*	80 (HTTP)	*	none		Allow direct HTTP from proxy server
	IPv4 TCP	*	*	*	80 (HTTP)	*	none		Block direct HTTP
	IPv4 TCP	192.168.1.100	*	*	443 (HTTPS)	*	none		Allow direct HTTPS from proxy server
	IPv4 TCP	*	*	*	443 (HTTPS)	*	none		Block direct HTTPS
	IPv4 *	LAN net	*	*	*	*	none		Default allow LAN to any rule
	IPv6 *	LAN net	*	*	*	*	none		Default allow LAN IPv6 to any rule

Apply these rules in pfSense. Check that you can access websites from the proxy server VM, and not from other clients. Other traffic such as *ping* should still work from all clients.

Proxy auto configuration

This is how Web Proxy Auto Detection works:

- The DHCP server gives clients a special option (number 252) which includes the URL of a WPAD server.
- If it doesn't, then clients will use the URL `http://wpad.<domainname>/wpad.dat`.
- The client will try to download this file (a Proxy Auto Configuration or PAC file) and execute it as JavaScript.
- The JavaScript can examine each requested URL, and must return the details of which proxy server to use for that URL.

Creating a PAC file

You need a web server to host the file for you. If you already installed Apache on the Ubuntu virtual machine (the proxy server VM) during the Linux Familiarization session, then you don't need to do anything. Otherwise, install Apache on the proxy server VM:

```
$ sudo apt-get install apache2
```

Use an editor to create the file `/var/www/wpad.dat`, for example:

```
$ sudo vi /var/www/wpad.dat
```

And add the following contents:

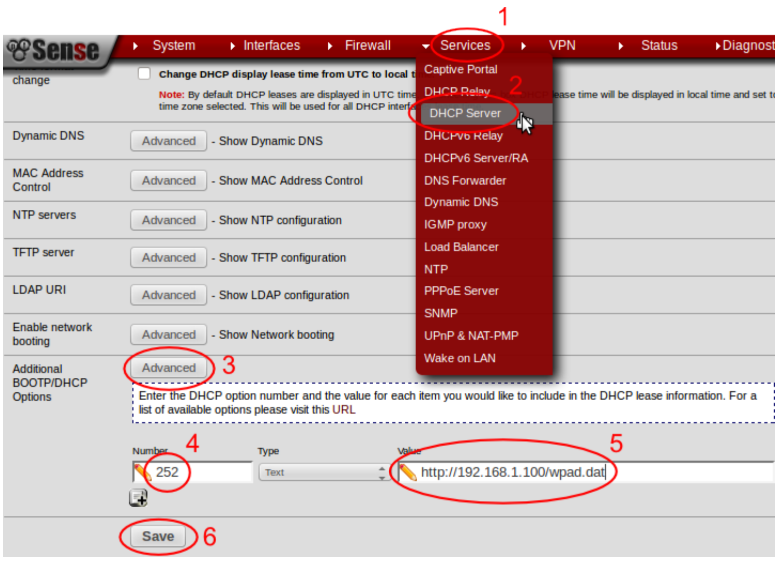
```
function FindProxyForURL(url, host)
{
        return "PROXY 192.168.1.1:3128";
}
```

Now you should be able to retrieve the file using a client's web browser, by visiting the URL `http://192.168.1.100/wpad.dat`. Otherwise, please check:

- the IP address of the proxy server (which may not be 192.168.1.100);
- that the Apache web server is running on it;
- the permissions on the `wpad.dat` file should be world readable.

DHCP server settings in pfSense

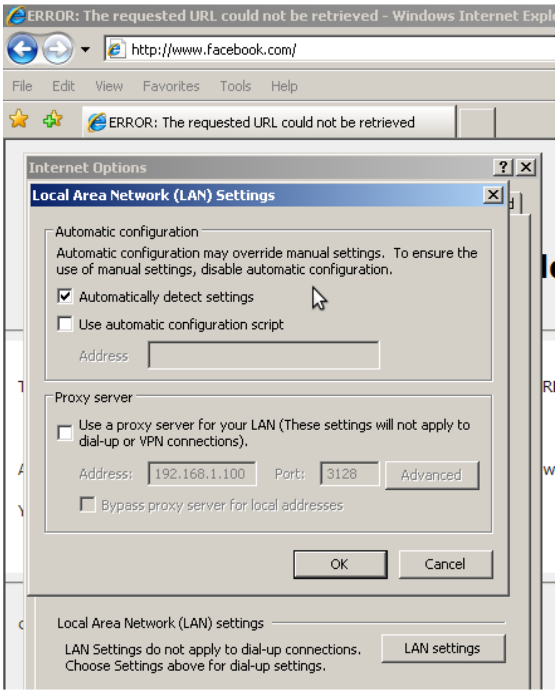
Now reconfigure the pfSense firewall to hand out the URL of the `wpad.dat` file to all DHCP clients:



- Open the pfSense webConfigurator and log in.
- From the menu choose "Services/DHCP Server".
- Scroll down to *Additional BOOTP/DHCP options* and click on the *Advanced* button.
- For *Number* enter 252, and for `Value` enter the URL of the `wpad.dat` file.
- Click the *Save* button.

Testing Proxy Auto Configuration

To test this, you may need to force your clients to renew their DHCP leases, and enable proxy autodetection. In Internet Explorer this is under Tools/Internet Options, Connections, LAN Settings, Automatically Detect Settings:



Proxy Authentication

The aim of proxy authentication is to:

- Ensure that unauthorised clients don't use your proxy servers (to carry out illegal activity on your behalf, or waste your bandwidth); and
- Ensure that each request is accountable to a particular user.

About RADIUS

What is RADIUS?

- Remote Authentication Dial-In User Service.
- Provides authentication: checking usernames and passwords against a database.
- Provides authorization: details about which services a user is allowed to access.
- Commonly used by network switches and access points to authenticate users for the 802.1x protocol.
- RADIUS service can be linked to an Active Directory server.

For more details on RADIUS, see this presentation or the Wikipedia page.

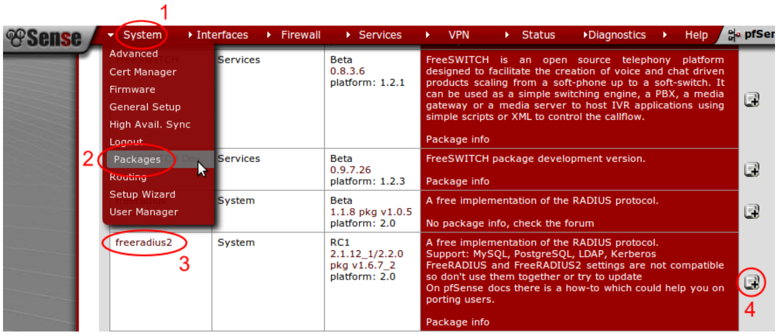
Setting up a RADIUS Server

RADIUS is a client-server protocol, so we need a server. It's easy to install and manage the FreeRADIUS software on pfSense, so we'll use that.

More detailed instructions on installing and using FreeRADIUS on pfSense can be found in the pfSense Documentation.

Installing FreeRADIUS on pfSense

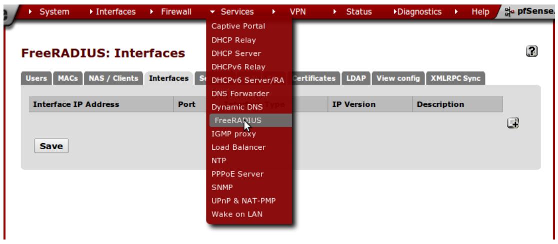
To quickly install a RADIUS server (FreeRADIUS):



- Open the pfSense webConfigurator and log in.
- From the menu choose *System/Packages*.
- Scroll down to *freeradius2*.
- Click on the + icon to right of the package details.

Configuring FreeRADIUS

Having installed FreeRADIUS, we have to configure it.



- In the pfSense webConfigurator menu, choose *Services/FreeRADIUS*.
- Click on the *Interfaces* tab, and click on the *Add a new item* icon on the right.
- Leave all the setting unchanged, and click on the *Save* button.
- Now click on the *NAS/Clients* tab, and click on the *Add a new item* icon on the right.
- For *Client IP Address* enter the IP address of the Squid server (which might be 192.168.1.100).
- For the *Client Shortname* enter `squid`.
- For the *Client Shared Secret* enter a long random password, that will also be entered on the Squid server. For testing purposes, set it to `testing123`. Please be sure to change this password if you move to production!
- For *Description* enter `Squid Proxy Server`.

Adding Users

- In the pfSense webConfigurator menu, choose *Services/FreeRADIUS*.
- Click on the *Users* tab, and click on the *Add a new item* icon on the right.
- Enter a *Username* and *Password* for the new user. Clients will have to log in as one of these users, to use the proxy server. For testing purposes, you can create a user called `john` with password `smith`. Please be sure to delete this user if you move to production!
- Leave the other settings unchanged and click on the *Save* button.

Testing RADIUS Authentication

On the Squid proxy server, install the `radtest` application:

```
$ sudo apt-get install freeradius-utils
```

And run a test against the server:

```
$ radtest john smith 192.168.1.1 1812 testing123
```

You should see an `Access-Accept` response if everything is OK:

```
Sending Access-Request of id 92 to 192.168.1.1 port 1812
        User-Name = "john"
        User-Password = "smith"
        NAS-IP-Address = 127.0.1.1
        NAS-Port = 1812
rad_recv: Access-Accept packet from host 192.168.1.1 port 1812, id=92, length=20
```

Otherwise please check:

- the IP address and shared secret for the server on the `radtest` command line;
- the username and password that you used, which must match a FreeRADIUS user on the pfSense firewall;
- the IP address of the Squid server and the shared secret, in the FreeRADIUS configuration of the pfSense firewall.

Can you access the RADIUS server from any other computer? Why, or why not? What's the benefit of this configuration?

Squid RADIUS Authentication

You need to configured the Squid proxy server with the details of the RADIUS server to connect to.

On the Squid server, create the file `/etc/squid3/radius_config` with the editor of your choice, for example:

```
$ sudo vi /etc/squid3/radius_config
```

Place the IP address of the RADIUS server (the pfSense firewall's LAN address) and the shared secret in this file. For example:

```
server 192.168.1.1
secret testing123
```

Test it by running `squid_radius_auth` on the command line:

```
$ /usr/lib/squid3/squid_radius_auth -f /etc/squid3/radius_config
```

Enter a RADIUS username and password, separated by a space, for example:

```
john smith
```

You should see the output `OK`. Press Ctrl+C to stop the authenticator process.

Now edit your Squid configuration and add the following lines, to require all Squid users to authenticate themselves, just before the existing line `http_access deny all` (which you don't need to duplicate):

```
auth_param basic program /usr/lib/squid3/squid_radius_auth -f /etc/squid3/radius_config
auth_param basic children 5
auth_param basic realm Web Proxy
auth_param basic credentialsttl 5 minute
auth_param basic casesensitive off

acl radius-auth proxy_auth REQUIRED
http_access allow radius-auth
http_access deny all
```

Remember to remove or comment out any `http_access allow` lines that give access to all users without authentication. Tell Squid to reload its configuration and test it.

Squid tends to kill itself if it has problems accessing an authenticator. So if it's not working, and you can't access any web pages, check that Squid is still running:

```
$ status squid3
```

If not (if it says `stop/waiting`) then check the cache log file to find out why it died:

```
$ sudo tail -30 /var/log/squid3/cache.log
```

For example, it might say this:

```
FATAL: auth_param basic program /usr/local/squid/libexec/squid_radius_auth: (2) No such file or directory
Squid Cache (Version 3.1.19): Terminated abnormally.
```

Which means that the path to the `squid_radius_auth` program is wrong in the Squid configuration file.

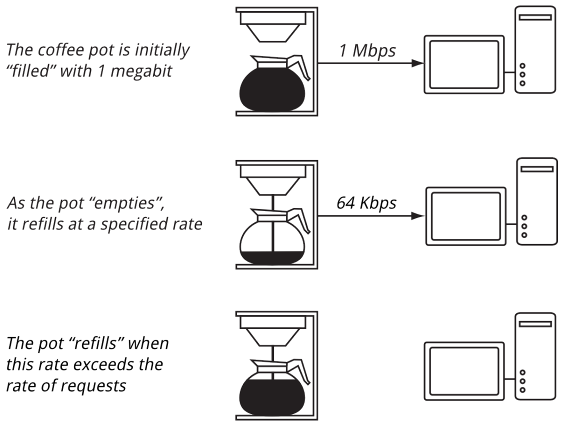
Squid Delay Pools

Squid has a feature called *delay pools* that can throttle users' bandwidth usage for web downloads to a certain amount.

Each pool behaves like a coffee pot:

- People remove large chunks of bandwidth (coffee) when they make a request.
- Requests are satisfied immediately while the pool is not empty (while coffee remains in the pot).
- When the pool (coffee pot) is empty, all requests must wait for it to refill.
- The pool refills at a fixed rate.

Technically this is known as a Token Bucket Filter (TBF).



Classes of delay pools

You can have any number of pools. You can configure each pool's type (class) to one of the five built-in classes:

class 1
: a single unified bucket which is used for all requests from hosts subject to the pool.

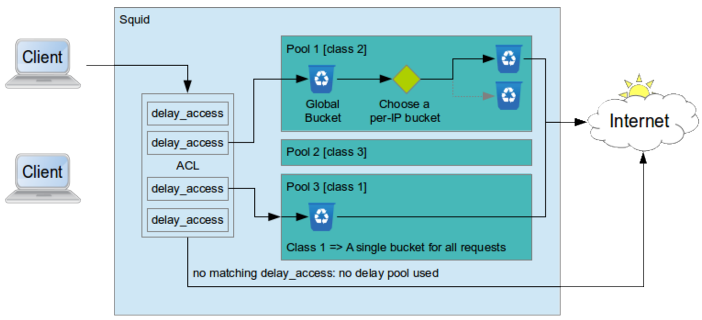
class 2
: one unified bucket and 255 buckets, one for each host on an 8-bit network (IPv4 class C).

class 3
: contains 255 buckets for the subnets in a 16-bit network, and individual buckets for every host on these networks (IPv4 class B).

class 4
: as class 3 but in addition have per authenticated user buckets, one per user.

class 5
: custom class based on tag values returned by external_acl_type helpers in http_access. One bucket per used tag value.

Request routing



The `delay_access` rules determine which pool is used for each request.

The type (class) of the pool, and the current state of its buckets, determine how much bandwidth is available for that request.

Limitations of pools

Each pool is completely independent of all other pools.

Each pool contains one or more buckets, which are completely independent of all other buckets.

The allocation of requests to buckets within a pool determines who shares bandwidth within the pool:

class 1 pool
: All users share the same bucket, and so they share bandwidth with each other.

class 2 pool
: All users share a bucket, but each has their own bucket (one per IP address) as well.

class 3 pool
: All users share a global bucket, and one bucket with their subnet. So all 192.168.1.x users share a bucket, and all 192.168.2.x share a different bucket.

class 4 pool
: In addition to class 3, each authenticated user gets their own bucket as well.

class 5 pool
: Only works if you use an `external_acl_type` ACL to assign a tag to each request. Each unique tag value gets its own bucket. You can use this to assign users to buckets in any custom scheme that you like.

Simple example

To have all users share a single pool with 256 kbps bandwidth, add the following to your Squid configuration:

```
delay_pools 1
delay_class 1 1
delay_parameters 1 32000/64000
delay_access 1 allow all
```

How can we test it? Using wget:

```
$ export http_proxy=http://john:smith@localhost:3128
wget http://www.mirrorservice.org/sites/mirror.centos.org/6/isos/x86_64/CentOS-6.4-x86_64-bin-DVD1.iso
```

Questions:

- What does this Squid configuration do?
- What speed do we expect to see?
- What happens at the beginning of the download?
- What happens if you run two downloads at the same time?

Answers:

delay_pools 1
: There is only one pool: number 1.

delay_class 1 1
: Pool 1 is a class-1 pool.

delay_parameters 1 32000/32000
: Pool 1 refills at 32 kilobytes per second, up to a maximum level of 64000 bytes.

delay_access 1 allow all

All requests are routed into pool 1.

We should see an initial high speed burst for 1-2 seconds, and then the download should slow down to 32 kiloBytes per second (kBps).

If more users download at the same time, they will share bandwidth equally between them (kBps each).

More advanced configuration

How would you give each authenticated user 512 kbps, and limit all users to 4 Mbps at the same time?

What class of delay pool do you want to use?

Hint: the delay_parameters line for this class has the following format:

```
delay_parameters <pool> <aggregate> <network> <individual> <user>
```

And you can use `-1/-1` as the value to have unlimited capacity in a certain set of buckets.

Answer:

```
delay_pools 1
delay_class 1 4
delay_parameters 1 64000/64000 -1/-1 -1/-1 512000/512000
delay_access 1 allow all
```

FIN

Any questions? Bandwidth Management with pfSense ——————

Objectives

On completion of this session, we hope you will be able to:

- Configure traffic queues in pfSense
- Classify traffic into queues
- Monitor and debug bandwidth management

If you are the facilitator, please tell the group:

> At the end of session I will ask if we have met the objectives – if not, we will discuss again.

License

Some materials reused under the Creative Commons Attribution-NonCommercial-ShareAlike 2.5 license:

- the Web Caching manual, by Richard Stubbs of TENET;
- the BMO Book, by various authors;
- the Squid Cache Wiki, by Amos Jeffries and other.

Introduction

What is bandwidth management?

Similar to traffic management on roads:

- Give some vehicles priority over others (e.g. emergency services)
- Keep one lane clear for priority vehicles
- Limit the number and length of car journeys
- Efficiency savings: reduce the need for car journeys (public transport, local markets and supermarkets)
- Make better use of unused capacity: encourage spreading of load into off-peak periods
- Increase the cost of petrol, or charge tolls
- Arrest people for driving slowly

Also called *traffic shaping* (which is reasonable) and *packet shaping* (which is not. What shape are your packets?)

Wired Magazine's take

> Most ISPs actively engage in traffic shaping, bandwidth throttling, connection denial or some such tactic to keep the amount of bandwidth consumed by high traffic applications on their networks to a minimum. While this does often ensure better performance for everyone in the neighborhood, it can mean painfully slow transfer speeds for [peer to peer file sharing applications.]
>
> While there are valid arguments for and against shaping, we're not here to debate. We just want the fastest BitTorrent transfers possible.

http://howto.wired.com/wiki/Optimize_BitTorrent_To_Outwit_Traffic_Shaping_ISPs

What are the limitations?

From Ginsberg's theorem (Laws of Thermodynamics):

You can't win.

Bandwidth management will not make your connection faster. It's just benefiting "more desirable" traffic at the expense of "less desirable." Everyone's traffic is desirable to them, so some people will always be upset and you need a strong policy argument to defend yourself with.

You can't break even.

Bandwidth management is not free. You have to reduce your total bandwidth in order to own and control the queues. And you have to invest a lot of effort into developing, understanding and maintaining your policy.

You can't even get out of the game.

Parkinson's law says that *Work expands so as to fill the time available for its completion.* The same applies to traffic and capacity. Unless you manage traffic, you will have chaos.

Some other limitations:

Traffic doesn't declare its type or priority.

In fact, users can try to hide their traffic, for example in a VPN or with encryption to evade restrictions on P2P or take advantage of higher service classes.

Bandwidth management is hard to do.

Bandwidth management applies to TCP/IP packets and network interfaces, and debugging tools are very limited (you can't even see what packets are in which queue), so you need a deep understanding of what's going on, end to end across the Internet. You often need to spend significant time monitoring and investigating traffic patterns, or create and test a theory, to understand and solve a problem.

For example you might have to differentiate between:

- Skype voice traffic (UDP encapsulated, encrypted, random ports, might be direct or via a gateway in Czechoslovakia or a Microsoft IP address)

- VPN traffic to a BitTorrent anonymiser in the Netherlands? (OpenVPN to UDP port 1194).
- Your own VPNs (OpenVPN to UDP port 1194).

Or how about distinguishing between large HTTP/SSL downloads and ordinary web page browsing?

You may have to write rules, exceptions and exceptions to exceptions, one per class of traffic, to stay on top of assigning traffic to the correct classes.

Sometimes it's easier to create a whitelist: for example you can filter traffic from known good sites (academic publishers, etc.) into classes with high priority access to bandwidth/capacity.

What can we do with pfSense?

- Keep one lane clear (reserve bandwidth)
- Limit the number and length of car journeys (restrict bandwidth)
- Efficiency savings (block some kinds of traffic)

Why use pfSense?

Linux also has a traffic management framework. Why do we use pfSense instead of Linux?

- Advantages: * Nice point and click interface * Graphical display of bandwidth used by each class * Slightly easier to use
- Disadvantages:
- Limited features: no SFQ? no per-connection byte counters?

For more information you can read:

- OpenBSD PF: Packet Queueing and Prioritization (pfSense uses the same `pf` packet filter as OpenBSD).
- Managing Traffic with ALTQ (pfSense is based on ALTQ).
- Hierarchical Fair Service Curve
- ALTQ/CBQ Performance (CPU overhead of scheduling packets)
- How Unfair can Weighted Fair Queuing be? (limitations of work-conserving qdiscs).

How do we start?

Limit the maximum bandwidth in and out of firewall.

- Advantage: allows us to control the queues.
- Disadvantages: * Requires that we know how much bandwidth is available; * Reduces the available bandwidth; * Limits are per-interface, so interface load balancing doesn't work.

Why is this a problem? Because we often don't know exactly how much bandwidth is available to us. Contention at the ISP may result in us having less bandwidth than expected at peak times. In order to control the queue, we have to limit bandwidth to the worst case that we expect, or live with imperfect control.

The ISP may impose their own bandwidth management on our traffic, which is outside our control.

Kilobits and kilobytes

Questions:

- What does kbps mean?
- What does kBps mean?
- Convert 128 kbps to kBps
- Convert 128 kBps to kbps

- Why do we use different units?

Answers:

- kbps is *kilobits per second* (little b = bits, because bits are smaller), also called Kbit/s or Kb/s.
- kBps is *kilobytes per second* (big B = bytes, because Bytes are Bigger), also called Kbyte/s or KB/s.
- 128 kbps / 8 = 16 kBps
- 128 kBps * 8 = 1024 kbps
- We use different units because interfaces transmit one bit at a time, so their capacity is measures in bits per second; but computers work with whole bytes, so bytes per second is a more logical measure?

Example configuration

Limit total bandwidth to:

- 1024 kbps download
- 256 kbps upload

Decide how much bandwidth we want to allocate, and to what. For example:

- Upload: * 50% reserved for Voice over IP (VoIP). * 30% reserved for HTTP, plus borrowing from remaining traffic (70%). * 20% remaining for all other traffic.
- Download: * 12.5% reserved for Voice over IP (VoIP). * 70% reserved for HTTP, plus borrowing from remaining traffic (70%). * 17.5% remaining for all other traffic.

Question? How much bandwidth (kbps) is reserved for each class?

Why different policies for upload and download?

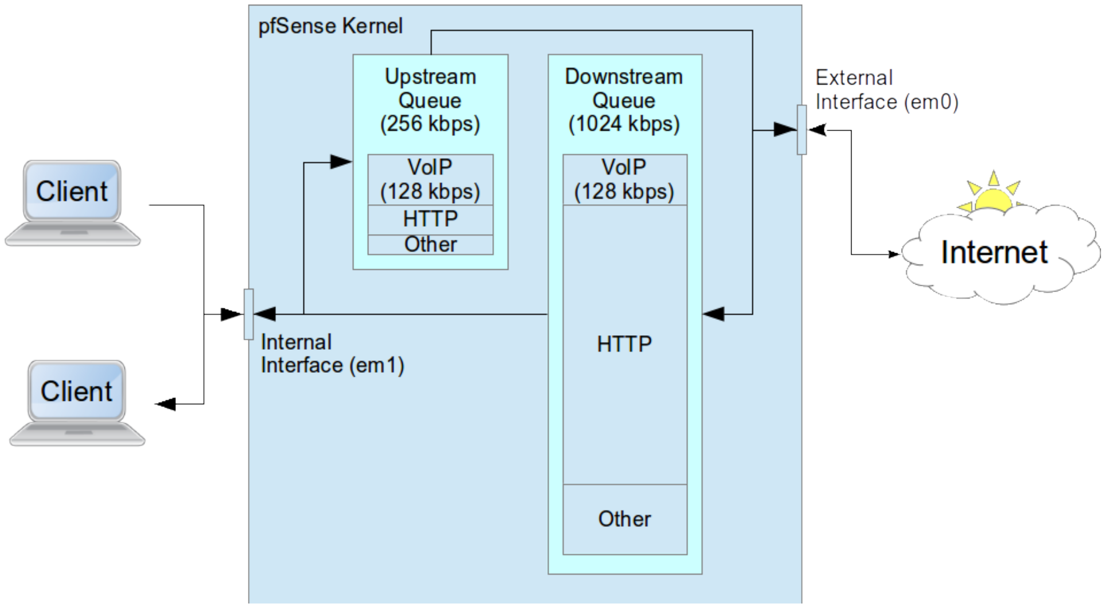
Answers:

- Upload: * VoIP: 50% x 256 kbps = 128 kbps * HTTP: 30% x 256 kbps = 76.8 kbps * Other: 20% x 256 kbps = 51.2 kbps
- Download: * 12.5% x 1024 kbps = 128 kbps * 70% x 1024 kbps = 716.8 kbps * 17.5% x 1024 kbps = 179.2 kbps

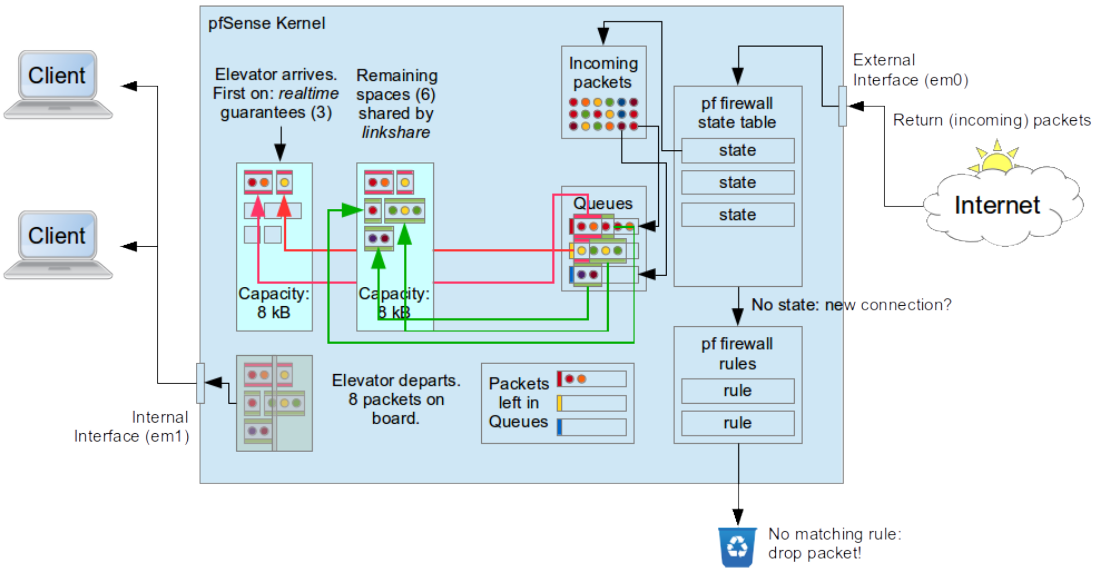
VoIP tends to be symmetrical. We'd like to allocate 128 kbps in both directions, which is a much bigger share of our upload bandwidth. (Welcome to asymmetric connections.)

HTTP tends to be highly asymmetric, and we want it to be fast, so we allocate the most download bandwidth to it.

Bandwidth Allocation



How HFSC Works



Imagine that an elevator arrives every second, with space to carry some packets to the other side.

Packets are assigned to queues by firewall rules. In pfSense, the queue has the **opposite** direction to the firewall rule. So a rule that allows **incoming** packets places the **replies** into the specified queue. That's normally what you want, because in most client-server protocols, the replies from the server are much bigger than the request.

So you have a rule that allows HTTP connections **outwards**. The replies are placed in a queue of your choice, coming back **inwards**.

pfSense uses the FreeBSD ALTQ framework, with a choice of schedulers:

Priority Queueing (PRIQ)

> Takes packets from the highest-priority queue first, then the second, and so on until it reaches the bandwidth limit assigned to the interface. It's easy for high-priority traffic

to take all the bandwidth, leaving low-priority traffic with none. Not recommended.

Class Based Queueing (CBQ)

Divides a network connection's bandwidth among multiple queues or classes. Queues are arranged in an hierarchy. Child queues are created under the root queue, each of which can be assigned some portion of the root queue's bandwidth. Queues are served in strict priority order. If any bandwidth remains, it can be borrowed by other queues to ensure that no bandwidth is wasted.

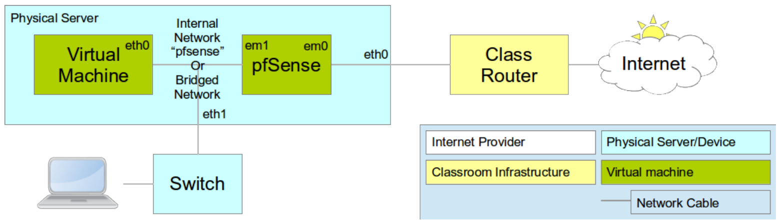
Hierarchical Fair Service Curve (HFSC)

Similar to CBQ, but adds real-time guarantees (bounded delay). This allows packets to skip the queue if their delay exceeds a fixed amount.

We will use HFSC for this exercise.

Configure pfSense as your router

To do these exercises using pfSense, configure your virtual network as follows:

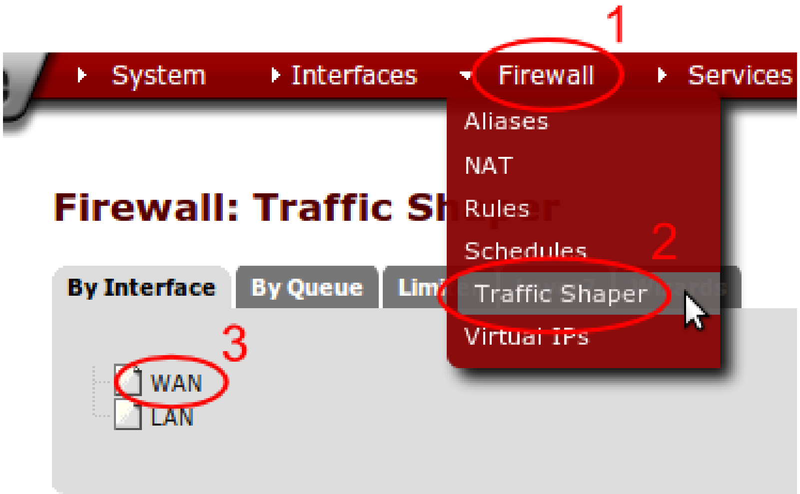


In other words:

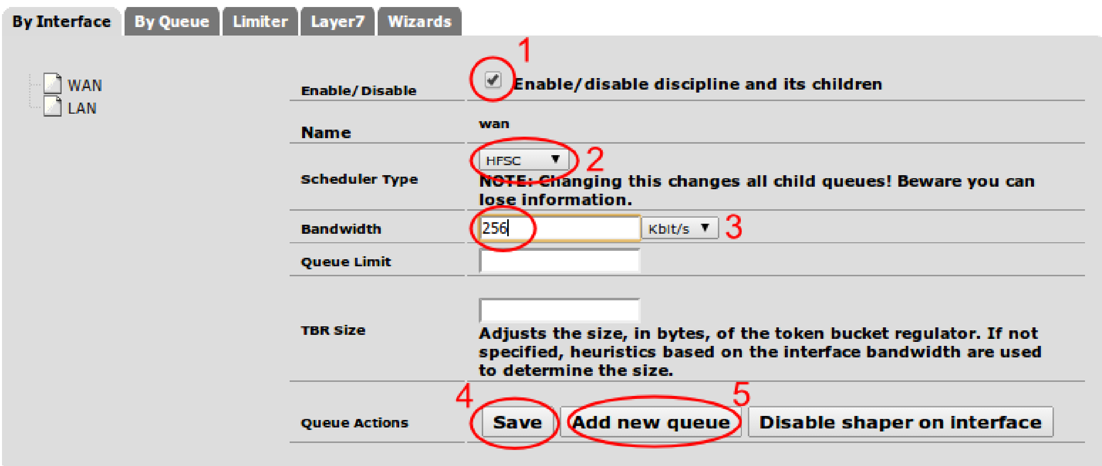
- The **external** interface of the pfSense virtual machine (*Network Adapter 1*) is Bridged with the external interface of your server (probably *eth0*).
- If your server has two network interfaces, then the **internal** interface of the pfSense virtual machine (*Network Adapter 2*) is Bridged with the internal interface of your server (probably *eth1*), and so is the only network interface (*Network Adapter 1*) of your client Virtual Machine. This allows you to connect laptops to *eth1* and use them to test your connection, as well as the client Virtual Machine.
- If your server has only one network interface, then the **internal** interface of the pfSense virtual machine (*Network Adapter 2*) is connected to the *Internal Network pfsense*, and so is the only network interface (*Network Adapter 1*) of your client Virtual Machine. This only allows you to test your connection from the client Virtual Machine.

Configure the Interfaces

We need to set the total bandwidth and the scheduler on each interface:



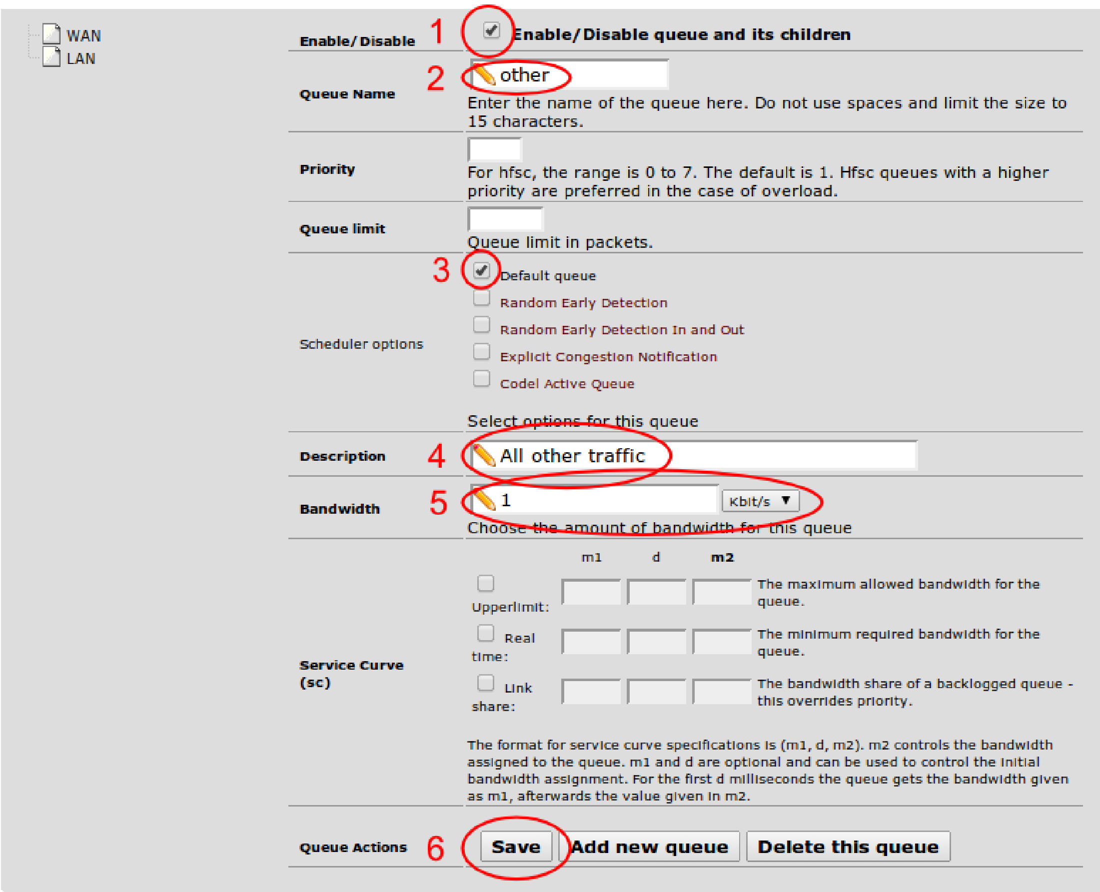
• Open the pfSense webConfigurator and log in.
• From the menu choose *Firewall/Traffic Shaper*.
• Click on the *WAN* interface.



• Check the box *Enable/disable discipline and its children*.
• Ensure that the scheduler type is set to *HFSC*.
• Set the *Bandwidth* to 256 Kbit/s.
• Click on the *Save* button.

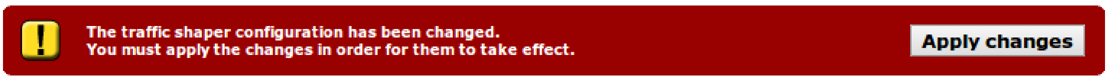
Now we need to add a queue to the interface.

• Click on the *Add new queue* button.



- Check the box *Enable/Disable queue and its children*.
- For the *Queue Name* enter `Other`.
- Check the box *Default queue*.
- For the *Description* enter `All other traffic`.
- For the *Bandwidth* enter `20` and choose *%*. * This is the WAN interface, so we are configuring the upstream bandwidth.
- Click on the *Save* button.

Repeat the whole process for the LAN interface, but set the *Bandwidth* of the interface to 1024 Kbit/s instead of 256. Create a queue called *other* on the LAN interface as well, but with the Bandwidth set to 17.5%.



Notice that pfSense tells you that you need to apply the changes to the traffic shaper configuration. Click on the *Apply* button.

Questions:

- Why do we have to create a queue?
- Why do we have to make it the Default queue?
- What speed will traffic be limited to on this interface?
- Which queue will all traffic be placed into, and why?

Answers:

There needs to be at least one queue, otherwise pfSense will not apply any bandwidth limits. If there is no Default queue, then traffic will not be placed into any queue, and therefore not be limited at all. pfSense will complain if there is at least one queue on the interface and none of them is the default queue, but it doesn't complain if there are no queues at all, it just doesn’t work.

No traffic is classified by any firewall rules, yet, so all traffic will go into the default queue.

Testing

From a computer behind the pfSense router (either your laptop or the client Virtual Machine), download a large file, for example:

```
$ wget -O /dev/null ftp://www.mirrorservice.org/sites/mirror.centos.org/6/isos/x86_64/CentOS-6.4-x86_64-bin-DVD1.iso
```

What speed do you get? How does it compare with the speed allocated to Other traffic above?

What happens if you edit the *Other* class on the *LAN* interface, enable *Upperlimit* and set the *Upperlimit m2* to 35%?

You should get a download speed of approximately 128 kBps, which equals 1024 kbps. This is because the Other class is allowed to borrow more bandwidth until it reaches the *Upperlimit*, or the speed limit on the interface, whichever is lower. In this case there is no *Upperlimit* set, so it can borrow up to 1024 kbps.

If you set an *Upperlimit* of 35% then it should not be able to use more than 35% of 1024 kbps, which is 44.8 kBps.

Traffic and Ping times

Try pinging the pfSense firewall (which will have the IP address 192.168.1.1 unless you’ve changed it in class.)

What happens to ping times with and without a download in progress? Why the difference?

Without a download in progress, you should see very short ping times, around 1 ms:

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=0.480 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=0.385 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=0.537 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=0.350 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=0.454 ms
```

With a download in progress, you should see much longer ping times, around 150 ms:

```
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_req=1 ttl=64 time=147 ms
64 bytes from 192.168.1.1: icmp_req=2 ttl=64 time=136 ms
64 bytes from 192.168.1.1: icmp_req=3 ttl=64 time=154 ms
64 bytes from 192.168.1.1: icmp_req=4 ttl=64 time=163 ms
64 bytes from 192.168.1.1: icmp_req=5 ttl=64 time=132 ms
```

This is because the ping packets must wait in the queue behind the download packets, when a download is in progress.

We can reduce this, at the cost of some dropped packets, by reducing the *Queue limit* on the WAN interface, Other class, to 5 or 10 packets. If a ping arrives when the output queue on the interface is full, then the reply packet will be dropped instead of placed in the queue.

Adding more queues

Edit the WAN interface and add two new classes:

- From the pfSense menu choose *Firewall/Traffic Shaper*.

- Click on the *WAN* interface.
- Click on the *Add new queue* button.
- Check the box *Enable/Disable queue and its children*.
- For the *Queue Name* enter `VoIP`.
- Make sure that the checkbox *Default queue* is not checked.
- For the *Description* enter `Voice over IP`.
- For the *Bandwidth* enter `50` and choose *%*. * This is the WAN interface, so we are configuring the upstream bandwidth.
- Check the box *Linkshare* and enter `30%` for the *m2* value.
- Click on the *Save* button.

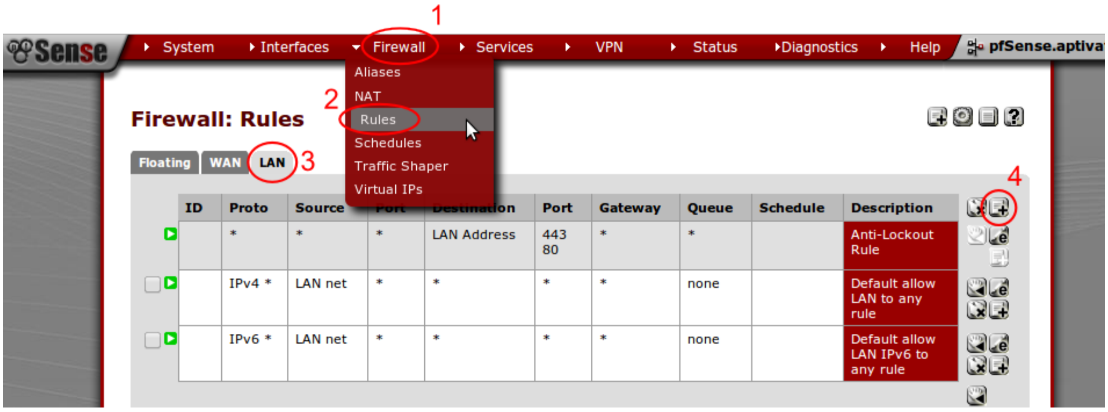
Add another queue called `HTTP`, with the description `Web traffic`, with 30% bandwidth and 30% linkshare.

Edit the LAN interface and add a queue called `VoIP`, as above, but with `12.5%` bandwidth.

Finally add another queue to the LAN interface, named `HTTP`, as above, but with `70%` bandwidth.

Filtering traffic into queues

We use firewall rules to assign traffic to a queue. The rule allows the *outbound* traffic, and at the same time assigns the returning packets into a queue.



- From the pfSense menu choose *Firewall/Traffic Shaper*.
- Click on the *LAN* tab.

If you already have a rule that applies to outbound HTTP traffic, you will need to change it, instead of creating a new rule:

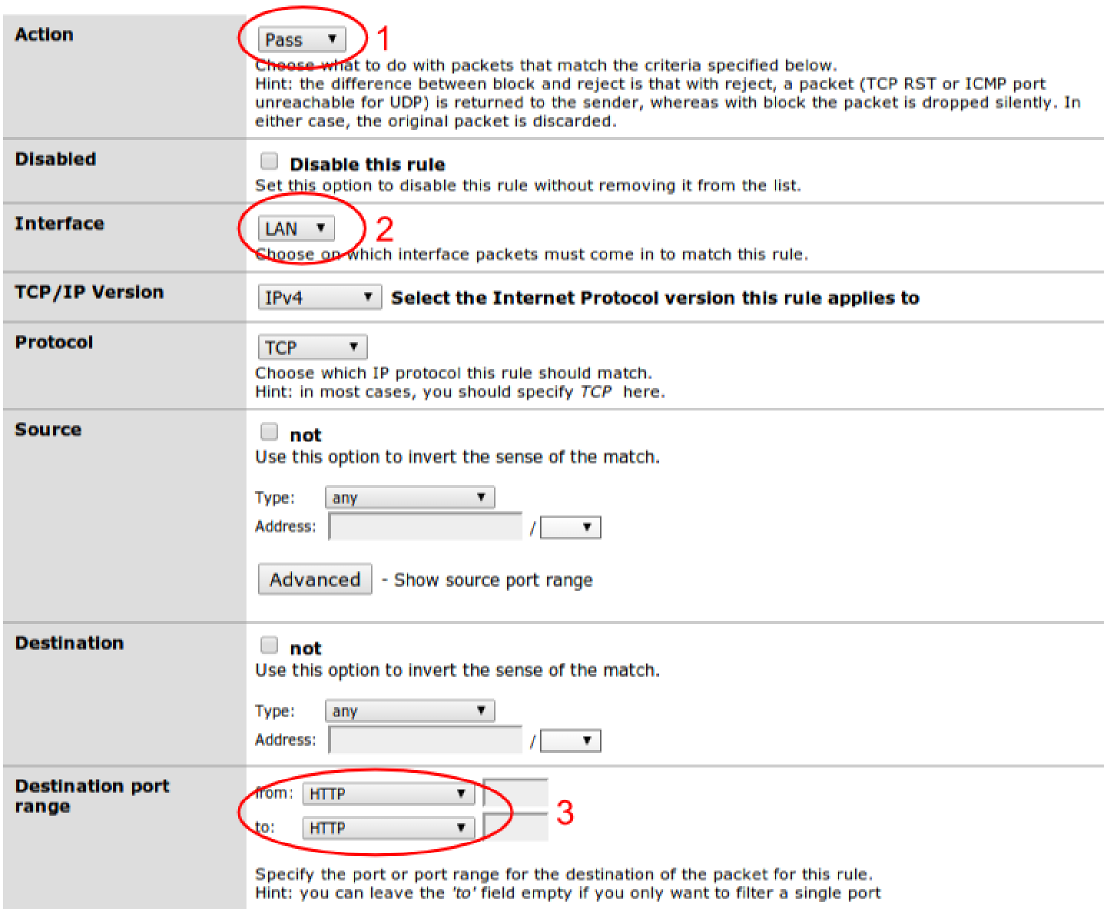
- If you create a new rule before that rule, it will override that rule because the firewall will match the new rule first.
- If you create a new rule after that rule, then it will never be hit, and your traffic will never be placed into the *http* queue.

If you have two rules left over from the Web Caching session, one which allows HTTP from the proxy server and one which blocks HTTP from all other computers, then you need to decide whether to only allow and restrict the speed of HTTP from the proxy server, or to allow all computers again.

I recommend that you modify the proxy server rule, to restrict traffic through the proxy. You'll also need to use the proxy server when conducting download speed tests below.

Adding filtering rules

- Click on the Add Rule button.



- For *Action* choose *Pass*. (should be the default).
- For *Interface* choose *LAN* (should already be set to this).
- For *Destination port range* choose HTTP.
- For *Description* enter `Place web traffic into http queue`.
- For *Ackqueue/Queue* click on the *Advanced* button, and choose *none/http*. * This is backwards for some bizarre reason. You probably always want to specify the queue and not the ackqueue).
- Click on the *Save* button.

Create another rule to filter UDP traffic into the VoIP queue:

- Click on the Add Rule button.
- For *Action* choose *Pass*. (should be the default).
- For *Interface* choose *LAN* (should already be set to this).
- For *Protocol* choose UDP.
- For *Description* enter `Place UDP into VoIP queue`.
- For *Ackqueue/Queue* click on the *Advanced* button, and choose *none/voip*.
- Click on the *Save* button.

Finally, create a very similar rule to place *ICMP* traffic (pings) into the VoIP queue. This allows us to measure VoIP latency and packet loss using the *ping* command.

You should see a prompt to apply changes to the firewall rules:

Click on the *Apply changes* button.

Note that we don't need to classify any traffic as *Other*. Because this is the default queue, all unclassified traffic will be placed in it automatically.

Testing

What effect is this likely to have on download speeds and ping times?

- Download speed is still about the same (119 kBps).
- Ping times massively reduced, to an average of 6 ms. (compared to 0.6 ms with no cross traffic, and 50-600 ms with cross traffic in the same queue).

How do classes share traffic?

If you run two downloads at the same time, for example run the following commands in separate terminals or on separate client VMs:

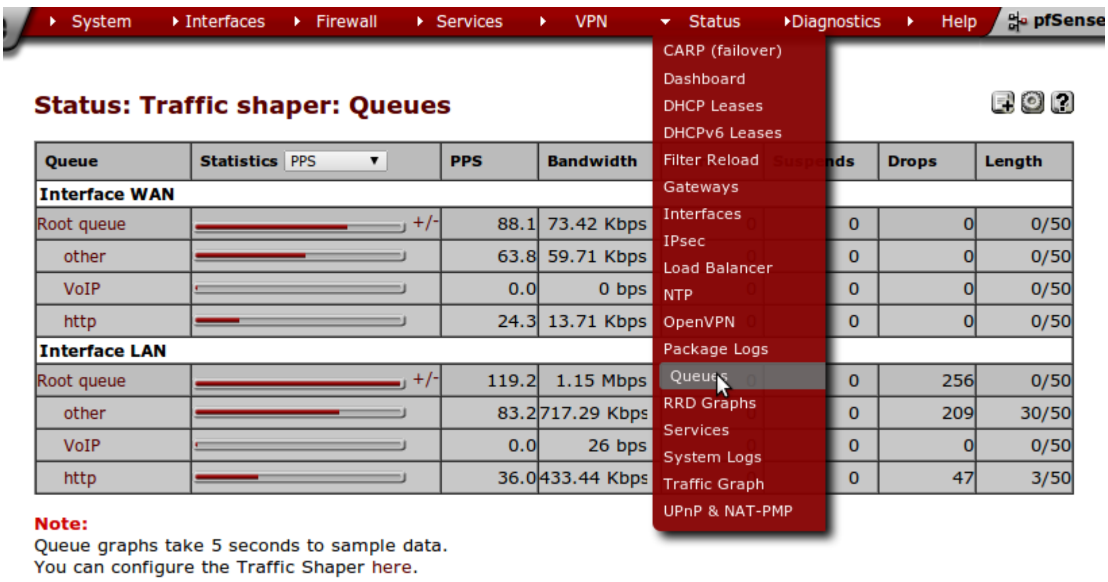
```
$ wget -O /dev/null ftp://www.mirrorservice.org/sites/mirror.centos.org/6/isos/x86_64/CentOS-6.4-x86_64-bin-DVD1.iso
$ wget -O /dev/null http://www.mirrorservice.org/sites/mirror.centos.org/6/isos/x86_64/CentOS-6.4-x86_64-bin-DVD1.iso
```

They should share bandwidth in the ratio of the *Bandwidth* assigned to each queue. So we'd expect to see something like this, assuming that the VoIP bandwidth is not being used, and therefore shared between the other classes in the ratio of their bandwidth allowance.

Protocol Realtime/Minimum Linkshare Total (%) Total (kbps) ========= ================= ========= ========== ============= HTTP 70% 10% 80% 819 FTP 17.5% 2.5% 20% 205 ========= ================= ========= ========== =============

Unfortunately the allocations are not very accurate, probably because the queues are sometimes empty, so there's no packet to send (see How Unfair can Weighted Fair Queuing be? for details of their results.)

You can see the current bandwidth used in each queue by choosing *Status/Queues* from the pfSense menu, which will give you a page like this:



Note that the *Queue Length* will vary as the TCP streams try to adjust their speed to the amount allocated by the traffic shaping. Every dropped packet will cause a TCP stream to reduce its speed, and cause the queue length to drop. The TCP stream will then try to adjust its speed slowly upwards, searching for the limit again. When the speed is higher than the allocated bandwidth, the queue will lengthen. When it becomes full again, another packet will drop and the speed will be reduced again. This process repeats as long as the TCP stream is running, like this:

Classifying inbound connections

Put a large file on the internal web server (Squid proxy VM). Add a port forwarding rule in pfSense, classifying traffic as HTTP:

- Choose *Firewall/NAT* from the pfSense menu.
- On the *Port Forwarding* tab, add a new rule.
- For *Destination port range* choose *HTTP*.
- For *Redirect target IP* enter `192.168.1.100` or the IP address of the internal web server/Squid proxy VM.
- For *Redirect target port* choose *HTTP*.
- For *Description* enter `Forward HTTP to internal web server`.
- Click on the *Save* button.

Now click on the *Edit* button next to the rule to edit it again, scroll down to *Filter rule association* and click on *View the filter rule*.

Scroll down to *Ackqueue/Queue*, click on the *Advanced* button and choose *none/none*. Then click on the *Save* button.

We also want to ping the pfSense external interface from outside, to measure the queue responsiveness. To do that, add a rule that Passes:

- Protocol ICMP, ICMP type echo-request
- Destination: WAN address
- Description: `Allow pings to pfSense external`.

You should see a prompt to apply changes to the firewall rules:

Click on the *Apply changes* button.

How can you test this?

Try to retrieve the file using the pfSense firewall's external IP address, which is forwarded to the internal server, and assuming that the pfSense WAN IP address is 192.168.6.128, use the following command:

```
$ wget http://192.168.6.128/bigfile -O /dev/null
```

And ping the external interface. What ping times do you get?

Now edit the NAT rule again, *View the filter rule*, and change the *Ackqueue/Queue* to *none/http*. Save the rule and click *Apply changes*.

You'll need to start the `wget` command again. What happens to the `ping` time, with and without the `wget` running?

FIN

Any questions?

Have we met the objectives?

- Configure traffic queues in pfSense
- Classify traffic into queues
- Monitor and debug bandwidth management

Please let us know if we haven't.