Network Management with the OpenBSD Packet Filter Toolset

BSDCan 2024

Ottawa, May 30, 2024

Peter Hansteen, Massimiliano Stucchi, Tom Smyth

Network: guottawa

Slides at https://nxdomain.no/~peter/pf_fullday.pdf

whoami (who am I), part Peter:

Peter Hansteen peter@bsdly.net

- Sysadmin, OpenBSD user since before the millenium
- Wrote The Book of PF, now in its third edition
- Blog at bsdly.blogspot.com about (lack of) sanity in IT
- Works at Tietoevry Create
- Yes, I'll do another book any decade now

whoami (who am I), part Max:

Massimiliano Stucchi

- Technical Advisor at The Internet Society
  - Here representing myself only
- IPv6 "Enthusiast"
- https://stucchi.ch
- @stucchimax@social.secret-wg.org

whoami (who am I), part Tom:

Tom Smyth

- working in IT since 2000
- CTO wireless Connect Ltd. an ISP in Ireland
- Opinions are mine and may be my companies also :)
- PF student, an avid reader of the Book of PF.
- I really Enjoy networking with OpenBSD
- Maintainer of the NSH network Shell for OpenBSD.

Introduce yourself

- A quick introduction about yourself:
  - Your name
  - Your favourite BSD
  - Your experience with networking
  - Your experience with PF
  - Your goal(s)

Agenda

1. PF Basics
2. Exercise: Host configuration
3. NAT and Redirects
4. Exercise: Setup a gateway
5. Hosting Services
6. Exercise: Hosting Services, redirects
7. Traffic Shaping
8. Exercise: Setting up queueing
9. Redundancy with CARP+pfsync
10. Exercise: Setting up failover firewalls
11. Tips
12. Troubleshooting
13. Exercise: NAT64
14. End

PF Basics

Section 1

It all started with a copyright violation

- OpenBSD up to 2.9 (2001) used Darren Reed's IPFilter
  - Almost, but not quite BSD licensed
  - No right to distribute changed versions
- IPFilter removed on May 29th, 2001
  - First commit of the new PF code June 24 2001 at 19:48:58 UTC
  - OpenBSD 3.0 release pushed to Dec 1 by the extra effort
- *License audit* of src tree + ports tree followed

But let's jump to the present -

Network design 101

How to build a maintainable network:

- Start with something simple
- Think dual stack (IPv6 and IPv4)
- Users
  - Access to local and remote services
  - and their security requirements
- The services you offer and the ones you consume
  - i.e: SMTP, WWW, DNS
- **Keep it simple, not stupid**

Know what a Firewall can (and cannot) do!

A firewall can:

- allow certain packets flowing to or through the firewall device
- drop certain packets flowing to or through the firewall device
- redirect or NAT packets according to policy

There are limits though

- Inbound interface bandwidth limitations
- CPU I/O interface driver packets per second(PPS) limitations

A firewall cannot

- block inbound Flood Denial of Service attacks that exceed:
  - the inbound interface capacity
  - the CPU I/O / interrupt capacity of the interface

Any software firewall can be overwhelmed with high packet rate Denial of Service attacks. If the scale of the attack exceeds the capability of the firewall

But first, how do we build rulesets?

The basic *rule* format is,

verb *criteria* **actions** ... options

```
pass in on egress proto tcp to egress port ssh
match out on egress nat-to egress
block all
```

You can do complete rulesets with only these, but you may also want to declare things ->

(**egress** ? -> the *interface group* consisting of interfaces that have a default route)

Next, declarations

Declaring objects your rules will use:

Macros:

```
macro-name = "definition"
```

Tables:

```
table <tablename>
```

Queues:

```
queue queuename on interface bandwidth number ...
```

You can also *set* options.

We'll be returning to all of those, in the meantime man pf.conf is always your friend.

Block by default

- A firewall is supposed to block by default, right?
- Let's start with the ruleset for a client
- Useful baseline:
  - block by default
  - pass traffic we generate

```
block
pass from (self)
```

In practice

- On my laptop in my home network, that expands to:

```
$ doas pfctl -vnf /etc/pf.conf
  block drop all
  pass inet6 from ::1 to any flags S/SA
  pass on lo0 inet6 from fe80::1 to any flags S/SA
  pass on iwm0 inet6 from fe80::a2a8:cdff:fe63:abb9 to any flags S/SA
  pass inet6 from 2001:470:28:658:a2a8:cdff:fe63:abb9 to any flags S/SA
  pass inet6 from 2001:470:28:658:8c43:4c81:e110:9d83 to any flags S/SA
  pass inet from 127.0.0.1 to any flags S/SA
  pass inet from 192.168.103.126 to any flags S/SA
```

Anatomy of a ruleset

- Ruleset evaluated top to bottom
- **Last match wins** (unless *quick* is used)
- Stateful by default
- For more details, see man pf.conf
    - http://man.openbsd.org/pf.conf

RULE order matters, keep in mind when you write your own later

A ruleset can contain

- **Options**
  - General configuration
- **Macros**
  - Content to be expanded
- **Tables**
  - When macros fall short (for IP addresses only)
- **Redirections**
- **Rules**
  - How PF should behave
- The order of the items in the ruleset used to be important, but not anymore

Options

- General configuration for pf
- Useful for debugging, applying default timeout values, etc.

```
set limit states 100000
```

- or

```
set debug debug
set loginterface dc0
set timeout tcp.first 120
set timeout tcp.established 86400
set timeout { adaptive.start 6000, adaptive.end 12000 }
```

Macros

- Content that will later be expanded in context
- Name must start with a *letter*, *digit*, or *underscore*
- Useful to keep rulesets simple and clean

```
ext_if = "vio0"
all_ifs = "{ $ext_if lo0 }"
pass out on $ext_if from any to any
pass in  on $ext_if proto tcp from any to any port 25
```

- Could also be port or address ranges

```
bacula_ports = "9101:9103"
dmz_hosts = "192.0.2.1 - 192.0.2.254"
```

Tables

- When IP address list macros grow too long, use tables instead
- Hold only *addresses* and *networks*
  - No port ranges
- Provide fast lookups

```
table <bruteforce> persist counters
```

- They can be used in rules

```
block from <bruteforce>
```

- Unlike macros with port or address lists, *do not* expand to multiple rules
  - More optimised ruleset

Tables continued

- Content can be loaded from files or from CLI

```
table <popflooders> persist counters file "/home/peter/popflooders"
```

useful if you save contents to preserve across reboots, such as

```
$ doas pfctl -t popflooders -T show >/home/peter/popflooders
```

possibly in cron jobs

- use pfctl to add/remove entries from command line

```
$ doas pfctl -t bruteforce -T add 192.0.2.11 2001:db8::dead:beef:baad:f00d
```

- optionally maintained by daemons such as spamd, bgpd and dhcpd

Implementing your spec

How would you do this in your home/soho infrastructure ?

Exercise 1

Protecting your host

Excercise 1 - Let's start

- Lab environment:
  - Open your favourite browser, then
  - Go to lab1.glevia.com
  - Username is "X" (where X is your assigned number)
  - Password is "pftutorial"

Exercise 1 - First steps

- Check that pf is indeed loaded and running (*hint*: pfctl)

- Wait until everyone has connected in order to proceed

Exercise 1 - net config

- Configure the external interface on gateway
- *vi /etc/hostname.vio0*

  Here, XX == your user # + 30

```
inet 10.255.255.XX/24
!route add 0/0 10.255.255.254
inet6 fd18:b5d:cafe::XX/64
!route add -inet6 2000::/3 fd18:b5d:cafe::a
!route add -inet6 fd00::/8 fd18:b5d:cafe::a
```

- and then *vi /etc/resolv.conf*

```
nameserver 10.255.255.254
nameserver fd18:b5d:cafe::a
```

followed by

```
sh /etc/netstart
```

Exercise 1 - on gateway

- Start with a block ruleset

```
block
pass quick inet6 proto tcp from fd18::/16 to port ssh
pass quick inet6 proto icmp6 from fd18::/16
```

- Allow traffic to be generated from your host, and allow ICMPv6

```
pass from self
```

and then, reload *pf.conf*

```
pfctl -vnf /etc/pf.conf
pfctl -f /etc/pf.conf
```

- **NB:** Reload pf this way after every statement in the exercises

Exercise 1 - Tests

- From your gateway ping a host
- First IPv6

```
# ping6 fd18:b5d:cafe::a
PING fd18:b5d:cafe::a (fd18:b5d:cafe::a): 56 data bytes
64 bytes from fd18:b5d:cafe::a: icmp_seq=0 hlim=64 time=0.548 ms
64 bytes from fd18:b5d:cafe::a: icmp_seq=1 hlim=64 time=0.492 ms
64 bytes from fd18:b5d:cafe::a: icmp_seq=2 hlim=64 time=0.494 ms
```

- Then IPv4

```
# ping stucchi.ch
PING stucchi.ch (45.129.224.40): 56 data bytes
64 bytes from 45.129.224.40: icmp_seq=0 ttl=56 time=6.264 ms
64 bytes from 45.129.224.40: icmp_seq=1 ttl=56 time=6.273 ms
64 bytes from 45.129.224.40: icmp_seq=2 ttl=56 time=6.117 ms
```

Exercise 1 - Wrap up

- Does ping work?
- Do other commands work?
  - working from total block, proceed to make restricted workstation
  - name resolution
  - *http* and *https*
- Access public web sites, other Internet resources.
- What would it take to access the other lab hosts?

Questions ?

Gateway, NAT and Redirects

Section 2

Easy! How do we improve on this?

- Make a 'firewall':
  - a point of policy enforcement
  - a gateway
  - filter for other hosts
  - redirection tricks

Introducing NAT

- **N**etwork **A**ddress **T**ranslation (RFC1631 onwards)
- -> 'Hide' several hosts behind 1 or more public addresses, using RFC1918 addresses
- -> can be used by ISPs for conserving scarce IP addresses in large networks (CG-NAT) 100.64.0.0/10
- Modern PF has *nat-to* on 'pass' and 'match' rules:

```
match out on $extif inet nat-to ($extif)
```

- *Neat trick*: egress is the interface group that has a default route, you can filter on it

```
match out on egress inet nat-to (egress)
```

- In modern networks we **should** (also) have IPv6 (inet6)
-

A (filtering) Gateway

"I decide which packets pass"

Enable forwarding:

- Temporarily set from command line with sysctl:

```
# sysctl net.inet.ip.forwarding=1
# sysctl net.inet6.ip6.forwarding=1
```

- Make permanent in /etc/sysctl.conf

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

The minimal gateway

- Do you *NAT* for IPv4? Of course you do.
- Do you run IPv6? Of course you do.

```
ext_if=vio0
int_if=vio1
match out on egress inet nat-to ($ext_if)
block all
pass proto tcp from { self, $int_if:network }
```

- The "pass" rule, withouth *inet* or *inet6* applies to both

Keep in mind: This is a point of policy enforcement

A Point of policy enforcement

- Now some policy, and macros

```
ext if=vio0
int if=vio1
client out = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
    https, 2628, 5999, 8000, 8080 }"
udp services = "{ domain, ntp }"
match out on egress inet nat-to ($ext_if)
block
pass quick proto { tcp, udp } to port $udp services keep state
pass proto tcp from $int if:network to port $client_out
pass proto tcp to self port ssh
```

- What services do your clients consume?

Letting dhcpd(8) direct access

OpenBSD dhcpd(8) can interact with your ruleset via tables:

/etc/rc.conf.local

```
dhcpd_flags="-L leased_ip_table -A abandoned_ip_table -C changed_ip_table bge1"
```

/etc/pf.conf

```
ext if=vio0
int if=vio1
table <abandoned ip table>  persist  counters
table <changed ip table>  persist  counters
table <leased ip table>  persist  counters
client out =  "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
                 https, 2628, 5999, 8000, 8080 }"
udp services =  "{ domain, ntp }"
match  out  on  egress  inet  nat-to  ($ext_if)
block
pass  quick  proto  { tcp, udp }  to  port  $udp services  keep  state
pass  proto  tcp  from  <leased ip table>  to  port  $client_out
pass  proto  tcp  to  self  port  ssh
```

=> only pass traffic from hosts with active leases from *me*

Redirects (and divert-to)

Modern PF has two classes of redirect

- **rdr-to** on match and pass rules - rewrite destination address while filtering (locally or even to other hosts)

```
pass in on egress to port www rdr-to $webserver
```

- **divert-to** on match and pass rules - divert() socket for local use

```
pass in on egress to port smtp divert-to 127.0.0.1 port spamd
```

FTP Proxy

- If your users need to access FTP services, ftp-proxy is what you need
- FTP does not easily pass through a block firewall, some help is needed

```
$ doas rcctl enable ftpproxy6
```

- or for IPv4

```
$ doas rcctl enable ftpproxy
```

- and then add an anchor and divert rules to your config

```
anchor "ftp-proxy/*"
 ...
pass in quick inet proto  tcp to port ftp divert-to 127.0.0.1 port 8021
pass in quick inet6 proto tcp to port ftp divert-to ::1 port 8021
pass out proto tcp from $proxy to port ftp
```

There is even a reverse mode (**-R**) for when you host FTP servers, see man ftp-proxy

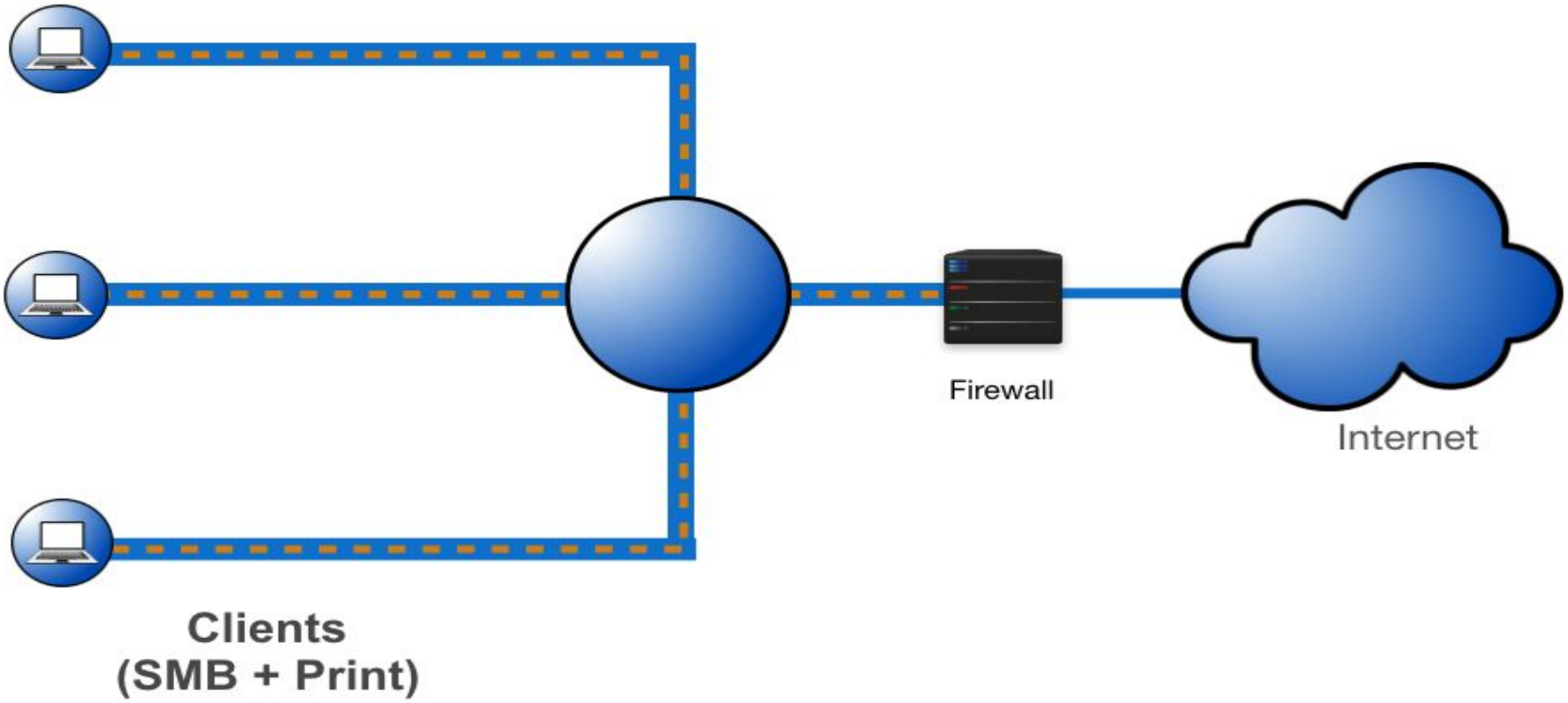
Exercise 2

Protecting your network

Exercise 2 - Goals

- Your network grows, you become a gateway
- Extend the configuration to enable the network to access the internet

Exercise 2 - Your network



Exercise 2

- Turn on ip forwarding (sysctl)

```
# sysctl net.inet.ip.forwarding=1
# sysctl net.inet6.ip6.forwarding=1
```

- Set up NAT

```
match out on egress inet nat-to (egress)
```

Also, pass traffic from that local net

Exercise 2 - preparation

- Configure the hosts with the following IPv6 addresses
  - *Gateway (vio1):* fd18:b5d:XX::a/64
  - *Host1:* fd18:b5d:XX::80/64
  - *Host2:* fd18:b5d:XX::25/64

- On Host1 and Host2, set fd18:b5d:XX::a as the default IPv6 gateway
- and also the following IPv4 addresses
  - *Gateway (vio1):* 192.168.XX.1/24
  - *Host1:* 192.168.XX.2/24
  - *Host2:* 192.168.XX.3/24
- On Host1 and Host2 set 192.168.XX.1 as the default IPv4 gateway

Exercise 2 - check your results

- From client 1, ping a host on the internet
- First IPv6

```
# ping6 stucchi.ch
PING stucchi.ch (2001:41d0:8:6ed8::80): 56 data bytes
64 bytes from 2001:41d0:8:6ed8::80: icmp_seq=1 hlim=56 time=7.414 ms
64 bytes from 2001:41d0:8:6ed8::80: icmp_seq=2 hlim=56 time=6.333 ms
64 bytes from 2001:41d0:8:6ed8::80: icmp_seq=3 hlim=56 time=6.441 ms
```

- Then IPv4

```
# ping stucchi.ch
PING stucchi.ch (37.59.51.141): 56 data bytes
64 bytes from 37.59.51.141: icmp_seq=0 ttl=56 time=6.264 ms
64 bytes from 37.59.51.141: icmp_seq=1 ttl=56 time=6.273 ms
64 bytes from 37.59.51.141: icmp_seq=2 ttl=56 time=6.117 ms
```

Exercise 2b: FTP

Try fetching *ftp://ftp.ripe.net/pub/stats/ripencc/delegated-ripencc-extended-latest*

```
# wget ftp://ftp.ripe.net/pub/stats/ripencc/delegated-ripencc-extended-latest
```

Check your result

If it didn't work, configure FTP-proxy and try again.

Hosting Services

Section 3

Hosting services behind your gateway

- Now you actually want some 'pass in on egress' rules :)
- Get your specifications clear, put in writing:
  - your services and the ports they use
  - the names could be in **/etc/services** already
  - the hosts (IP addresses) that run the services
  - decide who/what/where to be reachable for/from
- Check services requiring extra help (i.e. proxying).

Proxies and other helpers

- OpenBSD comes with several proxies and other service helper programs in the base system:
  - ftp-proxy (you guessed it)
  - relayd (load balancing and lots more)
  - spamd (if you run SMTP - annoy spammers)
  -
- There are also such things as squid and varnish (web proxies) in packages

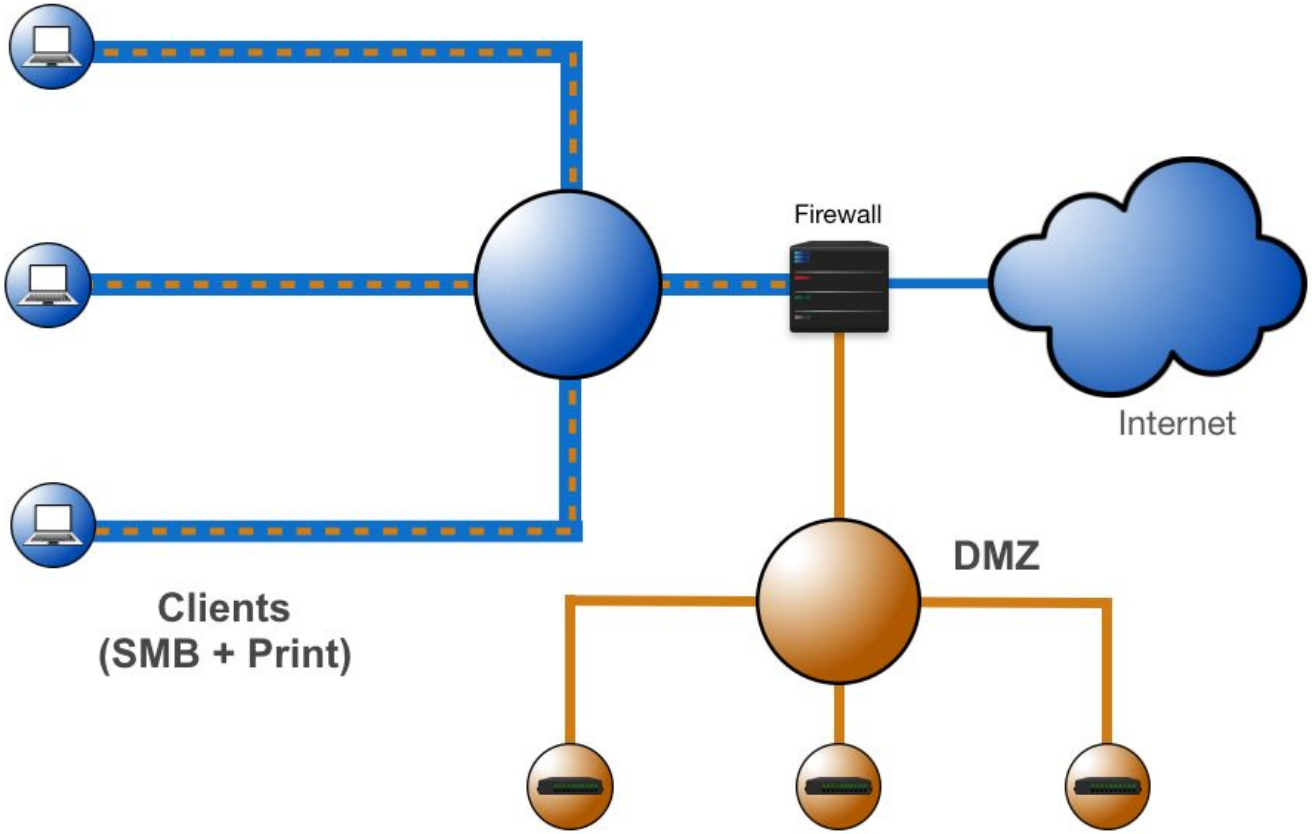
- Clients and services in the same subnet? Or do the DMZs?

DMZ, defined

'*D*e-*M*ilitarized *Z*one'

- when a group of host needs special treatment
- attached to separate interfaces(s), separate sub-rulesets
- And **YES**, you can have several
- Think multiple customers, *N* environments each

DMZ, illustrated



Allowing some services in

```
ext_if=vio0  # adjust to what your system has
int_if=vio1  # adjust to what your system has
client_out =  "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
            https, 2628, 5999, 8000, 8080 }"
udp_services =  "{ domain, ntp }"
webserver =  "192.0.2.227"
webports =  "{ http, https }"
emailserver =  "192.0.2.225"
email =  "{ smtp, pop3, imap, imap3, imaps, pop3s }"
nameservers =  "{ 192.0.2.221, 192.0.2.223 }"
match out on egress inet nat-to ($ext_if)
block
pass quick proto { tcp, udp }  to port $udp_services  keep state
pass proto tcp from $int_if:network  to port $client_out
pass proto tcp to self port ssh
pass proto tcp to $webserver  port $webports
pass proto tcp to $emailserver  port $email
pass log proto tcp from $emailserver  to port smtp
pass inet proto { tcp, udp }  to $nameservers  port domain
```

Now try loading this with *pfctl -vnf /etc/pf.conf* and see what this expands to (you can also fetch pf.services01.conf and try)

Tackling noise (attacks) with state-tracking options

Scenario: ssh bruteforcers

In our previous ruleset, add

```
table <bruteforce> persist counters
block from <bruteforce>
```

and change the **ssh** rule to

```
pass proto tcp to port ssh flags S/SA keep state \
   (max-src-conn 15, max-src-conn-rate 2/10, overload <bruteforce> flush global)
```

- Tune to taste
- More info on options in man pf.conf
- Remember *pfctl expire*

Tackling noise (attacks) with state-tracking options

Scenario: Wordpress site

- Wordpress is a usual target for many different attacks
- -> Make the attackers suffer by forcing them through smaller "windows"
  - Use tables and block by connection rates
- Mix and match settings, consult man pf.conf and remember *pfctl expire*
- Also see Forcing the password gropers through a smaller hole with OpenBSD's PF queues, Badness, Enumerated by Robots (blog posts) + The Book of PF

Scenario: Wordpress site

- You can re-use the *bruteforce* table or make a separate one, like

```
table <web_brutes> persist counters
block from <web_brutes>
```

- Now change the **$webports** rule to

```
pass proto tcp to port $webports flags S/SA keep state \
  (max-src-conn 15, max-src-conn-rate 5/10, overload <web_brutes> flush global)
```

- Alternatively, on a machine (the gateway) that does not run Wordpress, do

```
grep wp-login /var/www/logs/access.log | awk '{print $1}' | \
  sort -u | xargs doas pfctl -t web_brutes -T add
```

- (**Wordpressers**: Do these look right? Consult web logs and tcpdump output.)

Annoying spammers with spamd

- spamd(8) is good, clean, fun
- Speaks enough SMTP to do greylisting
- Can tarpit known bad senders and generate blacklists by greytrapping
- Default spamd.conf gives you one blacklist import and basics
  - (**Hint:** no real SMTP service required)
- You can even generate your own blacklists by *greytrapping* via *non-deliverable* spamtrap addresses in your own domain(s)
- Also see:
  - The Book of PF
  - In The Name Of Sane Email: Setting Up OpenBSD's spamd(8) With Secondary MXes
  - Maintaining A Publicly Available Blacklist

Annoying spammers with spamd

- Set up tables
- Divert traffic to the spamd process
- Only let the "good guys" pass to the real SMTP daemon

```
table <spamd-white> persist
table <nospamd> persist file "/etc/mail/nospamd"
pass in on egress proto tcp to any port smtp divert-to 127.0.0.1 port spamd
pass in on egress proto tcp from <nospamd> to any port smtp
pass in log on egress proto tcp from <spamd-white> to any port smtp
pass out log on egress proto tcp to any port smtp
```

- and then run

```
$ doas rcctl enable spamd
$ doas rcctl start spamd
```

TIP: check out smtpctl spf walk <*nospamd_domains.txt* to feed your *nospamd* table from live SPF data (in OpenBSD 6.3 onwards), see the blog post Goodness, Enumerated by Robots. Or, Handling Those Who Do Not Play Well With Greylisting

Scenario: SYN flood vs syncookies

A common Denial-of-Service (DOS) technique is to send large numbers of SYNs from spoofed addresses, filling up the state table.

In OpenBSD 6.3 and newer we have the pf.conf option

```
set syncookies
```

The default is off, if you enable with

```
set syncookies on
```

all SYNs get SYNACK answer, but no resources allocated until ACK received (pending match to **pass** rule)

The other option is **adaptive** with syncookies used only when half open TCP connections reach the **start** percentage, until the **end** level is reached

```
set syncookies adaptive (start 29%, end 15%)
```

Consider: what about that separate DMZ?

- What do you need?
- IP addresses: Segment off or allocate separate address ranges
- Attach each segment to separate interface, VLAN
- Do the ruleset surgery
  - which services do you run, where
  - do you need renumbering?
  - what traffic (in and out) is actually required?

Questions ?

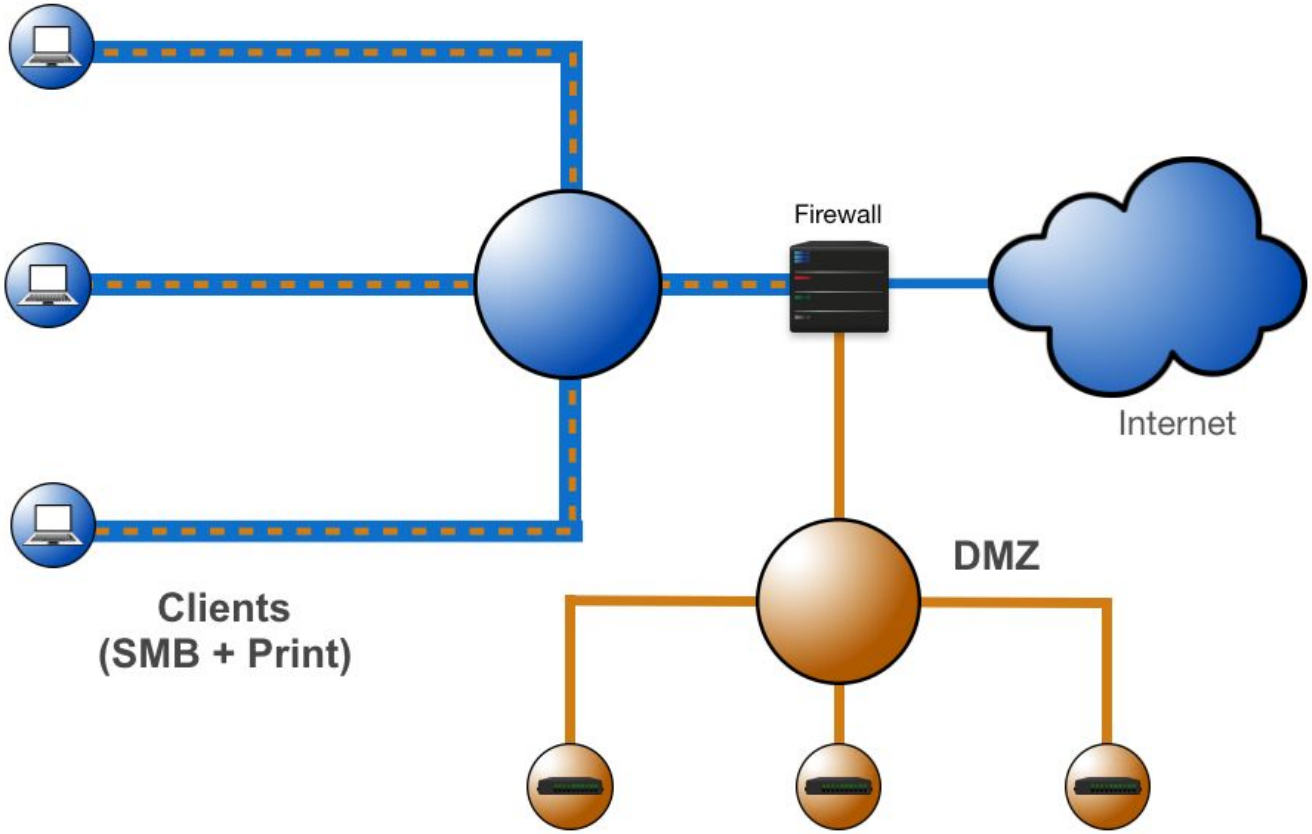
Exercise 3

Offering services

Excercise 3 - Goals

- You're now offering services
- **Host 1** will provide **http** service
- **Host 2** will provide **smtp** service
- We need to setup:
  - The services
  - Redirects
  - Firewall rules

Exercise 3 - Network



Exercise 3 - on Host1

- We need to configure and start httpd

```
# cp /etc/examples/httpd.conf /etc/httpd.conf
< comment out the HTTPS part >
# rcctl enable httpd
# rcctl start httpd
httpd(ok)
```

Exercise 3 - on Host2

Stop the *smtpd* daemon

```
# rcctl stop smtpd
```

- Change the config to listen on all interfaces:
  - Change the appropriate line in

/etc/mail/smtpd.conf

```
 listen on all
```

- Then start the daemon

```
# rcctl enable smtpd
# rcctl start smtpd
smtpd(ok)
```

- (It might take a while)

Exercise 3 - on gateway

/etc/pf.conf

```
webserver_v4 = "$IP addr of host1"
webserver_v6 = "fd18:b5d:XX::80"
webports = "{ http, https }"
emailserver_v4 = "$IP addr of host2"
emailserver_v6 = "fd18:b5d:XX::25"
email = "{ smtp, pop3, imap, imaps, pop3s }"
match in on egress inet proto tcp to egress port $webports rdr-to $webserver_v4
match in on egress inet proto tcp to egress port $email rdr-to $emailserver_v4
pass inet proto tcp to $webserver_v4 port $webports
pass inet proto tcp to $emailserver_v4 port $email
pass log inet proto tcp from $emailserver_v4 to port smtp
pass inet6 proto tcp to $webserver_v6 port $webports
pass inet6 proto tcp to $emailserver_v6 port $email
pass log inet6 proto tcp from $emailserver_v6 to port smtp
```

- **NB:** No redirects are needed for IPv6

Exercise 3 - checks

- Try connecting to the HTTP and SMTP port of your friends/neighbours:
- From Gateway:

```
telnet -6 fd18:b5d:XX::80 80
telnet -4 10.255.255.XX 80
```

- and

```
telnet -6 fd18:b5d:XX::25 25
telnet -4 10.255.255.XX 25
```

Tips

- Decide your network topology
  - DMZ (?)
  - Multi-customer (?)
  - Multi-customer, Multi-DMZ(?)
- Segment off your subnets
  - IPv4 (Do you NAT)?
  - IPv6
  - Do you do NAT64?
- Per subnet (customer)
  - Which services do you expose?
  - Write the rules
  - pamper^H^H^H^H^Hproxying

Traffic Shaping

Section 4

Traffic shaping

- OpenBSD has three separate shaping techniques:
  - *priorities* (set prio), introduced in OpenBSD 5.0
  - *queues*, introduced in OpenBSD 5.5, and
  - *flows*, introduced in OpenBSD 6.2 (aka FQ-CoDel)

- **Remember:**
  - Traffic shaping is about *dropping packets*
  - But is only relevant when there's a *reason* to start dropping
  - Then you get to pick which ones using the traffic shaping tools

- Works only one way, **outbound**

Traffic shaping with priorities

- In OpenBSD, every packet has a priority
- Possible values are 0 (garbage) through 7 (want!)
  - Default for most traffic is 3.
- So if you want *all* ssh traffic to move ahead of other traffic you could do a
- ```
  pass proto tcp to port ssh set prio 6
  ```
- and assign specific, non-default (!= 3) values to others.
  ```

Beating the FIFO with prio

- By default, packets are serviced on a first come, first served basis
  - or 'First in, First out' (FIFO).
- But: TCP wants ACKs for sent packets within a reasonable time
  - Otherwise the packet is considered lost, and retransmit will follow (transfer stalls).
- ACKs are tiny and have their TOS set to lowdelay, and this trick will cheat the FIFO:

```
match out on $ext_if proto tcp from $ext_if set prio (3, 7)

match in  on $ext_if proto tcp to $ext_if set prio (3, 7)
```

as per the man page,

If two priorities are given, TCP ACKs with no data payload and packets which have a TOS of lowdelay will be assigned to the second one. Packets with a higher priority number are processed first, and packets with the same priority are processed in the order in which they are received.

- See Prioritizing empty TCP ACKs with pf and ALTQ by Daniel Hartmeier for the ALTQ way

FQ-CoDel flows for fair bandwidth sharing

Introduced in OpenBSD 6.2, the FQ-CoDel algorithm (see RFC8290) defines **flows** to set up fair sharing for a specified number of simultaneous connections

Enable for your configuration with something like

```
queue outq on vio0 bandwidth 18M max 18M flows 1024 qlimit 1024 \
                default
```

Estimate your approximate high number of simultaneously actively transmitting sessions, put that number in (up to 32767)

Shaping with HFSC queues - fixed sizes

- When priorities don't quite cut it, you can slice your bandwidth into queues.
- For static shaping, give bandwidth values in absolute values:
- Only leaf queues can be assigned traffic
  - make sure allocations sum up to parent queue allocation
- Unless quotas approach saturation, no actual shaping (dropping) will take place

Shaping with HFSC queues - fixed sizes

```
queue main on $ext_if bandwidth 20M
     queue defq parent main bandwidth 3600K default
     queue ftp parent main bandwidth 2000K
     queue udp parent main bandwidth 6000K
     queue web parent main bandwidth 4000K
     queue ssh parent main bandwidth 4000K
       queue ssh_interactive parent ssh bandwidth 800K
       queue ssh_bulk parent ssh bandwidth 3200K
 queue icmp parent main bandwidth 400K
```

Shaping with HFSC queues - fixed assignment

- You can either do queue assignment with match rules:

```
match log quick on $ext_if proto tcp to port ssh \
   queue (ssh_bulk, ssh_interactive)
match in quick on $ext_if proto tcp to port ftp queue ftp
match in quick on $ext_if proto tcp to port www queue http
match out on $ext_if proto udp queue udp
match out on $ext_if proto icmp queue icmp
```

- treat filtering (block, pass) in separate rules,
- *or*
  - append 'set queue' to individual pass rules
- Any traffic not explicitly assigned goes in the default queue

Shaping with HFSC queues - flexible allocations (min, max, burst)

You can build in flexibility:

```
queue rootq on $ext_if bandwidth 20M
       queue main parent rootq bandwidth 20479K min 1M max 20479K qlimit 100
            queue qdef parent main bandwidth 9600K min 6000K max 18M default
            queue qweb parent main bandwidth 9600K min 6000K max 18M
            queue qpri parent main bandwidth 700K min 100K max 1200K
            queue qdns parent main bandwidth 200K min 12K burst 600K for 3000ms
       queue spamd parent rootq bandwidth 1K min 0K max 1K qlimit 300
```

Note here the added *min* and *max* values: combinded queue bandwidth can exceed actual sum; this gets you upper and lower bound within physical limits.

burst N for M - allow bursts of that size and length

qlimit - size of the queue's holding buffer - larger values *may* delay (packets are kept longer before sending)

Systat

- Available on all the BSDs
  - OpenBSD version has added functionalities
- Offers view into your system's traffic
  - Live queues, states, rules
- Check how your rules are behaving on your system in realtime
- More info in the man page systat(1)

queue monitoring - systat queues

```
 1 users Load 2.56 2.27 2.28                        skapet.bsdly.net 20:55:50
QUEUE                  BW SCH   PRI     PKTS    BYTES    DROP_P    DROP_B QLEN BOR SUS   P/S    B/S
rootq on vio0         20M               0       0        0         0   0           0     0
main                  20M               0       0        0         0   0           0     0
 qdef                  9M         6416363  2338M      136     15371   0         462 30733
 qweb                  9M          431590 144565K        0         0   0         0.6   480
 qpri                  2M         2854556 181684K        5       390   0          79  5243
 qdns                100K          802874  68379K        0         0   0         0.6    52
spamd                  1K          596022  36021K  1177533  72871514 299           2   136
```

or

queue monitoring - pfctl

```
$ doas pfctl -vsq
 [ pkts:          0  bytes:          0  dropped pkts:      0 bytes:      0 ]
 [ qlength:   0/ 50 ]
queue rootq on bge0 bandwidth 20M qlimit 50
 [ pkts:          0  bytes:          0  dropped pkts:      0 bytes:      0 ]
 [ qlength:   0/ 50 ]
queue main parent rootq bandwidth 20M, min 1M, max 20M qlimit 100
 [ pkts:          0  bytes:          0  dropped pkts:      0 bytes:      0 ]
 [ qlength:   0/100 ]
queue qdef parent main bandwidth 9M, min 8M, max 18M default qlimit 50
 [ pkts:    6517150  bytes: 2458545319  dropped pkts:    136 bytes:  15371 ]
 [ qlength:   0/ 50 ]
queue qweb parent main bandwidth 9M, min 8M, max 18M qlimit 50
 [ pkts:     431741  bytes:  148072219  dropped pkts:      0 bytes:      0 ]
 [ qlength:   0/ 50 ]
queue qpri parent main bandwidth 2M, min 700K, max 2M burst 4M for 3000ms qlimit 50
 [ pkts:    2855418  bytes:  186101241  dropped pkts:      5 bytes:    390 ]
 [ qlength:   0/ 50 ]
queue qdns parent main bandwidth 100K, min 12K burst 600K for 3000ms qlimit 50
 [ pkts:     803548  bytes:   70079760  dropped pkts:      0 bytes:      0 ]
 [ qlength:   0/ 50 ]
queue spamd parent rootq bandwidth 1K, max 1K qlimit 300
 [ pkts:     596424  bytes:   36910940  dropped pkts: 1178456 bytes: 72928604 ]
 [ qlength: 300/300 ]
```

add another v (pfctl -vvsq) for continuously updating display

Questions ?

Exercise 4

Queueing

Exercise 4 - Goals

- With the configs from exercise 3, now add:
- A set of queues, and
- Statements to add rules to the queues

Exercise 4 - on Gateway

- Configure the queues

/etc/pf.conf

```
queue rootq on $ext_if bandwidth 20M
   queue main parent rootq bandwidth 20479K min 1M max 20479K qlimit 100
       queue defq parent main bandwidth 9600K min 6000K max 18M default
       queue http parent main bandwidth 9600K min 6000K max 18M
       queue smtp parent main bandwidth 9600K min 6000K max 18M
   queue spamd parent rootq bandwidth 1K min 0K max 1K qlimit 300
```

Exercise 4 - on Gateway

- and then apply them to the match statements

/etc/pf.conf

```
match in on egress inet proto tcp to egress port $webports rdr-to $webserver_v4 \
     queue http
match in on egress inet proto tcp to egress port $email rdr-to $emailserver_v4 \
     queue smtp
pass inet6 proto tcp to $webserver_v6 port $webports set queue http
pass inet6 proto tcp to $emailserver_v6 port $email set queue smtp
pass log inet6 proto tcp from $emailserver_v6 to port smtp set queue smtp
```

Exercise 4 - Check

- Check the queues have been effectively created

```
# systat queues
```

- or, alternatively

```
# pfctl -vsq
```

CARP and pfsync

Section 5

CARP and pfsync

Common Address Redundancy Protocol (CARP)

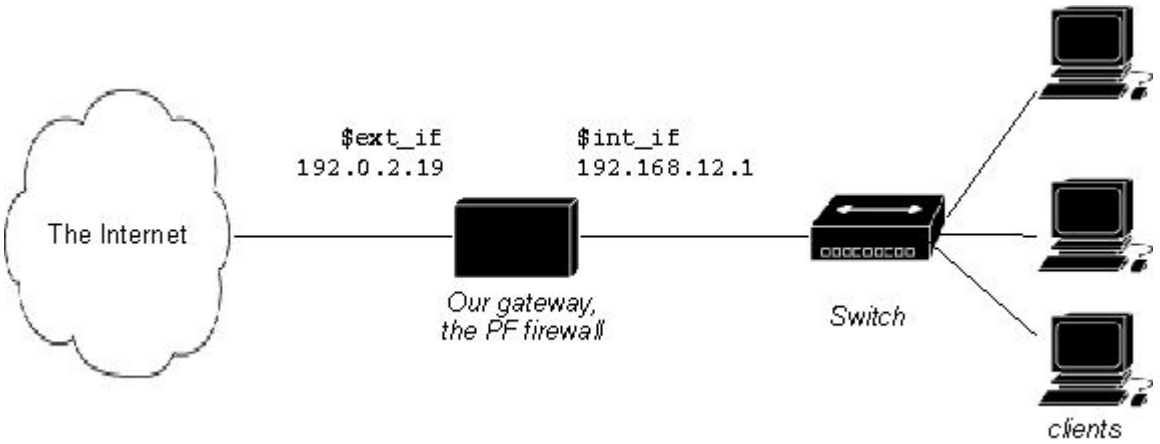
- Introduced with OpenBSD 3.5
- Patent free alternative to VRRP (RFC 2281, 3768, patent owners: Cisco, IBM, Nokia)
- Firewall/server redundancy
- Virtual network interface for automatic failover

pfsync

- Virtual network interface (assigned to physical interface)
- Handles syncronization between PF firewalls (in advance of failover)

CARP: project spec

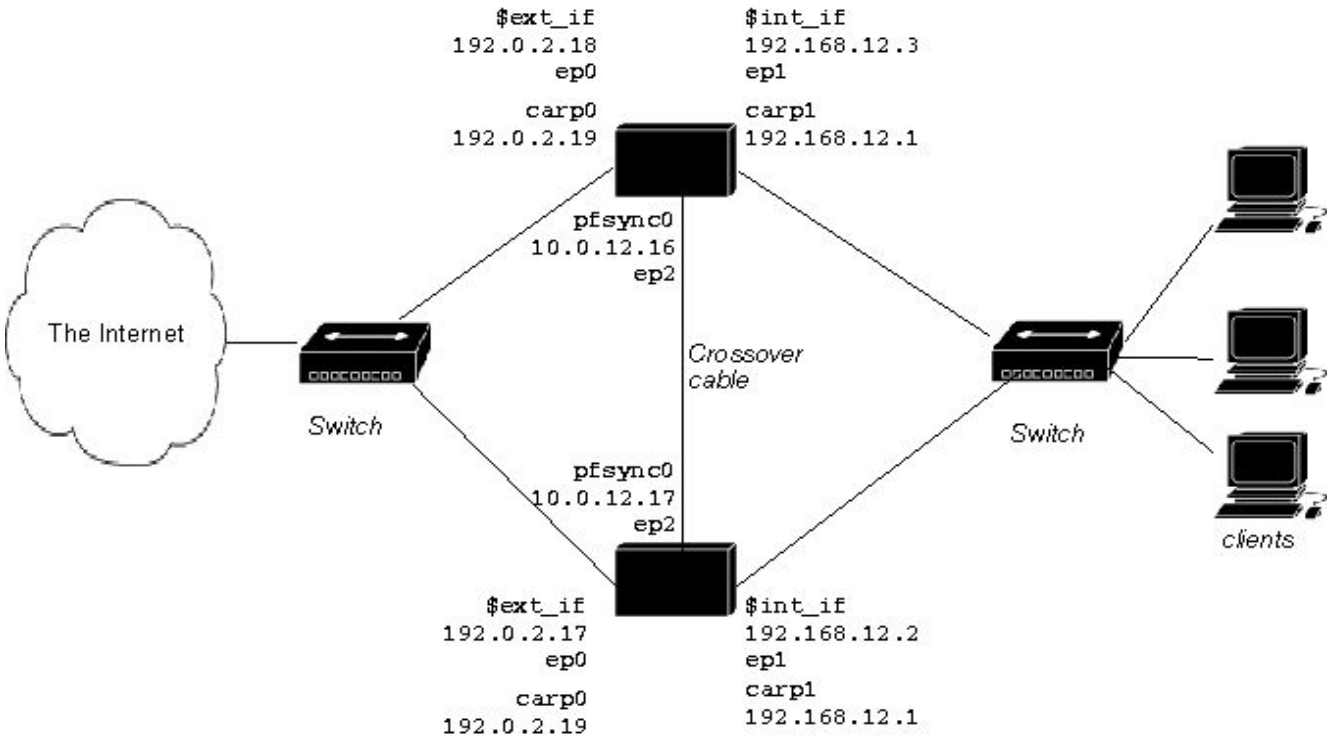
Our network - How to build a maintainable network:



becomes (...)

CARP: project spec

Our network becomes:



CARP: project spec cont'd

Our network should

- Keep functioning much the same way as it did earlier
- Have better availability with no noticeable downtime
- Experience graceful failover with no interruption of active connections

Tall order, huh?

Is Your System CARP Ready?

- OpenBSD: GENERIC kernel comes with carp and pfsync devices compiled in
- FreeBSD: GENERIC kernel does not have carp or pfsync devices enabled, must be enabled in kernel config and compiled in

Setting up CARP

You need the sysctls

```
$ sysctl net.inet.carp.allow
net.inet.carp.allow=1
$ sysctl net.inet.carp
net.inet.carp.allow=1
net.inet.carp.preempt=0
net.inet.carp.log=0
```

to let the magic work, we need

```
$ doas sysctl net.inet.carp.preempt=1
```

CARP: ifconfig

on the *master (XX == user #)*

```
$ doas ifconfig carp0  10.255.255.X9  carpdev vio0 vhid  1
$ doas ifconfig carp1  192.168.XX.19  carpdev vio1 vhid  2
```

on the *backup*

```
$ doas ifconfig carp0  192.255.255.X9  carpdev vio0 vhid  1 advskew 100
$ doas ifconfig carp1  192.168.XX.19  carpdev vio1 vhid  2 advskew 100
```

NOTE: On OpenBSD 5.7 onwards, explicit *carpdev* is required

the master announces every (1 + 0/256) seconds

the backup announces every (1 + 100/256) seconds

Note: Multicast by default. Use *carppeer* option for unicast. It' also possible to set the MAC address explicitly with the *lladdr* option.

Also see Henning Brauer's notes

pfsync

Use a physically separate net (crossover cable, separate VLAN)

```
$ doas ifconfig pfsync0 syncpeer 10.0.0.1 syncdev vio2
$ doas ifconfig pfsync0 syncpeer 10.0.0.2 syncdev vio2
```

What Happens To The Rule Set?

Pass CARP traffic on the appropriate interfaces

```
pass on $carpdevs proto carp keep state
```

Pass pfsync traffic on the appropriate interfaces

```
pass on $syncdev proto pfsync
```

Some traffic doesn't make sense to fail over

```
pass in on $int_if from $ssh_allowed to self keep state (no-sync)
```

PF sees the traffic on the physical interface

CARP Config Example (*cont*)

master:

sysctl.conf

```
net.inet.carp.preempt=1
```

hostname.carp0

```
pass mekmitasdigoat 10.255.255.X9 carpdev vio0 vhid 1
```

hostname.carp1

```
pass mekmitasdigoat 192.168.XX.X9 carpdev vio1 vhid 2
```

CARP config example (*cont'd*)

backup:

sysctl.conf

```
net.inet.carp.preempt=1
```

hostname.carp0

```
pass mekmitasdigoat 10.255.255.X9 carpdev vio0 vhid 1 advskew 100
```

hostname.carp1

```
pass mekmitasdigoat 192.168.XX.X9 carpdev vio1 vhid 2 advskew 100
```

CARP - remember pfsync

master:

hostname.pfsync0

```
syncpeer 10.0.0.2 syncdev vio2
```

backup:

hostname.pfsync0

```
syncpeer 10.0.0.1 syncdev vio2
```

CARP Ruleset

```
ext_if=vio0

int_if=vio1

carpdevs="vio0 vio1"

int_carp=carp1

ext_carp=carp0

match out inet on $ext_if from $int_if:network nat-to ($ext_carp)
```

CARP Load Balancing Mode

Load balancing mode: many masters

first node */etc/hostname.carp0*

```
pass mekmitasdigoat 192.0.2.19 balancing ip carpnodes 5:100,6:0 carpdev vio0
```

first node */etc/hostname.carp1*

```
pass mekmitasdigoat 192.168.12.1 balancing ip carpnodes 3:100,4:0 carpdev vio1
```

second node */etc/hostname.carp0*

```
pass mekmitasdigoat 192.0.2.19 balancing ip carpnodes 5:0,6:100 carpdev vio0
```

second node */etc/hostname.carp1*

```
pass mekmitasdigoat 192.168.12.1 balancing ip carpnodes 3:0,4:100 carpdev vio1
```

Load Balancing CARP: ifconfig

```
$ ifconfig carp
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>mtu 1500
       lladdr 01:00:5e:00:01:05
       priority: 0
       carp: carpdev vio0 advbase1 balancing ip
               state MASTER vhid 5 advskew 0
               state BACKUP vhid 6 advskew 100
       groups: carp
       inet 192.0.2.19 netmask 0xffffff00 broadcast 192.0.2.255
       inet6 fe80::200:24ff:fecb:1c10%carp0 prefixlen 64 scopeid 0x7
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST>mtu 1500
       lladdr 01:00:5e:00:01:03
       priority: 0
       carp: carpdev vio1 advbase1 balancing ip
               state MASTER vhid 3 advskew 0
               state BACKUP vhid 4 advskew 100
       groups: carp
       inet 192.168.12.1 netmask 0xffffff00 broadcast 192.168.12.255
       inet6 fe80::200:24ff:fecb:1c10%carp1 prefixlen 64 scopeid 0x8
pfsync0: flags=41<UP,RUNNING> mtu 1500
       priority: 0
       pfsync: syncdev: vio2 syncpeer:10.0.12.17 maxupd: 128 defer: off
       groups: carp pfsync
```

Also, run *systat states* on all nodes, watch states -- it's good fun :)

Exercise 5

CARP and pfsync

CARP: config initial master

sysctl.conf

```
net.inet.carp.preempt=1
```

hostname.carp0

```
pass mekmitasdigoat 10.255.255.X9 carpdev vio0 vhid 1
```

hostname.carp1

```
pass mekmitasdigoat 192.168.XX.X9 carpdev vio1 vhid 2
```

CARP: config initial backup

sysctl.conf

```
net.inet.carp.preempt=1
```

hostname.carp0

```
pass mekmitasdigoat 10.255.255.X9 carpdev vio0 vhid 1 advskew 100
```

hostname.carp1

```
pass mekmitasdigoat 192.168.xx.X9 carpdev vio1 vhid 2 advskew 100
```

Exercise 5 - Goals

- Set up redundant pair of gateway / firewalls
  - carp0 (external interface)
  - carp1 (internal interface)
  - pfsync0 (state table sync)
- test failover
  - ping carp addresses
  - ping from LAN to pftutorial.net
  - force failover (boot current master)
  - observe results

Exercise 5

Consider: Clients’ default gateway

Make already configured address virtual and change the physical one on our gateway?

Or add a virtual address and change the default gateway for clients?

Discuss :)

Tips

Section 6

Choosing your ISP, a quick guide

- Are they national or regional IX members?
- Do they have geographical redundancy ?
  - or do you need to arrange that for yourself ?
- Do they actually understand your questions about peering, routing, multiple paths?
  - (avoid consumer oriented SOHO-only shops)
- Do they *suck*?

Getting transit

- Find well peered transit providers
  - Can improve quality and shorten AS paths
  - No capacity problems

- Find your top traffic destinations:
  - Can improve quality
  - Peer with them or find closer upstream
  - Traffic profile from flow collectors can be useful

Common mistakes

- No diversity
  - All reached over same cable
  - All connect to the same transit
  - All have poor onward transit and peering arrangements

- Signing up with too many transit providers
  - Lots of small circuits
  - These cost more per Mbps than larger ones

Basic OpenBGPd configuration, operation and interaction with PF

- **B**order **G**ateway **P**rotocol
  - Manage and exchange route information with BGP peers
- Once you have the ASn registered, do the basic config.
- In your *pf.conf*:
  - enable BGP to pass between your routers and your peers' -- **TCP and UDP 179**
- **Neat trick**: Define tables in your pf.conf
  - bgpd maintains them via **pftable** attributes on bgpd.conf objects

Use cases for OSPF, BGP or ECMP

- **OSPF: O**pen **S**hortest **P**ath **F**irst
  - is a IGP **I**nterior **G**ateway **P**rotocol
  - Each router maintains link state information for links and networks within your AS
  - Calculates routing cost
  - Use ospf6d for IPv6
  - Use ospfd for IPv4
  - Need to *pass proto ospf* between routers.
- **BGP:** announces and receives routes
  - can be both an IGP or EGP **E**xterior **G**ateway **P**rotocol
  - highly scalable (Internet scale)
  - can be used for signaling and sending additional information with route announcements
  - Use bgpd
  - need to *pass proto tcp port 179* between routers

Use cases for OSPF, BGP or ECMP (cont)

- **ECMP: E**qual **C**ost **M**ulti-**P**ath
  - target reachable via more than one route
  - load distribution or redundancy over multiple links
  - **Tip** Use ifstated to handle link downtime.

BCP38, MANRS and Internet peering

"**BCP38**" -- Discussed also in another effort

Mutually **A**greed **N**orms for **R**outing **S**ecurity (MANRS)

- Define four concrete actions network operators should implement
- Coordination
  - Keep your contacts updated
- Validation
  - Route objects, RPKI, BGPSec
- Anti-spoofing
  - uRPF
  - Filtering on external Interfaces facing external suppliers
  - Drop inbound Traffic with a src IP claiming to be from your networks / private networks.
  - Drop outbound Traffic with a src IP address that is not in your Public IP network range.
- Build a visible community of security-minded operators
- Valuable resource: The Routing Manifesto

Introducing VXLAN in your network

vxlan - the **V**irtual e**X**tensible **L**ocal **A**rea **N**etwork tunnel interface

- Pushes layer 2 network (Ethernet frames) over layer 3 (IP) tunnels
  - 24-bit *vnetid* (vs max 4k VLANs)

- Has *no* built in security
- Intended for '*trusted*' (Datacenter, inter-hypervisor) environments
  - Otherwise, consider transport over IPSEC.

- Default transport over **UDP 4789** (aka **vxlan**)
  - make sure that traffic passes between endpoints

Introducing VXLAN in your network

```
# ifconfig vxlan0 tunnel  192.168.100.101  192.168.200.201  vnetid 17
# ifconfig vxlan0  10.11.12.100/24
```

```
# ifconfig vxlan0 tunnel  192.168.200.201  192.168.100.101   vnetid 17
# ifconfig vxlan0  10.11.12.101/24
```

```
table <vxendpoints> { 192.168.200.201 192.168.200.204 }
pass from <vxendpoints> to port vxlan
```

Buy Reyk a beer.

Readable and maintainable toolsets

- **Macros**
  - descriptive names, keep uniform
- **Tables**
  - descriptive names
  - consider daemon/scripting interface
- **Interface groups**
  - you know egress already
  - make your own and filter on them
- **Anchors**
  - group rules by common criteria
  - tagging
  - interface or group
- Service names vs port numbers
- **Comments** - yes, you **will** forget why this was a good idea

Useful 3rd party packages (ports) for OpenBSD

OpenBSD base operating system can be supplimented by the following packages and features:

- pftop - a curses-based utility for real-time display of active states and rules for pf. It is a cross between top and pfctl -sr and pfctl -ss.
  - pftop can be installed with the following command
- `pkg_add pftop`
  - nsh **n**etwork **sh**ell
  - nsh can be installed with the following command
- `pkg_add nsh`

Now let's add wireless

- Wireless used to be hard, (WPA in particular), now it's 'just another interface'
- 802.11* support in OpenBSD has a,b,g,n, ac only in some drivers (bwfm(4), iwx(4))
- Not all drivers support hostap
  - check man pages before buying kit for access point use
- Optionally setup with commercial APs for radio part
  - do DHCP, filtering, authentication and so forth from OpenBSD

Questions ?

Troubleshooting

Section 6

"It's all your fault. Until you track down and fix the root cause."

Troubleshooting 101: ICMP(v6)

- ICMP: **I**nternet **C**ontrol **M**essage **P**rotocol
- The *ping of death* scare is almost over, let's enable ping:

```
icmp_types = "{ echoreq, unreach }"

pass inet proto icmp all icmp-type $icmp_types keep state

pass inet proto icmp from $localnet icmp-type $icmp_types

pass inet proto icmp to $ext_if icmp-type $icmp_types

pass inet6 proto icmp6 from $localnet icmp6-type $icmp6_types

pass inet6 proto icmp6 to $ext_if icmp6-type $icmp6_types
```

- **echoreq**: lets ping do its thing
- **unreach**: lets you do *path MTU discovery* (PMTUD)

Troubleshooting 101: ICMP not all messages created equal!

- Some ICMP messages are frivolous:
- Type 4 — Source Quench (Deprecated)
- Type 5 — Redirect (if you need this you are doing it wrong)
- Type 6 — Alternate Host Address (Deprecated)
- Type 13 — Timestamp
- Type 14 — Timestamp Reply
- Type 15 — Information Request (Deprecated)
- Type 16 — Information Reply (Deprecated)
- Type 17 — Address Mask Request (Deprecated)
- Type 18 — Address Mask Reply (Deprecated)

Troubleshooting 101: basic diagnostics built in in OpenBSD base

Basic IP connectivity

```
ping <host>
traceroute <host>
telnet <host> <tcp port>
tcpdump -i <interfacename>
```

- netcat command - nc
- nc -z **<host> <startport>-<endport>**

Troubleshooting 101: not all pings were created equal!

ping packets can be manipulated in many ways

- set a specific source address (useful for routers)
- ping [-I **<src>**address>] **<destaddress>**
- -set ping packet size (MTU / Path discovery Issues)
- ping [-s **<packet-size>**] **<destaddress>**
- set do not fragment bit (MTU / Fragmentation issues)
- ping [-D] <destaddress>

Troubleshooting 101: basic diagnostics built in in OpenBSD base

DNS functionality basic commands

`ping <hostdnsname>`
`telnet **<hostdnsname> <tcp port>**`

- in depth dns testing
- `**dig** <hostdnsname>`
- `nslookup **<hostdnsname>**`

Troubleshooting 101: diagnostic tools available in ports

IP connectivity

- mtr (not the gtk version)
- mtr **<host>**
- tcp traceroute
- tcptraceroute **<host>**

Troubleshooting 101: Statistics

- Statistics can be had with **pfctl -s info**

For statistics (bytes/packets passed per rule) attach *labels* per rule

```
pass log proto { tcp, udp } to $emailserver port smtp label "mail-in"
pass log proto { tcp, udp } from $emailserver to port smtp label "mail-out"
$ doas pfctl -vs rules
pass inet proto tcp from any to 192.0.2.225 port = smtp flags S/SA keep state
label "mail-in"
[ Evaluations: 1664158 Packets: 1601986 Bytes: 763762591 States: 0 ]
[ Inserted: uid 0 pid 24490 ]
pass inet proto tcp from 192.0.2.225 to any port = smtp flags S/SA keep state
label "mail-out"
[ Evaluations: 2814933 Packets: 2711211 Bytes: 492510664 States: 0 ]
[ Inserted: uid 0 pid 24490 ]
```

Troubleshooting 101: Statistics

- If you need to pass the data to a script
  - Or a database
  - A graphing engine

```
$ doas pfctl -zvsl
mail-in 1664158 1601986 763762591 887895 682427415 714091 81335176
mail-out 2814933 2711211 492510664 1407278 239776267 1303933 252734397
```

Troubleshooting 101: log to pflog

Rules with the **log** keyword log packet data to the pflog device(s)

```
# log blocked packets
block log(all)
# logs initial packet of matching connections:
pass log proto tcp to port ssh
# logs all matching packets:
pass log(all) proto tcp to port ssh log(all)
# logs matches on this and all succeeding rules
pass log(matches) proto tcp to port ssh
# logs all packets matches on this and all succeeding rules
pass log(all, matches) proto tcp to port ssh
match log(all, matches) # log *everything*
```

Troubleshooting 101: tcpdump, read from pflog

- tcpdump is your friend
- Let it loose on the pflog device:

```
$ doas tcpdump -n -e -ttt -i pflog0
tcpdump: WARNING: snaplen raised from 116 to 160
tcpdump: listening on pflog0, link-type PFLOG
May 29 21:06:27.165561 rule def/(match) pass in on bge1: 192.168.103.126.15526 >
213.187.179.198.22: . ack 2951513182 win 16332 (DF) [tos 0x10]
May 29 21:06:27.166934 rule 16/(match) pass in on bge0: 158.36.191.135.22 >
213.187.179.198.59516: . ack 1734404306 win 64800 [tos 0x8]
May 29 21:06:27.166939 rule 2/(match) match in on bge0: 158.36.191.135.22 >
213.187.179.198.59516: . ack 1 win 64800 [tos 0x8]
May 29 21:06:27.168340 rule def/(match) pass out on bge1: 213.187.179.198.22 >
192.168.103.126.15526: P 69:153(84) ack 0 win 17520 [tos 0x10]
May 29 21:06:27.169150 rule def/(match) pass out on bge1: 213.187.179.198.22 >
192.168.103.126.15526: P 153:333(180) ack 0 win 17520 [tos 0x10]
May 29 21:06:27.169265 rule def/(match) pass out on bge1: 213.187.179.198.22 >
```

- **NB** rule number, matches your *loaded* rule set

Troubleshooting 101: Hitting and avoiding limits

- On busy systems, you may need to raise limits from default values
- Check with:

```
$ doas pfctl -s info
```

- versus the output of **pfctl -s memory** and **pfctl -s timeouts**
- You may need to bump up from defaults:

```
# increase state limit from 10'000 states on busy systems
set limit states 100000
# increase no of source nodes
set limit src-nodes 100000
```

Troubleshooting 101: quick flow analysis using iftop (package)

- Sometimes we need to check what traffic is flowing on a given interface
- iftop ncurses package can be installed and allows visibility on top traffic flows on an interface
- iftop allows that quick diagnostics check without needing a full flow anlaysis infrastructure
- iftop ncurses package can be installed on any OpenBSD system using the following command

```
pkg_add iftop
```

Troubleshooting 101: using iftop (package) example

- Checking flow on the first intel 1Gbs interface on a router
- iftop -i em0
- command output

```
        1.91Mb      3.81Mb      5.72Mb      7.63Mb    9.54Mb
                                                                
10.0.2.15           => di-in-f190.1e100.net     23.5Kb  11.8Kb  2.95Kb
                <=                  3.22Mb   805Kb   201Kb
10.0.2.15           => dj-in-f106.1e100.net      0b   18.1Kb  6.53Kb
                <=                    0b    708Kb   218Kb
10.0.2.15           => dj-in-f94.1e100.net       0b   4.41Kb  1.10Kb
                <=                    0b   57.5Kb  14.4Kb
10.0.2.15           => dj-in-f100.1e100.net      0b   4.14Kb  1.03Kb
                <=                    0b   43.3Kb  10.8Kb
10.0.2.15           => dh-in-f95.1e100.net       5.69Kb  4.36Kb  1.60Kb
                <=                  133Kb  29.5Kb  8.06Kb
10.0.2.15           => dj-in-f119.1e100.net      0b   3.26Kb   834b
                <=                    0b   19.6Kb  4.91Kb
10.0.2.15           => ig-in-f94.1e100.net       0b   4.66Kb  1.16Kb
                <=                    0b   12.8Kb  3.21Kb
10.0.2.15           => dj-in-f148.1e100.net      12.5Kb  2.49Kb   638b
                <=                  49.9Kb  9.99Kb  2.50Kb
10.0.2.15           => dg-in-f102.1e100.net      0b   6.57Kb  1.64Kb
                <=                    0b   5.51Kb  1.38Kb
                                                                
TX:             cum:   832KB   peak:   89.6Kb  rates:   51.7Kb  62.4Kb  20.1Kb
```

Troubleshooting 101: netflow aka pflow (IPFIX)

- Records TCP/IP *flow* metadata
  - srcIP
  - dstIP
  - (srcPort, dstPort)
  - startTime
  - endTime
  - Packets
  - Bytes
- OpenBSD has the pflow(4) virtual network interface
  - which generates the datagrams from the state table
- Useful for network monitoring, DDoS protection, etc.

Troubleshooting 101: netflow setup

- Set up a *sensor*:

```
$ cat /etc/hostname.pflow0
flowsrc 192.168.103.1 flowdst 192.168.103.252:9995
pflowproto 10
```

- Then configure your *collector* at the **flowdst** IP address for analysis and network overlordship.

- Lots of collector options available in ports: nfsen, flow-tools, pmacct, FastNetMon and others.
- More info:
  - Michael W. Lucas: Network Flow Analysis
  - and Peter N. M. Hansteen: Yes, You Too Can Be An Evil Network Overlord - On The Cheap With OpenBSD, pflow And nfsen.

Flow Analyser example Fastnetmon

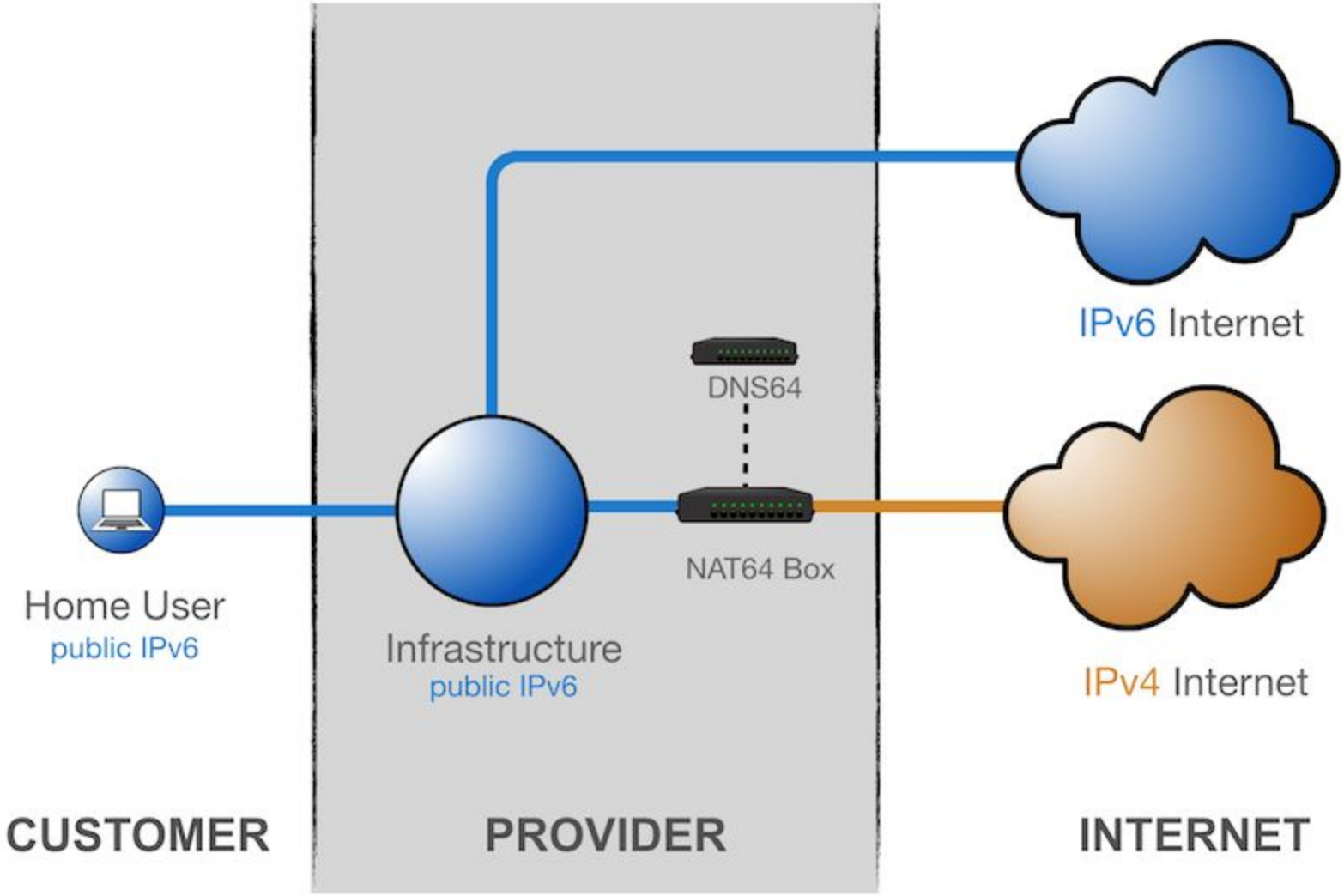
- Example of a typcial flow anlayser software fastnetmon:
  - User can view FastNetMon statistics via the CLI client fastnetmon_client
  - # fastnetmon_client
  - FastNetMon 1.1.7 master git- Try Advanced edition: https://fastnetmon.com
  - IPs ordered by: packets
  - Incoming traffic 1505664 pps 15397 mbps 85 flows
  - 37.203.[redacted] 59184 pps 485 mbps 0 flows
  - 37.203.[redacted] 45040 pps 504 mbps 0 flows
  - 37.203.[redacted] 26924 pps 270 mbps 0 flows
  - 185.55.[redacted] 24211 pps 240 mbps 0 flows
  - 5.134.[redacted] 23872 pps 290 mbps 0 flows
  - 45.11.[redacted] 23634 pps 250 mbps 0 flows
  - 185.55.[redacted] 22451 pps 255 mbps 0 flows
  - 45.11.[redacted] 20943 pps 254 mbps 0 flows
  - 185.55.[redacted] 20298 pps 246 mbps 0 flows
  - 5.134.[redacted] 20188 pps 236 mbps 0 flows
- With FastNetMon one can implement mitigations based on tresholds
  - Packets per second pps
  - Bandwidth per second Mbps

Exercise 5

NAT64

NAT64

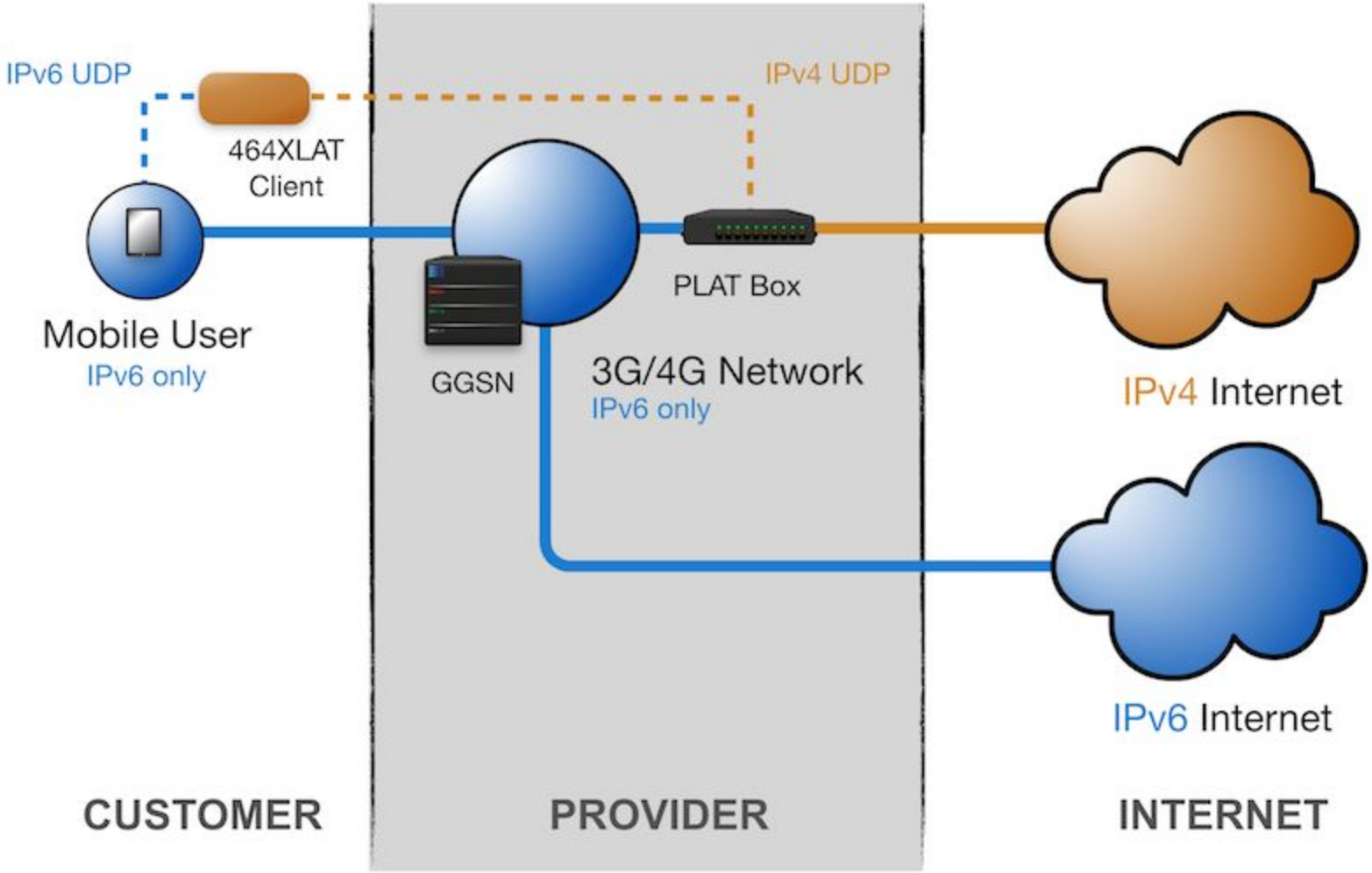
- Single-stack clients will only have IPv6
- Translator box will strip all headers and replace them with IPv4
- Requires some DNS “magic”
  - Capture responses and replace A with AAAA
  - Response is crafted based on target IPv4 address
- Usually implies address sharing on IPv4


IPv6 Internet
DNS64
NAT64 Box
Home User
public IPv6
Infrastructure
public IPv6
IPv4 Internet
CUSTOMER
PROVIDER
INTERNET

464-XLAT

- Extension to NAT64 to access IPv4-only applications (like Skype or Whatsapp)

- Handset pretends there is an IPv4 address (CLAT) and sends IPv4 packets in UDP over IPv6


IPv6 UDP
IPv4 UDP
464XLAT
Client
PLAT Box
Mobile User
IPv6 only
GGSN
3G/4G Network
IPv6 only
IPv4 Internet
IPv6 Internet
CUSTOMER
PROVIDER
INTERNET

Exercise 6 - Goals

- Define the translating prefix
  - 64:ff9b::/96 is reserved by IETF
- Add NAT64 configuration to PF
- Remove IPv4 from the internal network
- Configure DNS64 on unbound

Exercise 6

- *pf.conf*

```
pass in quick on $int_if inet6 from any to 64:ff9b::/96 \
  af-to inet from (egress:0) keep state
```

- *unbound.conf*

```
module-config: "dns64 validator iterator"
dns64-prefix: 64:FF9B::/96
```

and it's done!

Questions ?

Last chance...

or questions@pftutorial.net

Web accessible resources

OpenBSD website and documentation

http://www.openbsd.org/ The official OpenBSD website – to donate: http://www.openbsd.org/donations.html and please do donate, corporates may prefer https://www.openbsdfoundation.org/ - a Canadian non-profit

The PF User Guide on the OpenBSD web site

OpenBSD online man pages

Note: You can convert the man page of pf.conf to PDF for reading in your favourite reader with the command:

```
man -T pdf pf.conf > pf.conf.pdf
```

Resources

Books / e-Books

Michael W Lucas: Absolute OpenBSD, 2nd ed.

Peter N. M. Hansteen: The Book of PF, 3rd ed.

Elizabeth D. Zwicky, Simon Cooper, D. Brent Chapman Building Internet Firewalls, 2nd ed.

Blogs

http://undeadly.org/ - The OpenBSD Journal news site

http://bsdly.blogspot.com/ - Peter's rants^H^H^H^H^Hblog posts

http://www.tedunangst.com/flak/ tedu@ on developments


The End!
Край
Y Diwedd
Fí
Finis
النهاية
Соңы
վերջ
Liðugt
Ende
Finvezh
Кінець
Konec
Kraj
Ënn
Fund
پایان
Kpaj
Lõpp
Beigas
Vége
Son
An Críoch
הסוף
Endir
Fine
Sfârşit
Fin
Τέλος
Einde
Конец
Slut
Slutt
დასასრული
Pabaiga
Tmiem
Koniec
Amaia
Loppu
Fim


English
The End!
Bulgarian
Край
(Kraj)
Welsh
Y Diwedd
Arabic
النهاية
(An-Nahaya)
Kazakh
Соңы
Armenian
վերջ
(Verj)
Catalan
Fí
Latin
Finis
Faroese
Liðugt
German
Ende
Breton
Finvezh
Ukrainian
Кінець
(Kinec))
Czech
Konec
Letzeburgisch(LUX)
Albanian
Fund
Persian
پایان
(Payan)
Croatian
Kraj
Ënn
Turkish
Son
Serbian
Kpaj
(Kraj)
Estonian
Lõpp
Latvian
Beigas
Hungarian
Vége
Irish
An Críoch
Hebrew
הסוף
(Ha-sof))
Icelandic
Endir
Romanian
Sfârşit
Italian
Fine
French
Fin
Greek
Τέλος
(Telos))
Dutch
Einde
Belorussian
Канец
(Kanec))
Russian
Конец
(Konec))
Swedish, Danish
Slut
Norwegian
Slutt
Georgian
დასასრული
(Dasasruli))
Lithuanian
Pabaiga
Maltese
Tmiem
Polish, Slovak
Koniec
Basque
Amaia
Finnish
Loppu
Portuguese
Fim