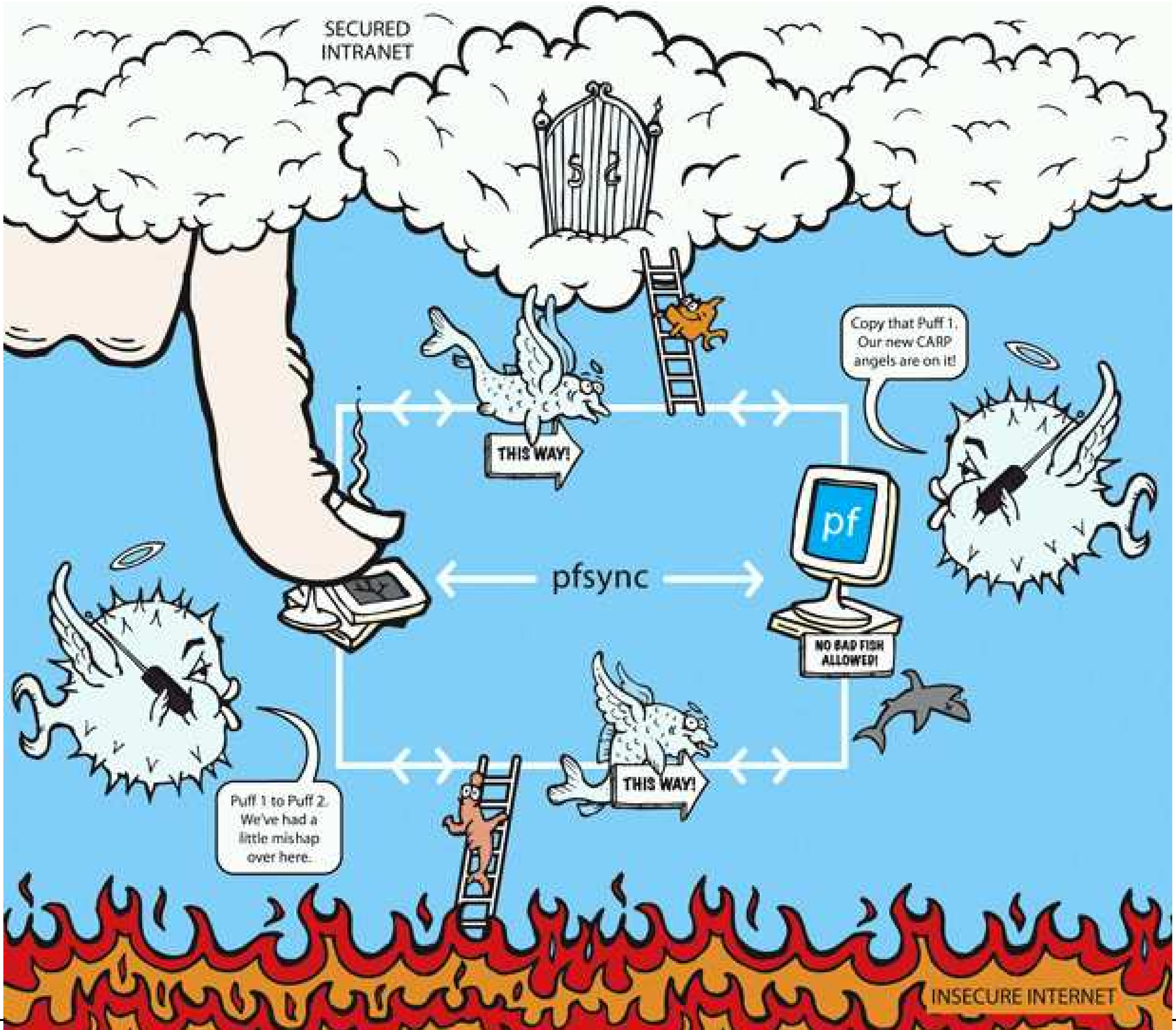
Robust Firewalls with OpenBSD and PF



Overview

- Design Philosophy (and what PF doesn’t do)
- The Basics
  - Normalisation
  - Filtering
  - Translation
- Advanced Toolkits
  - Denial of Service Mitigation
  - Firewall Redundancy
  - Load Balancing
- Comparison & Rant

Design Philosophy

- Free software
- Correct, readable code
- Secure, robust packet filtering
- Flexible but simple to use
- Good performance

Application filtering

- Two implementation options
  - Simple but simplistic
    - Trivial to defeat
    - False positives
  - Comprehensive but complex
    - Complexity == security risk
    - Too bloated for kernel
    - Extremely difficult to do correctly
- Solution: Userland proxy
  - No kernel bloat
  - Security risk of complex code can be contained
  - priviledge revocation/separation, chroot, etc.

User-level access control

- Like application filtering, this should be handled in userland
  - e.g. authpf
    - authentication and session timeout handled by ssh
    - modifies ruleset or table

Normalization (scrub)

- Sanitizes packet content to remove ambiguity:
  - IP fragment reassembly
  - IP normalisation
  - IPID randomisation
  - TCP normalisation
  - Illegal flag combinations
  - TCP options
  - PAWS (Protect Against Wrapped Sequence Numbers)
  - Enforce minimum TTL

Filterable Attributes

Source/destination address
Interface
Direction
Address family
Protocol
TOS
Fragments
IP options
Tagging
Route
ICMP code and type (ICMP)
User/group (TCP and UDP)
TCP flags (TCP)
Source OS (TCP)
Source/destination port (TCP and UDP)

OS Fingerprints

- Source OS only
- Looks at initial TCP packet
- Based on p0f, by lcamtuf@coredump.cx
- Can filter by general OS or specific version/patchlevel

- Can be spoofed
    - A policy tool, not a security tool

Tagging

- Rules can apply a named tag to a packet
- Only one tag per packet
- Pass rules with tagging must be stateful
- Subsequent rules can match on that tag
- Bridge code can also tag packets

- Allows the separation of classification and policy

Stateful Rules

- States indexed in a red-black tree
    - State searches are faster than rule lookup
- States increase security
    - Can control who initiates a connection
    - TCP segments must be within window
    - reset must be on edge of window

Tables

Tables provide a mechanism for increasing the performance and flexibility of rules with large numbers of source or destination addresses.

- Implemented as radix tree
  - Very fast lookups
- Bytes/packet counters for each table entry
- Can be loaded multiple ways
  - In pf.conf
  - From a file
  - On the command line with pfctl

Anchors

An anchor is a container that can hold rules, address tables, and other anchors.

- Placeholder for rules to be loaded later
- Changing anchor does not change main ruleset
- Can be nested

- Used by tools such as authpf to dynamically modify the ruleset

Translation

- nat - source address translation
- rdr - destination address translation
- binat - bidirectional address translation

Solving real world problems

Denial of Service Attack Mitigation

- Caveat: very difficult to combat bandwidth-based DDoS

- Techniques include:
  - synproxy
  - Adaptive Timeouts
  - max-src-states and max-src-nodes
  - max-src-conn and max-src-conn-rate
  - Input queue congestion handling
  - ALTQ

synproxy

- pf completes the 3 way handshake
- Does 3 way handshake with destination
- Remaining traffic is a normal stateful connection
    - (with modulated sequence numbers)

Adaptive Timeouts

- Scales timeouts as the total number of states increases
    - Unused states die more quickly

max-src-states and max-src-nodes

- Works with ’source-tracking’
- states tracked by source IP
  - max-src-states limits states per source
  - max-src-nodes limits number of sources

max-src-conn and max-src-conn-rate

The 3-way handshake ensures the source is not spoofed...

so we introduce per-source limits on TCP connections

completing the 3-way handshake

- **`max-src-conn 10`**
  - Number of open connections
- **`max-src-conn-rate 10/60`**
  - Rate of new connections (connections over time)
  - Estimate calculated on a moving average
- **`’overload <bad_guys> flush global’`**
  - Optional automatic response to the limit
  - Add the offending address to a table
  - Kill existing connections from the source

Input queue congestion handling

- Under some dDoS attacks CPU is overloaded
  - Input queue fills up
  - Machine becomes unresponsive
- When input queue is full stop evaluating ruleset
  - stateful packets are passed
  - stateless packets dropped unconditionally

- Packets would have gotten dropped anyways
- Machine stays responsive

ALTQ

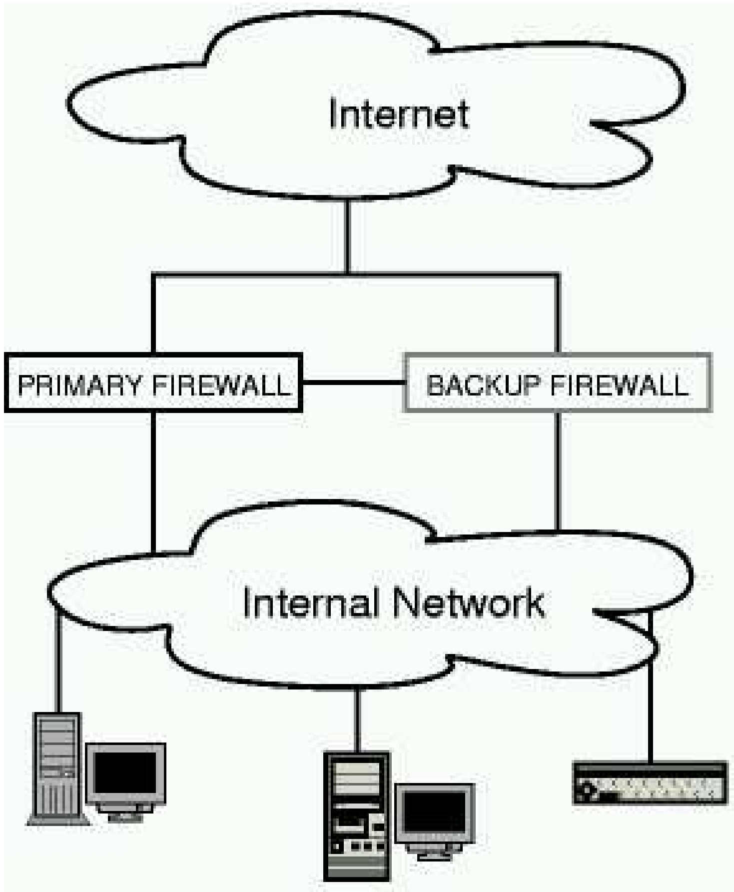
- Bandwidth shaping
- Can filter traffic based on filter attributes
- Works only with stateful rules
- Multiple queueing disciplines supported

- Most effective in front of bandwidth bottleneck
    - eg at upstream ISP(s)

Combination of Techniques

- Individual features become powerful weapons when used together:
  - synproxy + max-states + adaptive timeouts
  - synproxy + max-src-conn-rate
  - ALTQ + OS Fingerprinting

Firewall Redundancy



pfsync

The pfsync protocol synchronises state information between multiple firewalls.

- Each firewall sends out state changes via multicast
- Best effort - Systems tend towards complete synchronisation
- Some mechanisms to limit packets (and thus interrupts)
- pfsync is architecture independant

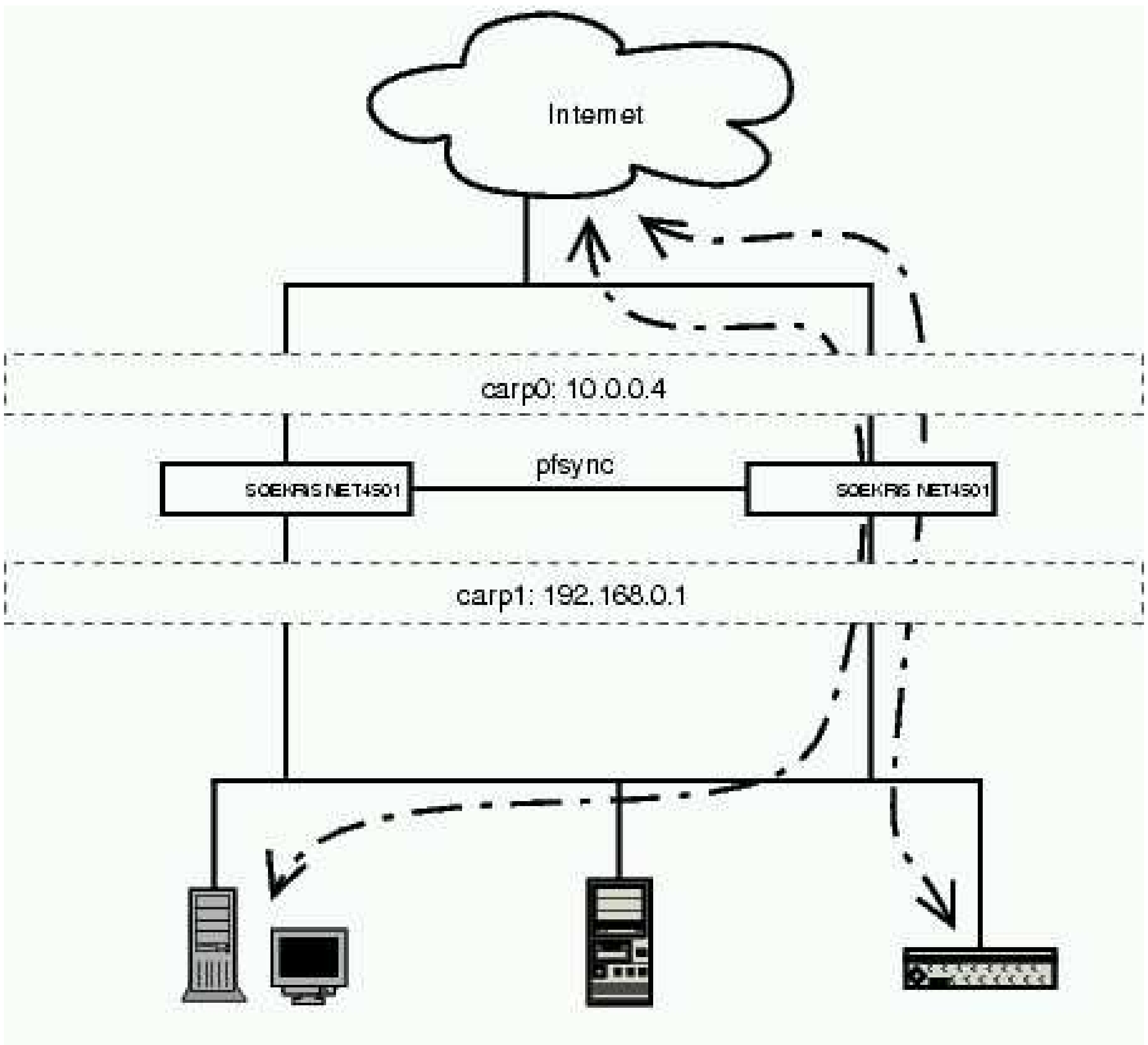
CARP

- Similar in some ways to VRRP
  - Multicast Advertisement
  - Address moved by moving a virtual MAC address
- Multiple virtual addresses on same network
- Variable advertisement interval
  - most frequent advertiser becomes master
- Advertisement protected by a SHA1 HMAC
- Addresses not in Advertisement, but in HMAC
- Supports layer 2 load balancing (ARP based)
- IPv4 and IPv6 support

pfsync and CARP integration

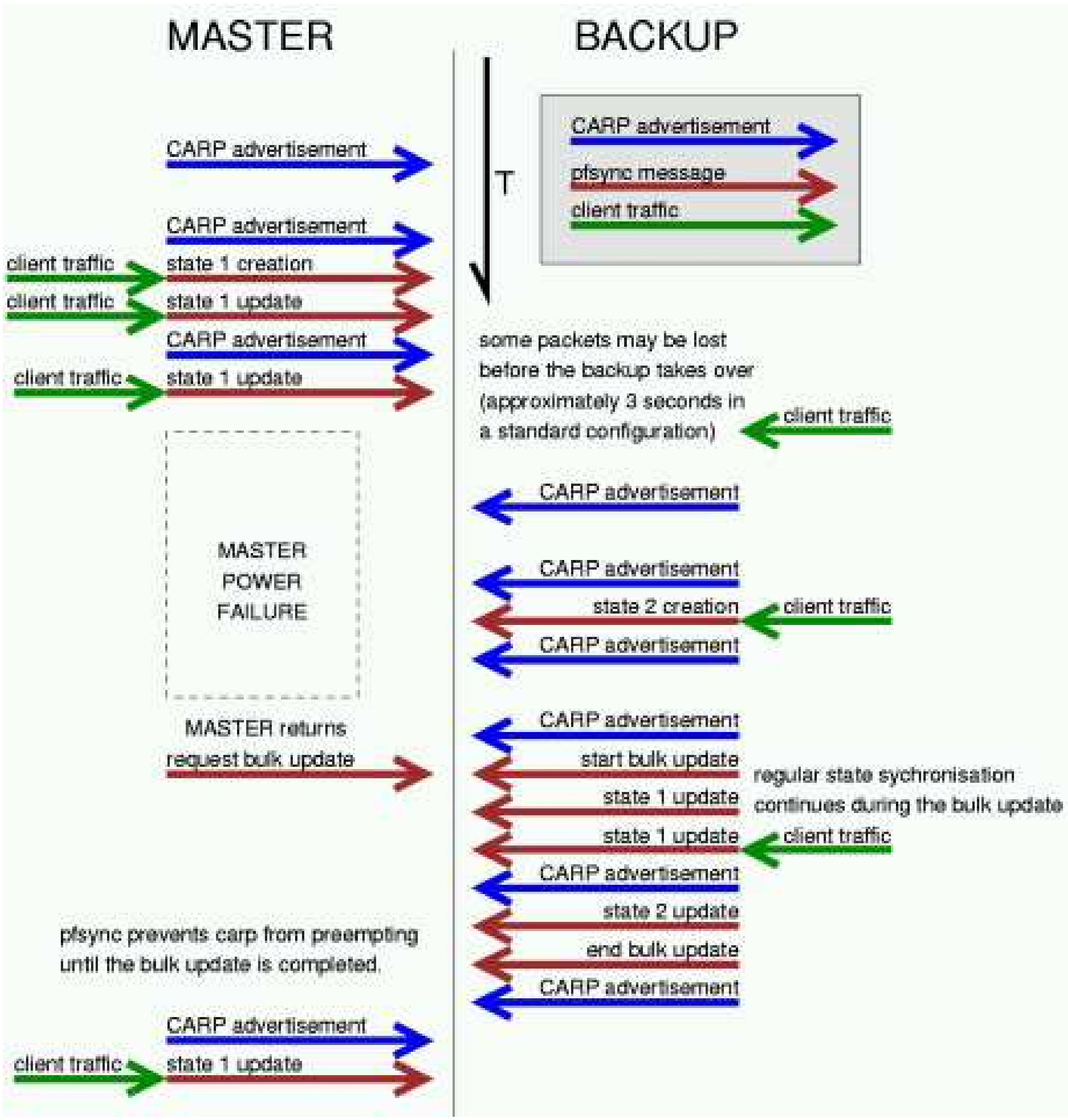
- pfsync requests a bulk update when system comes up
- Prevents CARP preemption until bulk update complete

Example


Internet
carp0: 10.0.0.4
pfsync
SOEKRIS NET4501
SOEKRIS NET4501
carp1: 192.168.0.1

Firewall Redundancy

Timeline



rdr / nat with multiple addresses

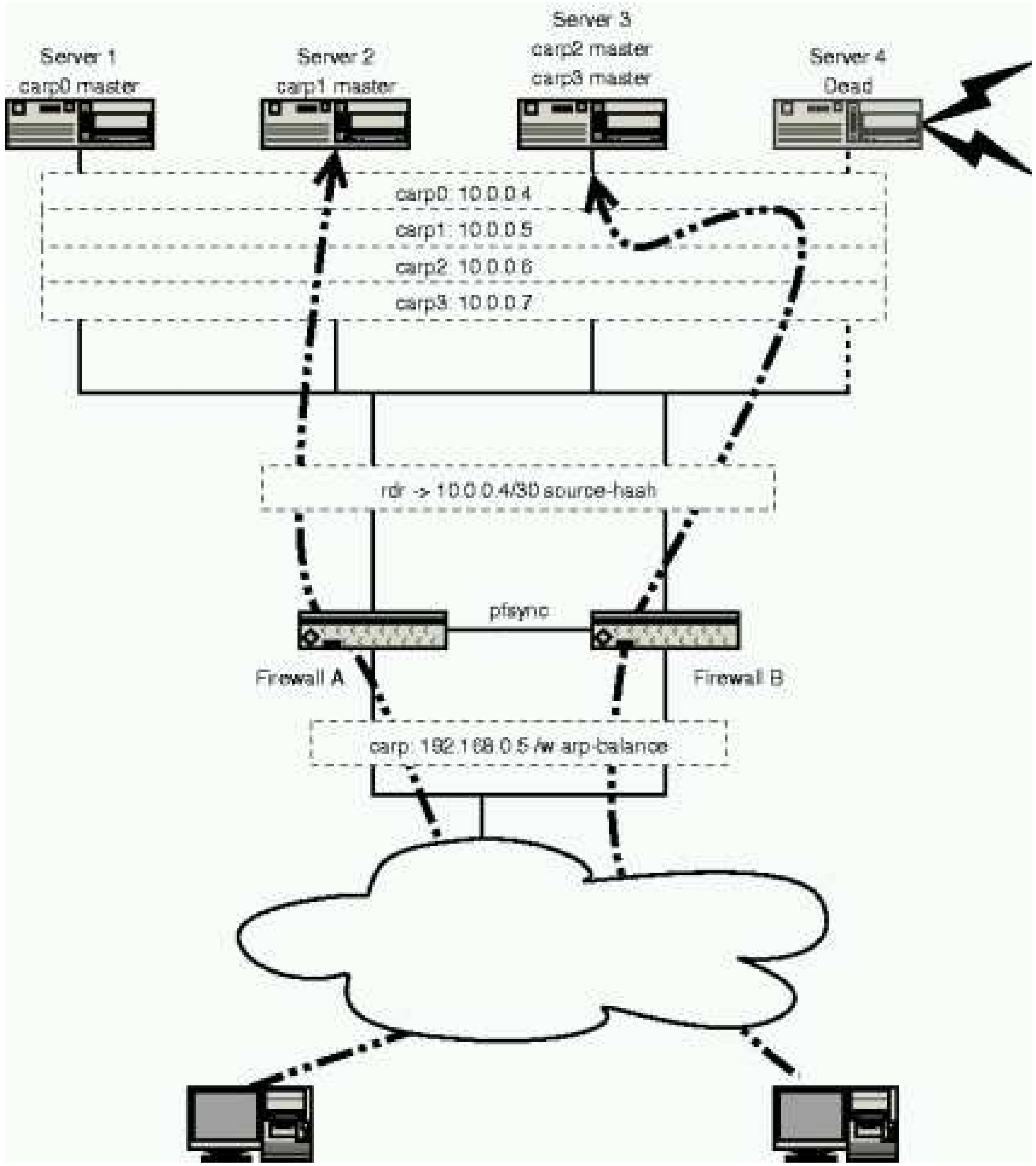
- Several address selection options
  - bitmask
  - source-hash
  - random
  - round-robin
- sticky-address
  - Can be used with ’random’ and ’round-robin’
  - Ties the source address to the translation address

CARP

- Can also provide failover to hosts as well as routers
- ’arpbalance’ balances based on arp requests
  - Multiple carp groups (one per host)
  - Group selected based on ARP request source
  - Master of that group responds with ARP
  - Only works on local segment

Load Balancing

Example



Comparison

- Commercial
  - Checkpoint
  - Pix
- Open Source
  - ipf
  - iptables

Feature Comparison

- All run on pc-style hardware
  - including "hardware firewalls" like Pix and Nokia Checkpoint.
- pix and checkpoint
  - Less flexible at the packet level
  - Do more at the application level
  - Centralised administration tools available
  - Unreliable failover
  - Poor logging formats
  - Licensing hassles
- iptables and ipfilter
  - Rulesets more complicated - this has security impacts!
  - Some application level filtering - history of security holes

Support and Training

- Training
  - Firewall administration is non-trivial
  - Training is required regardless of the pretty GUI
  - Network fundamentals more important than product specifics
  - PF does not obfuscate the network fundamentals
- Support
  - Vendor support for commercial products is often weak
  - 3rd party support for PF available
    - The difference: you have a choice
  - Support often means "someone to blame" if something goes wrong
    - Read the license - you can’t blame the vendor

Cost Comparison

Approximate cost of a failover configuration capable of 1 gigabit/s:

- Nokia Checkpoint EUR 35,000
- Cisco Pix (no ’fail-back’) EUR 40,000
    - "Support" and software updates extra!

- OpenBSD & PF (Hardware & CDs) EUR 6,000

Conclusion

- PF does one thing - packet filtering - and does it right:
  - Secure
  - Maintainable
  - Flexible
  - Easy to use
  - Fast
- And as an added bonus:
  - Cost competetive

More information

- OpenBSD manual pages
- PF User’s Guide: http://www.openbsd.org/faq/pf/

- Building Firewalls with OpenBSD and PF [2nd edition] by Jacek Artymiak
  - 3rd editon covering OpenBSD 3.7 coming soon, from O’Reilly