Packet Filter (pf)

An Extended Introduction

Max Laier

http://people.freebsd.org/ mlaier/

FreeBSD.ORG

http://www.FreeBSD.ORG/

pf - History

How it started

- OpenBSD's desire to improve it's firewall capabilites

How it started

- OpenBSD's desire to improve it's firewall capabilites
- License issues with ipfilter (ipf)

How it started

- OpenBSD's desire to improve it's firewall capabilites
- License issues with ipfilter (ipf)
- Solution:
  - Rewrite from scratch
  - At least 3 competting solutions

How it started

- OpenBSD's desire to improve it's firewall capabilites
- License issues with ipfilter (ipf)
- Solution:
  - Rewrite from scratch
  - At least 3 competting solutions
  - Daniel Hartmeier's pf choosen due to:
    - ipf-compatible syntax (*almost*)
    - Simplicity
    - Ease of extension

How it started

- OpenBSD's desire to improve it's firewall capabilites
- License issues with ipfilter (ipf)
- Solution:
  - Rewrite from scratch
  - At least 3 competting solutions
  - Daniel Hartmeier's pf choosen due to:
    - ipf-compatible syntax (*almost*)
    - Simplicity
    - Ease of extension
  - Ideas from the other approaches were merged

How it started

- OpenBSD's desire to improve it's firewall capabilites
- License issues with ipfilter (ipf)
- Solution:
  - Rewrite from scratch
  - At least 3 competting solutions
  - Daniel Hartmeier's pf choosen due to:
    - ipf-compatible syntax (*almost*)
    - Simplicity
    - Ease of extension
  - Ideas from the other approaches were merged
- OpenBSD 3.0 (*December 1, 2001*) first release with pf

Ports

- **November 5, 2002** – *Joel Wilsson* to NetBSD
- **March 25, 2003** – *Pyun YongHyeon* to FreeBSD
- **June 27, 2003** – KAME integrates source from OpenBSD current
- **February 26, 2004** – FreeBSD integrates the port into the base system
- **June 22, 2004** – NetBSD integrates the port into the base system
- Ongoing work to port to DragonFlyBSD

Ports

- **November 5, 2002** – *Joel Wilsson* to NetBSD
- **March 25, 2003** – *Pyun YongHyeon* to FreeBSD
- **June 27, 2003** – KAME integrates source from OpenBSD current
- **February 26, 2004** – FreeBSD integrates the port into the base system
- **June 22, 2004** – NetBSD integrates the port into the base system
- Ongoing work to port to DragonFlyBSD

- **Ports might behave differently!**
  - FreeBSD 5-STABLE will be compatible to OpenBSD 3.5
  - NetBSD is level with OpenBSD 3.5 at the moment
  - KAME seems to track OpenBSD-current (on and off)

Releases

- OpenBSD makes a relase aprox. every 6 month
- pf still gains a **lot** of features with every release
- Overlapping release cycles might cause considerable deltas between OpenBSD and the various ports
- Most of the porting efforts follow OpenBSD's lead when it comes to new features, but divergence might occure where required.

Releases

- OpenBSD makes a relase aprox. every 6 month
- pf still gains a **lot** of features with every release
- Overlapping release cycles might cause considerable deltas between OpenBSD and the various ports
- Most of the porting efforts follow OpenBSD's lead when it comes to new features, but divergence might occure where required.
- **Check the documentation!**

pf - Features

Basics

- Ruleset based configuration
  - Anchors allow nesting [a]
  - Tables allow dynamic adaption of rules [a]
  - Interface address management [b]
- Default (logical) order required
  1. Macro and table definitions
  2. (Global) Options
  3. Traffic Normalization (*scrub*)
  4. Network address translation (*NAT*) and redirection
  5. Filtering rules
- **Last** matching rule wins
  - This can be changed with the ‘quick’ keyword
- Macros allow easier reading/writing and understanding of rules

[a] introduced in OpenBSD 3.3

[b] introduced in OpenBSD 3.2

Basics

- Ruleset based configuration
  - Anchors allow nesting [a]
  - Tables allow dynamic adaption of rules [a]
  - Interface address management [b]
- Default (logical) order required
  1. Macro and table definitions
  2. (Global) Options
  3. Traffic Normalization (*scrub*)
  4. Network address translation (*NAT*) and redirection
  5. Filtering rules
- **Last** matching rule wins
  - This can be changed with the ‘quick’ keyword
- Macros allow easier reading/writing and understanding of rules

[a] introduced in OpenBSD 3.3
[b] introduced in OpenBSD 3.2

Basics

- Ruleset based configuration
  - Anchors allow nesting [a]
  - Tables allow dynamic adaption of rules [a]
  - Interface address management [b]
- Default (logical) order required
  1. Macro and table definitions
  2. (Global) Options
  3. Traffic Normalization (*scrub*)
  4. Network address translation (*NAT*) and redirection
  5. Filtering rules
- **Last** matching rule wins
  - This can be changed with the ‘quick’ keyword
- Macros allow easier reading/writing and understanding of rules

[a] introduced in OpenBSD 3.3
[b] introduced in OpenBSD 3.2

Basics

- Ruleset based configuration
  - Anchors allow nesting [a]
  - Tables allow dynamic adaption of rules [a]
  - Interface address management [b]
- Default (logical) order required
  1. Macro and table definitions
  2. (Global) Options
  3. Traffic Normalization (*scrub*)
  4. Network address translation (*NAT*) and redirection
  5. Filtering rules
- **Last** matching rule wins
  - This can be changed with the ‘quick’ keyword
- Macros allow easier reading/writing and understanding of rules

[a] introduced in OpenBSD 3.3
[b] introduced in OpenBSD 3.2

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Traffic Normalization

Normalization or "scrubbing" summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Traffic Normalization

Normalization or “scrubbing” summarises a couple of packet sanity checks to protect against evildoer and information leaks and some rewrites to improve security for weak(er) peers

- IP normalization
- IP fragment reassembly
- Random IP ID rewrite
- TCP normalization
  - TCP reassembly
  - Illegal flag combinations (nmap)
  - TCP options
  - PAWS (Protection Against Wrapped Sequence Numbers)
- Enforce minimum TTL

Scrubbing also helps to hide peer uptimes and thus can prevent NAT detection

NAT and redirection

- BINAT – bidirectional address translation
- NAT – source address translation
  - static port – prevents sourceport rewrite
- RDR – destination address translation

NAT and redirection

- BINAT – bidirectional address translation
- NAT – source address translation
  - static port – prevents sourceport rewrite
- RDR – destination address translation

NAT and especially RDR can be combined with address loadbalancing. The following disciplines to choose an address from the pool are available:

- random
- round-robin
- bitmask
- source-hash

NAT and redirection

- BINAT – bidirectional address translation
- NAT – source address translation
  - static port – prevents sourceport rewrite
- RDR – destination address translation

NAT and especially RDR can be combined with address loadbalancing. The following disciplines to choose an address from the pool are available:

- random
- round-robin
- bitmask
- source-hash

The **sticky-address**-option can be applied to *random* and *round-robin* to improve the redirection mapping

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- **In**comming
- **Out**going

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- IPv4 “inet”
- IPv6 “inet6”

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- "any"
- "no-route"
- "self"
- IPv4 dotted quad (127.0.0.1)
- IPv6 coloned hex (2001:200:0:8002:203:47ff:fea5:3085)
- CIDR compatible since OpenBSD 3.3
- Hostname (freebsd.org)
- Netmasks can be applied
- pf can (dynamically) extract addresses from a local interface

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- (IP)Protocol number
- Protocol name/alias (etc/protocols)

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- ‘lowdelay’
- ‘throughput’
- ‘reliability’
- Explicit hex-code

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- Disallow (default)
- Allow

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- Type
- Code

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- Source/Destination port(-range)
- Flags
- Socket credentials (‘owner’ of the sending/receiving application)
- *OS type* (passiv fingerprinting of the initial SYN packet)

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- Source/Destination port(-range)
- Socket credentials (‘owner’ of the sending/receiving application)

Filtering - Attributes

pf is an IP-level (OSI Layer 3) firewall, but also keeps track of TCP and UDP (OSI Layer 4) . . .

- Interface
- Direction
- Address family
- Source/Destination IP(-range)
- Protocol
- ToS
- IP options
- ICMP/ICMP6
- TCP
- UDP

- Source/Destination port(-range)
- Socket credentials ('owner' of the sending/receiving application)

pf filters on everything and provides understandable syntax

Filtering - Stateful

States are fast:

- Indexed in a reb-black tree => fast lookup (O(lg(n)))
- State lookup is much faster than ruleset evaluation

Filtering - Stateful

States are fast:

- Indexed in a reb-black tree => fast lookup (O(lg(n)))
- State lookup is much faster than ruleset evaluation

States increase security:

- Control who initiates a connection
- Apply additional sanity checks to TCP connections
  - Segments must be in the window
  - Restes must be on the edge of the window
  - Window scaling available
  - pf must ‘see’ the initial handshake (flags S/SA)
- Help to gather accounting information (pfflowd e.g.)
- Additional peer protection against ISN prediction attacks available with *modulate state* and *synproxy*

Filtering - Stateful

States are fast:

- Indexed in a reb-black tree => fast lookup (O(lg(n)))
- State lookup is much faster than ruleset evaluation

States increase security:

- Control who initiates a connection
- Apply additional sanity checks to TCP connections
  - Segments must be in the window
  - Restes must be on the edge of the window
  - Window scaling available
  - pf must ‘see’ the initial handshake (flags S/SA)
- Help to gather accounting information (pfflowd e.g.)
- Additional peer protection against ISN prediction attacks available with *modulate state* and *synproxy*

States are also used to keep track of address translations.

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- 'Not' modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

- Can be changed with pfctl
- Can be read from a file (spam lists)

Filtering - Tables

Tables provide sophisticated means to manage large, sparse address lists.

- Can take hosts and networks
- Most specific match is returned
- ‘Not’ modifier
- Implemented as radix tree => fast lookups (as known from routing tables)
- 64Bit statistics counters for every table entry (easy accounting)

- Can be changed with pfctl
- Can be read from a file (spam lists)

Short example

pf - Advanced notes

Interesting options

- timeouts
  - Set fragment cache time
  - Set cleanup intervals
  - Finetune TCP handshake timeouts
  - Finetune UDP/ICMP/other state hold time
  - **adaptive.start adaptive.end**
    Used to scale timeouts according to current state load. As the total number of states increase, unused states die more quickly.
- limits
  - States
  - Fragements
  - Source-tracking nodes

- [pf - History](#)
- [pf - Features](#)
- [pf - Advanced notes](#)
  - ❖ Interesting options
  - ❖ Stateful I
  - ❖ Stateful II
  - ❖ Anchors
  - ❖ Misc
- [ALTQ (short)](#)
- [CARP](#)

Interesting options

- timeouts
  - Set fragment cache time
  - Set cleanup intervals
  - Finetune TCP handshake timeouts
  - Finetune UDP/ICMP/other state hold time
  - **adaptive.start adaptive.end**
    Used to scale timeouts according to current state load. As the total number of states increase, unused states die more quickly.
- limits
  - States
  - Fragements
  - Source-tracking nodes

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful I

Stateful rule options:

- Per-rule timeouts
- Per-rule limits
- **source-track**
  - Per-*client* limits
  - Maximum number of peers
  - Maximum connections per peer

Modulate state:

- Randomize the initial sequence numbers of a (TCP) connection
- Store original ISN the client in a state
- Rewrite packets of that connection using the *better* sequence numbers

Stateful II

Synproxy states:

- Gateway completes the 3-way handshake
- Gateway initiates connection with the destination
- Further traffic is a normal stateful connection
  Same mechanisms used as for modulate-state connections

Stateful II

Synproxy states:

- Gateway completes the 3-way handshake
- Gateway initiates connection with the destination
- Further traffic is a normal stateful connection
  Same mechanisms used as for modulate-state connections

Stateful II

Synproxy states:

- Gateway completes the 3-way handshake
- Gateway initiates connection with the destination
- Further traffic is a normal stateful connection
  Same mechanisms used as for modulate-state connections

Stateful II

Synproxy states:

- Gateway completes the 3-way handshake
- Gateway initiates connection with the destination
- Further traffic is a normal stateful connection
  Same mechanisms used as for modulate-state connections

Useful to protect *web*- and *DB*-server against SYN-flood attacks.

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be ‘limited’ to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be ‘limited’ to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be ‘limited’ to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be 'limited' to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be 'limited' to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be 'limited' to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition
  - **Plan**: per-jail rulesets managed by jail-root (in FreeBSD)

Anchors

- Structure rulesets
- Easily changeable
- Good for script actions
- Used by authpf
- Can be ‘limited’ to:
  - Direction (in/out)
  - Interface
  - Address family
  - Protocol
  - Host definition
  - **Plan**: per-jail rulesets managed by jail-root (in FreeBSD)
- **New in 3.6**: Recursive anchors

Misc

- Skip-steps
  - Jump over rules that cannot match
  - Optimize ruleset evaluation
  - **New in 3.6**: Ruleset optimization
- Tagging
  - Only with stateful rules
  - Allows to seperate classification and policy
  - Used to support layer 2 filtering
- OS Fingerprints
  - Base on p0f
  - Only for the initial SYN packet

Misc

- Skip-steps
  - Jump over rules that cannot match
  - Optimize ruleset evaluation
  - **New in 3.6**: Ruleset optimization
- Tagging
  - Only with stateful rules
  - Allows to seperate classification and policy
  - Used to support layer 2 filtering
- OS Fingerprints
  - Base on p0f
  - Only for the initial SYN packet

Misc

- Skip-steps
  - Jump over rules that cannot match
  - Optimize ruleset evaluation
  - **New in 3.6**: Ruleset optimization
- Tagging
  - Only with stateful rules
  - Allows to seperate classification and policy
  - Used to support layer 2 filtering
- OS Fingerprints
  - Base on p0f
  - Only for the initial SYN packet

Misc

- Skip-steps
  - Jump over rules that cannot match
  - Optimize ruleset evaluation
  - **New in 3.6**: Ruleset optimization
- Tagging
  - Only with stateful rules
  - Allows to seperate classification and policy
  - Used to support layer 2 filtering
- OS Fingerprints
  - Base on p0f
  - Only for the initial SYN packet
  - **This can be spoofed! Not a security tool**

ALTQ (short)

Basics

- ALTQ provides several disciplies to queue packets
- Can be used to control bandwidth allocation and packet priorization
- Most effective in front of a bandwidth bottleneck (i.e. on the uplink gateway)

Basics

- ALTQ provides several disciplies to queue packets
- Can be used to control bandwidth allocation and packet priorization
- Most effective in front of a bandwidth bottleneck (i.e. on the uplink gateway)

Basics

- ALTQ provides several disciplies to queue packets
- Can be used to control bandwidth allocation and packet priorization
- Most effective in front of a bandwidth bottleneck (i.e. on the uplink gateway)

Basics

- ALTQ provides several disciplies to queue packets
- Can be used to control bandwidth allocation and packet priorization
- Most effective in front of a bandwidth bottleneck (i.e. on the uplink gateway)

Detailated per-application evaluation is required to implement ALTQ

ALTQ and pf

- pf gained ALTQ linkage in OpenBSD 3.3
- Stateful pass rules assign thier traffic to a queue
- 'Lowdelay' and empty ACKs can be treated specially
- Queues are setup from the ruleset directly

ALTQ and pf

- pf gained ALTQ linkage in OpenBSD 3.3
- Stateful pass rules assign thier traffic to a queue
- 'Lowdelay' and empty ACKs can be treated specially
- Queues are setup from the ruleset directly

ALTQ and pf

- pf gained ALTQ linkage in OpenBSD 3.3
- Stateful pass rules assign thier traffic to a queue
- ‘Lowdelay’ and empty ACKs can be treated specially
- Queues are setup from the ruleset directly

ALTQ and pf

- pf gained ALTQ linkage in OpenBSD 3.3
- Stateful pass rules assign thier traffic to a queue
- ‘Lowdelay’ and empty ACKs can be treated specially
- Queues are setup from the ruleset directly

CARP

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - ✦ Most frequent advertiser becomes master
  - ✦ Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - ✦ HMAC covers logical addresses (which are not part of the advertisement)
  - ✦ **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - Most frequent advertiser becomes master
  - Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - HMAC covers logical addresses (which are not part of the advertisement)
  - **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - Most frequent advertiser becomes master
  - Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - HMAC covers logical addresses (which are not part of the advertisement)
  - **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - Most frequent advertiser becomes master
  - Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - HMAC covers logical addresses (which are not part of the advertisement)
  - **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - Most frequent advertiser becomes master
  - Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - HMAC covers logical addresses (which are not part of the advertisement)
  - **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

Basics

The Common Address Redundancy Protocol is a **free** replacement for VRRP

- Multiple hosts share a common MAC address (but only the master replys)
- The master advertises via multicast
- Variable advertisement intervals
  - ✦ Most frequent advertiser becomes master
  - ✦ Failover after (3 * advertisement interval)
- Advertisement protected by SHA1 HMAC
  - ✦ HMAC covers logical addresses (which are not part of the advertisement)
  - ✦ **Pending**: Replay protection
- Supports IPv4 **and** IPv6
- Supports layer 2 load balancing (ARP based)

CARP and pf(sync)

In order to support firewall/gateway redundancy every firewall must know about open connections (=states). The pfsync pseudo interface enables state synchronization:

- Each Firewall send out state changes via multicast
  - Insert
  - Update
  - Delete
- States have an unique ID
- Best effort synchronization
  Tends towards complete state view on every firewall
- Mechanisms to limit bandwidth and processing overhead
  - Updates contain only the changing information
  - Multiple updates are merged into one transfer
- pfsync prevents CARP preemption until states are synchronized

Example setup

CARP and pf(sync)

In order to support firewall/gateway redundancy every firewall must know about open connections (=states). The pfsync pseudo interface enables state synchronization:

- Each Firewall send out state changes via multicast
  - Insert
  - Update
  - Delete
- States have an unique ID
- Best effort synchronization
  Tends towards complete state view on every firewall
- Mechanisms to limit bandwidth and processing overhead
  - Updates contain only the changing information
  - Multiple updates are merged into one transfer
- pfsync prevents CARP preemption until states are synchronized

Example setup

CARP and pf(sync)

In order to support firewall/gateway redundancy every firewall must know about open connections (=states). The pfsync pseudo interface enables state synchronization:

- Each Firewall send out state changes via multicast
  - Insert
  - Update
  - Delete
- States have an unique ID
- Best effort synchronization
  Tends towards complete state view on every firewall
- Mechanisms to limit bandwidth and processing overhead
  - Updates contain only the changing information
  - Multiple updates are merged into one transfer
- pfsync prevents CARP preemption until states are synchronized

Example setup

CARP and pf(sync)

In order to support firewall/gateway redundancy every firewall must know about open connections (=states). The pfsync pseudo interface enables state synchronization:

- Each Firewall send out state changes via multicast
  - Insert
  - Update
  - Delete
- States have an unique ID
- Best effort synchronization
  Tends towards complete state view on every firewall
- Mechanisms to limit bandwidth and processing overhead
  - Updates contain only the changing information
  - Multiple updates are merged into one transfer
- pfsync prevents CARP preemption until states are synchronized

Example setup

CARP and pf(sync)

In order to support firewall/gateway redundancy every firewall must know about open connections (=states). The pfsync pseudo interface enables state synchronization:

- Each Firewall send out state changes via multicast
  - Insert
  - Update
  - Delete
- States have an unique ID
- Best effort synchronization
  Tends towards complete state view on every firewall
- Mechanisms to limit bandwidth and processing overhead
  - Updates contain only the changing information
  - Multiple updates are merged into one transfer
- pfsync prevents CARP preemption until states are synchronized

Example setup

More CARP

But CARP can not only do 'simple' firewall failover.

- Load balancing
  - Built-in ARP source-hash
  - pf and RDR/NAT
  - DNS round-robin
- Use it on servers
- ...

More complicated example

More CARP

But CARP can not only do ‘simple’ firewall failover.

- Load balancing
  - ✦ Built-in ARP source-hash
  - ✦ pf and RDR/NAT
  - ✦ DNS round-robin
- Use it on servers
- . . .

More complicated example

More CARP

But CARP can not only do ‘simple’ firewall failover.

- Load balancing
  - Built-in ARP source-hash
  - pf and RDR/NAT
  - DNS round-robin
- Use it on servers
- . . .

More complicated example

Questions

Links

These slides are at: http://people.freebsd.org/ mlaier/sucon.pdf

- OpenBSD
- OpenBSD pf
- FreeBSD
- FreeBSD pf
- NetBSD
- NetBSD pf
- DragonFly BSD
- DragonFly BSD
- p0f
- ALTQ additional reading
- CARP additional reading

Support

If you like pf, please

Buy OpenBSD CD-Sets! Support the developers!

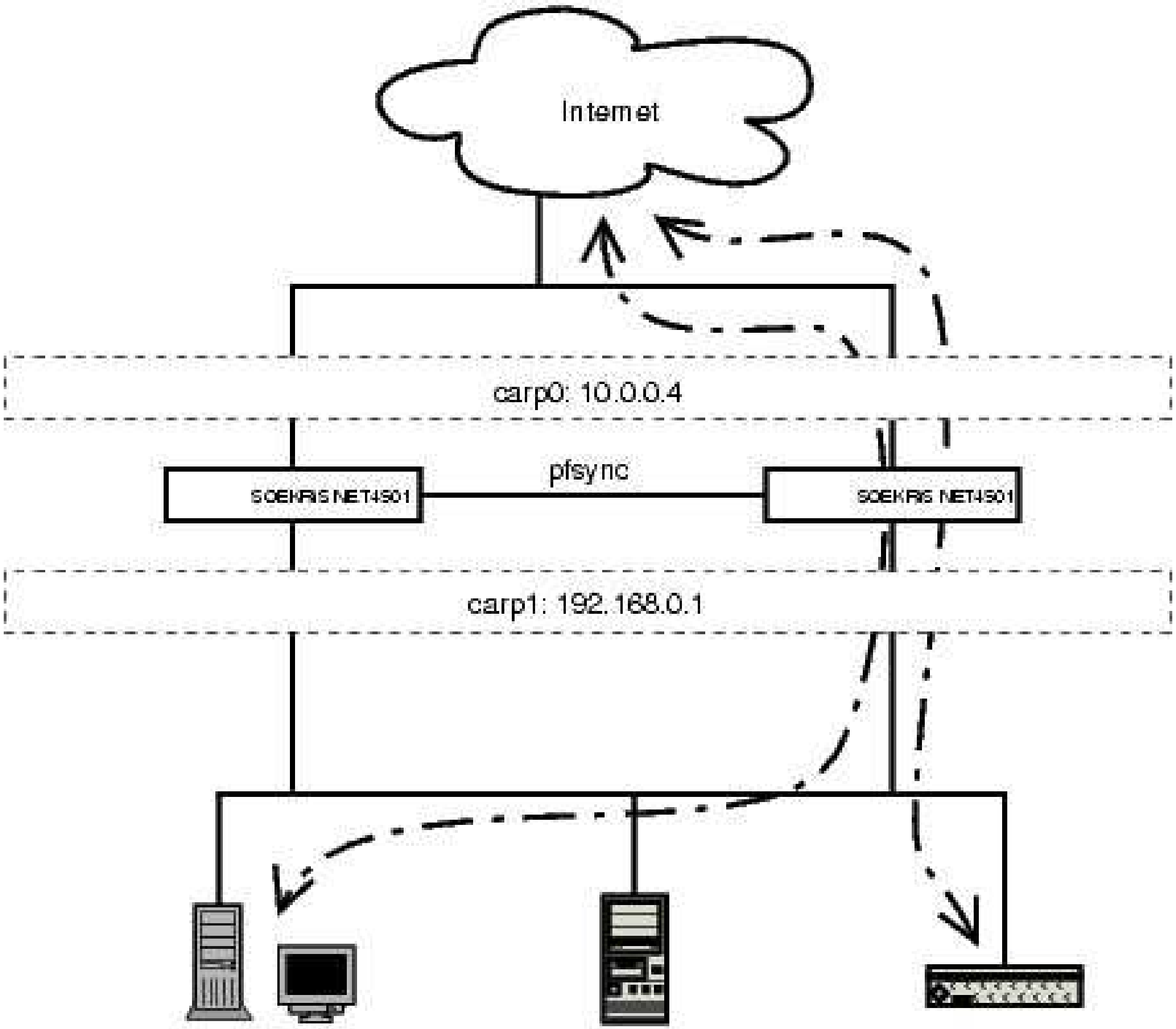
Examples

Table - Example


```
pf.test:
  table <test> persist { 10/8, 10.0.10/24, !10.0.10.1 }
  block all
  pass out to <test> keep state
# pfctl -ef pf.test
# ping -c 1 10.0.0.10 && ping -c 2 10.0.10.10
# ping -c 1 10.0.10.1
PING 10.0.10.1 (10.0.10.1): 56 data bytes
ping: sendto: Operation not permitted
^C
# pfctl -t test -vTshow
   10.0.0.0/8
        Cleared:     Wed Sep  1 19:57:38 2004
        In/Block:    [ Packets: 0                 Bytes: 0                 ]
        In/Pass:     [ Packets: 0                 Bytes: 0                 ]
        Out/Block:   [ Packets: 0                 Bytes: 0                 ]
        Out/Pass:    [ Packets: 1                 Bytes: 84                ]
   10.0.10.0/24
<...>
        Out/Pass:    [ Packets: 2                 Bytes: 168               ]
  !10.0.10.1
        Cleared:     Wed Sep  1 19:57:38 2004
        In/Block:    [ Packets: 0                 Bytes: 0                 ]
        In/Pass:     [ Packets: 0                 Bytes: 0                 ]
        Out/Block:   [ Packets: 0                 Bytes: 0                 ]
        Out/Pass:    [ Packets: 0                 Bytes: 0                 ]
```

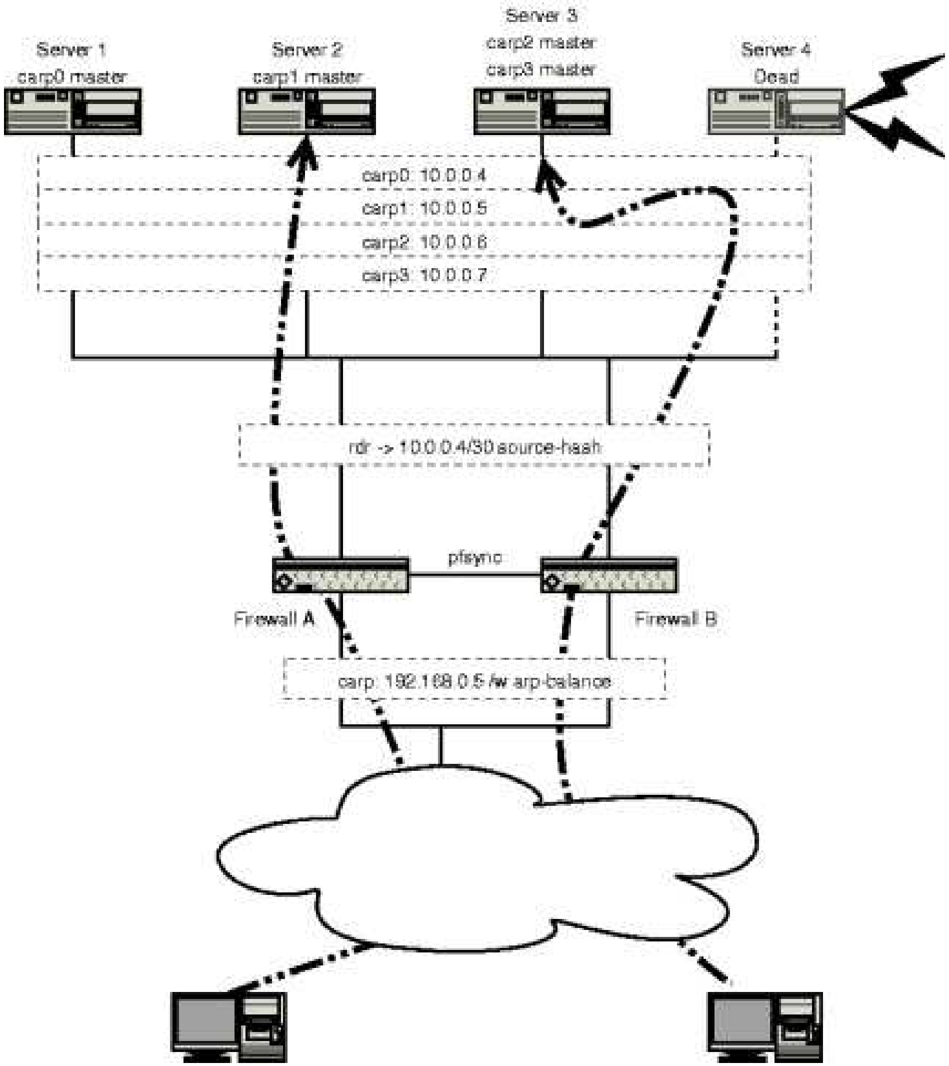
. . . back

CARP - Simple



. . . back

CARP - Advanced



. . . back