

Redundant firewalls with OpenBSD, CARP and pfsync

Author: [Daniele Mazzocchio](#)

Applies to: OpenBSD 4.8

Last update: Nov 20, 2010

Latest version: <http://www.kernel-panic.it/openbsd/carp/>

Table of Contents

1. Introduction.....	2
2. Network layout.....	3
3. Base configuration.....	5
4. The CARP protocol.....	7
4.1 Configuration parameters.....	7
4.1.1 The demotion counter.....	8
4.1.2 Load balancing.....	9
4.2 Parameters configuration.....	10
4.2.1 Active/standby configuration.....	10
4.2.2 Active/active configuration.....	11
5. The pfsync protocol.....	14
6. PF rules!.....	15
7. Appendix.....	18
7.1 References.....	18
7.2 Bibliography.....	18

1. Introduction

Firewalls are among the most critical network components, since their failure may cause entire groups of machines to remain offline. The damage may range from the public (web, mail, etc.) servers to become unreachable from the outside world up to being unable to surf this web site!

Using firewall clusters can dramatically reduce these risks, making the failure of a firewall completely transparent to users. Furthermore, maintenance (patching, upgrading, rebooting...) becomes much easier and faster when relying on a backup machine, thus indirectly increasing systems security and reliability.

On the other hand, it's true that redundancy raises hardware costs and can't solve each and every problem, like transparent transfer of certain protocols (e.g. SSH or IRC) between systems or synchronizing data between clustered machines (in matter of fact, we will rely on two different protocols for [failover](#) and [synchronization](#)).

The tools we will use to build our failover cluster are:

[OpenBSD](#)

largely considered one of the most secure OSes around, with “*only two remote holes in the default install, in a heck of a long time!*”;

[Packet Filter \(PF\)](#)

OpenBSD's system for filtering TCP/IP traffic and doing Network Address Translation;

[CARP \(Common Address Redundancy Protocol\)](#)

the protocol that achieves system redundancy, by having multiple computers creating a single, virtual network interface between them;

[pfsync](#)

the protocol that allows PF state tables to be synchronized between multiple firewalls.

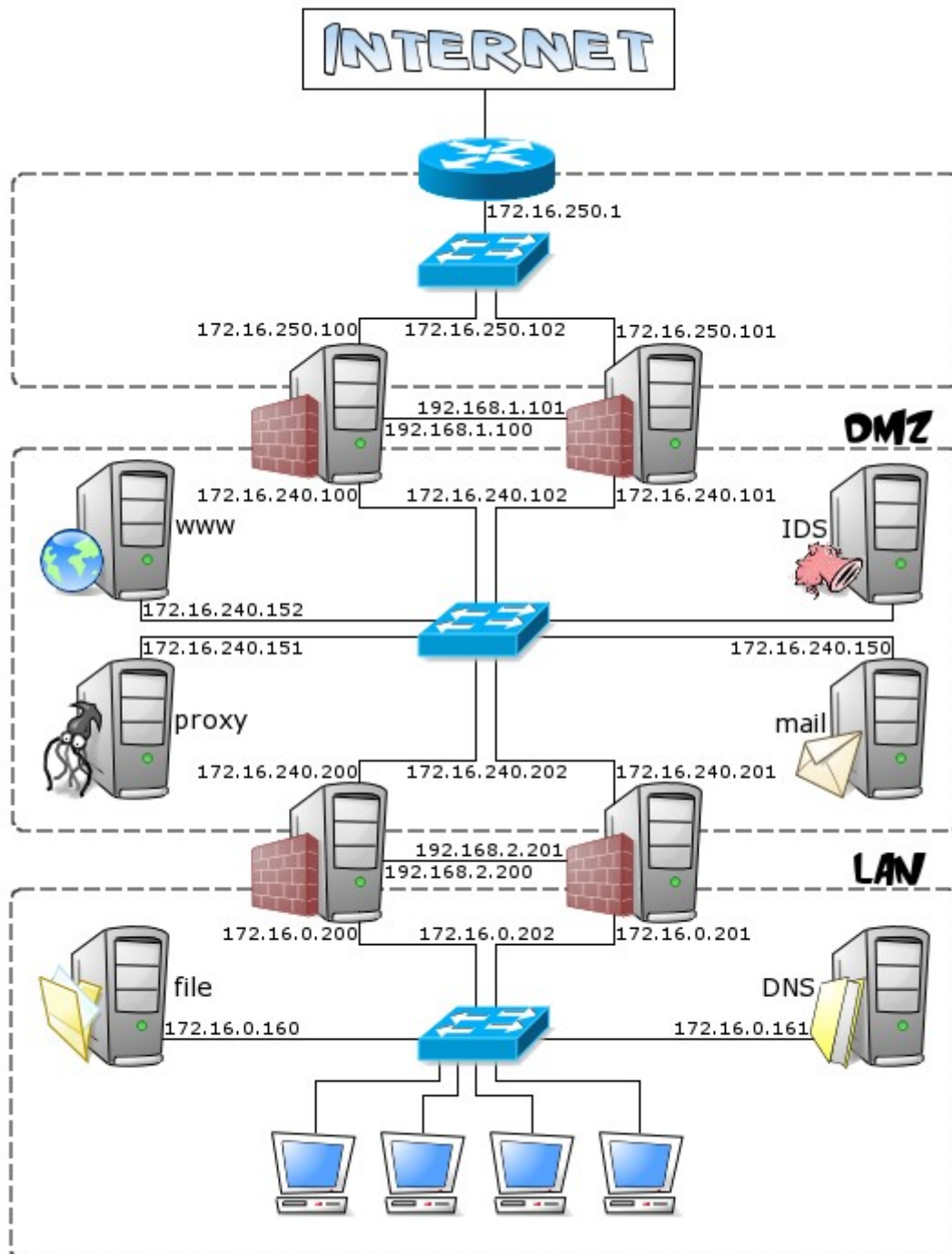
A good knowledge of OpenBSD and PF is assumed, since we won't dwell much on topics like PF maintenance commands and rules syntax. Anyway, the [appendix](#) contains some useful links for delving into these topics.

2. Network layout

First, let's take a look at the environment in which our firewall cluster will operate. It's a very simple and "classic" network, made up of:

- a DMZ (172.16.240.0/24), containing the publicly accessible machines (e.g. web and mail servers) and the intrusion detection sensors;
- a LAN (172.16.0.0/24), containing clients and servers not accessible from the public Internet (file server, DHCP server, internal DNS server...);
- a router, in a small subnet (172.16.250.0/24), to connect the network to the Internet.

This environment requires that we setup two firewall clusters: the first separating the DMZ from the Internet (we won't take into account any router filtering); the second between the LAN and the DMZ. The network looks roughly like this:



The utmost merit of this topology is that, needing two groups of firewalls, it will allow us to look over two slightly different cluster configurations. Jokes apart, these are some of its major benefits:

- in case of a firewall compromise, the LAN is protected by an additional layer of filtering (though it would be preferable to use different firewall platforms, to prevent attackers from compromising the internal firewalls with the same technique [[MISC17](#)]);
- a single (clustered) firewall, filtering both LAN and DMZ traffic, is a single point of failure;
- on each firewall, rules apply only to LAN or DMZ traffic, thus making PF rulesets cleaner and easier to maintain;

but there are also a few drawbacks:

- besides its own traffic, the DMZ must support the traffic load from the internal network to the Internet;
- double-filtering LAN traffic increases security but (slightly) affects performances;
- the cost of additional hardware may not be irrelevant.

3. Base configuration

Let's take a brief look at the base system configuration, which applies to all of our firewalls.

We won't go through the installation of the operating system, which is deeply documented on the [OpenBSD web site](#). The only (obvious) remark is that you should install only the bare minimum, to prevent firewall security and reliability from being compromised by unnecessary software. Therefore, during installation, you only need to select file sets marked as **Required** by the [documentation](#), i.e.:

- `bsd`, the kernel;
- `baseXX.tgz`, the base system;
- `etcXX.tgz`, the configuration files in `/etc`.

There should be no need to install the compiler (`compXX.tgz`), also to avoid providing such a useful tool to possible intruders (see [\[PUIS\]](#)).

After the first reboot, we can start setting up some configuration files; by default, OpenBSD comes with very few services enabled through [inetd\(8\)](#):

```
$ grep -v ^# /etc/inetd.conf
ident      stream  tcp     nowait  _identd /usr/libexec/identd  identd -el
ident      stream  tcp6    nowait  _identd /usr/libexec/identd  identd -el
127.0.0.1:comsat dgram  udp     wait    root    /usr/libexec/comsat  comsat
[::1]:comsat dgram  udp6    wait    root    /usr/libexec/comsat  comsat
daytime    stream  tcp     nowait  root    internal
daytime    stream  tcp6    nowait  root    internal
time       stream  tcp     nowait  root    internal
time       stream  tcp6    nowait  root    internal
$
```

The system is considered secure also with these services turned on (see [\[ABSO\]](#)); anyway, disabling them all will make no harm.

It's also a good practice to edit the `/etc/motd` file to give as few information as possible about the system and to warn users, whether legitimate or not, that all access is being logged and that any unauthorized access will be prosecuted (see [\[PUIS\]](#)).

You should already have configured the network during installation; anyway, if you need to make some changes, these are the main files to edit:

[/etc/hostname.if\(5\)](#)

containing information regarding the configuration of each network interface (address, netmask, etc.);

[/etc/mygate\(5\)](#)

containing the address of the gateway host;

[/etc/myname\(5\)](#)

containing the symbolic hostname (FQDN) of the machine;

[/etc/resolv.conf\(5\)](#)

containing the resolver configuration settings (name servers, local domain name, etc.).

Considering the large amount of DNS-based attacks, it is also preferable, especially on firewalls, not to rely on DNS to resolve names and addresses of the most critical systems, but rather inserting them into the [/etc/hosts\(5\)](#) file. To make sure the [/etc/hosts\(5\)](#) file has a higher priority than DNS, just make sure that the first line in [/etc/resolv.conf\(5\)](#) is:

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

Packet Filter is enabled by default and loads rules from the [/etc/pf.conf\(5\)](#) file; a different path can be specified by assigning it to the `pf_rules` variable in [/etc/rc.conf.local\(8\)](#).

```
/etc/rc.conf.local
```

```
pf_rules=/new/path/to/pf.conf
```

You may also set [pflogd\(8\)](#) flags in the variable `pflogd_flags`. Last, don't forget to enable IP forwarding by issuing the command:

```
# sysctl net.inet.ip.forwarding=1  
net.inet.ip.forwarding: 0 -> 1  
#
```

and to uncomment the following line in [/etc/sysctl.conf\(5\)](#) to re-enable it after reboot:

```
/etc/sysctl.conf
```

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

4. The CARP protocol

CARP (*Common Address Redundancy Protocol*) is the protocol that achieves system redundancy, by having a group of hosts on the same network segment (*redundancy group*) to share an IP address, so that, should a machine fail, another host in the redundancy group can respond in its place. CARP also allows a degree of [load sharing](#) between systems.

Although creating redundant firewalls is one of its most common uses, CARP isn't a firewall-specific protocol. It can be used to ensure service continuity and/or load sharing to a number of network services.

Initially, the OpenBSD team wanted to produce a free implementation of the IETF standard protocols, VRRP (*Virtual Router Redundancy Protocol*), defined in [[RFC3768](#)], and HSRP (*Hot Standby Router Protocol*), defined in [[RFC2281](#)]; but Cisco, claiming patent rights on it, firmly informed the OpenBSD community that Cisco would defend its patents for VRRP implementation (see [[CARP](#)] for more details), thus forcing the OpenBSD developers to create a new, competing protocol designed to be fundamentally different from VRRP.

CARP is a multicast protocol, grouping several physical computers together under one or more virtual addresses. Of these, one system is the master and responds to all packets destined for the group; the other systems (backups) just stand by, waiting for any problem to take its place (as it happens between co-workers...).

At configurable intervals, the master advertises its operation on IP protocol number 112. If the master goes offline, the other hosts in the redundancy group begin to advertise. The host that's able to advertise most frequently becomes the new master. When the main system comes back up, it becomes a backup host by default, although it can be configured to try to become master again.

As you can see, CARP only creates and manages the virtual network interface; it's up to the system administrator to synchronize data between applications, using [pfsync\(4\)](#) (which we'll discuss in the [next chapter](#)), [rsync](#) or whatever protocol is appropriate for the specific application.

4.1 Configuration parameters

CARP configuration is done via the [sysctl\(8\)](#) and [ifconfig\(8\)](#) commands. There are three relevant [sysctl\(3\)](#) variables:

`net.inet.carp.allow`

it defines whether the host handles CARP packets or not. It is enabled by default;

`net.inet.carp.log`

it defines whether to log CARP errors or not. It may be a value between 0 and 7, corresponding to the [syslog\(3\)](#) priorities, and defaults to 2 (i.e. only CARP state changes are logged);

`net.inet.carp.preempt`

if set to 0 (default), the host won't try to become master if it receives CARP advertisements from another master. Otherwise, it will try to become master if it is able to advertise more frequently than the current master. This option also enables failing over all interfaces in the event that one interface goes down. In fact, if one physical CARP-enabled interface goes down, CARP will increase by 1 the demotion counter (see [below](#)) for all groups that the interface belongs to, thus allowing the election of new masters on all subnets.

The syntax for configuring CARP with [ifconfig\(8\)](#) is:

```
ifconfig carpN create

ifconfig carpN [advbase n] [advskew n] [balancing mode] \
[carpnodes vhid:advskew, vhid:advskew, ...] [carpdev iface] \
[[-]carppeer peer_address] [pass passphrase] [state state] [vhid host-id]
```

carpN

the name of the [carp\(4\)](#) virtual interface.

advbase, advskew

these values determine the interval between two consecutive CARP advertisements. This interval (in seconds) is given by the formula $(advbase + (advskew / 255))$; increasing advbase (the default value is 1 second) will decrease network traffic but increase the delay in electing the new master. Small advskew values allow a host to advertise more frequently, increasing its probability to become master. advbase must be a value between 1 and 255, advskew between 0 (the default) and 254;

balancing

sets the load balancing mode (which will be discussed [later](#)); valid modes are arp, ip, ip-stealth, and ip-unicast;

carpnodes

a comma-separated list of vhid:advskew pairs to actually define how the load should be shared among the configured carp nodes (see [below](#) for further details);

carpdev

specifies the physical interface that belongs to this redundancy group. By default, CARP uses the physical interface that belongs to the same subnet as the virtual interface;

carppeer

allows you to specify the IP address of the other CARP peer(s), instead of using the default multicast group;

pass

the authentication password to use when talking to other CARP-enabled hosts in the redundancy group. This must be the same on all members of the group;

state

Force a [carp\(4\)](#) interface into a certain state (init, backup or master);

vhid

the Virtual Host ID. This is a unique number (between 1 and 255) that is used to identify the redundancy group to the other nodes on the network.

4.1.1 The demotion counter

Besides basic configuration, the [ifconfig\(8\)](#) command also allows you to tweak the CARP *demotion counter*, which is “a measure of how “ready” a host is to become master of a CARP group” (the higher the counter, the less ready the host). Let's see it in more detail.

CARP interfaces are divided in *groups* (by default all [carp\(4\)](#) interfaces are members of the "carp" interface group) and each group is assigned a demotion counter, whose value can be viewed by running the following command:

```
$ ifconfig -g carp
carp: carp demote count 0
```

The demotion counter comes in handy mainly when:

- you want to momentarily prevent a host from becoming master: for instance, at boot time, the [rc\(8\)](#) script increases the demotion counter by 128 before starting the network and decreases it by the same amount once all interfaces have been initialized and all network daemons have been started (the demotion counter can't be set to an absolute value, but only increased or decreased by a certain amount):

```
/etc/rc
```

```
ifconfig -g carp carpdemote 128
```

[...]

```
ifconfig -g carp -carpdemote 128
```

- you want to gracefully failover only a limited number of a host's [carp\(4\)](#) interfaces (not all of them, as it happens when an interface goes down and `preempt` is enabled). In the following example, we will failover the `carp1` and `carp2` interfaces and leave the state of the others unchanged:

```
# ifconfig carp1 group morituri
# ifconfig carp2 group morituri
# ifconfig morituri
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      carp: MASTER carpdev sis0 vhid 1 advbase 1 advskew 100
      groups: carp morituri
      inet 1.2.3.4 netmask 0xffffffff broadcast 1.2.3.255
carp2: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      carp: MASTER carpdev sis1 vhid 2 advbase 1 advskew 100
      groups: carp morituri
      inet 2.3.4.5 netmask 0xffffffff broadcast 2.3.4.255
# ifconfig -g morituri
morituri: carp demote count 0
# ifconfig -g morituri carpdemote 50
# ifconfig -g morituri
morituri: carp demote count 50
```

For further details on the CARP demotion counter, please refer to [\[PFFAQ\]](#).

4.1.2 Load balancing

CARP provides two different methods for load balancing incoming network traffic among a set of CARP-enabled hosts: ARP balancing and IP balancing.

Both methods require that you first create a load balancing group by configuring, on each balanced [carp\(4\)](#) interface, as many VHIDs as hosts in the balancing group; the `advskew` on each VHID will be configured so that each host will be the master on a separate VHID (see [below](#) for a practical example).

ARP balancing works by applying a hash function to the source MAC address of ARP requests to determine which VHID should handle the request. The ARP request will be answered solely by the host whose [carp\(4\)](#) interface is master for that VHID. ARP load balancing can be enabled through [ifconfig\(8\)](#) by setting the value of the `balancing` option to "arp" on all hosts; for instance:

```
# ifconfig carp0 balancing arp carpnodes 1:0,2:100
```

IP load balancing works in a very similar manner to ARP balancing, but uses the hash of the source and destination addresses of the IP packet to determine which VHID (and therefore which host) should accept the packet.

IP balancing requires that traffic destined to the CARP address be received by all CARP hosts. It can be enabled using [ifconfig\(8\)](#), by setting the `balancing` option to "ip"; this will cause CARP to use a multicast MAC address, forcing the switch to send incoming traffic to all nodes in the redundancy group. For example:

```
# ifconfig carp0 balancing ip carpnodes 1:0,2:100
```

Alternatively, you can set the `balancing` option to "ip-stealth" (stealth mode), in order to prevent hosts from sending packets with their virtual MAC address as source; this will prevent the switch from learning the virtual MAC address, forcing it to flood the traffic to all its ports. Last, if you're using a hub

or a switch that supports some kind of monitoring mode, you can set `balancing` to `"ip-unicast"`.

The choice between the two load balancing mechanisms mostly depends on the network environment in which the systems will be placed: ARP balancing only works for clients in the local network and cannot balance traffic that crosses a router, as routed traffic always contains the MAC address of the router as its source address. Therefore, if clients are on remote networks, IP balancing is the only option; the only drawback of IP balancing is that traffic destined towards the load balanced IP addresses must be received by all CARP-enabled hosts, resulting in a higher network load.

4.2 Parameters configuration

Now it's time to configure CARP on our firewalls. To examine two slightly different CARP configurations, we will set up the two internal firewalls (Mickey and Minnie, between LAN and DMZ) in active/stand-by mode, with only one system filtering the whole network traffic and the other one acting as a hot spare; the two external firewalls (Donald and Daisy, separating the DMZ from the internet), instead, will be in active/active mode, sharing the traffic load.

So let's recap the firewalls addresses, as we have seen them in the [network diagram](#):

	Mickey	Minnie	Virtual address
LAN	172.16.0.200	172.16.0.201	172.16.0.202
DMZ	172.16.240.200	172.16.240.201	172.16.240.202
pfsync	192.168.2.200	192.168.2.201	
	Donald	Daisy	Virtual address
DMZ	172.16.240.100	172.16.240.101	172.16.240.102
Internet	172.16.250.100	172.16.250.101	172.16.250.102
pfsync	192.168.1.100	192.168.1.101	

4.2.1 Active/standby configuration

Let's start with Mickey and Minnie: first, we need to create the `carp*` devices and configure them with [ifconfig\(8\)](#):

```
mickey# ifconfig carp0 172.16.0.202/24 vhid 1 pass password1 advbase 1 advskew 0
mickey# ifconfig carp1 172.16.240.202/24 vhid 2 pass password2 advbase 1 advskew 0
```

```
minnie# ifconfig carp0 172.16.0.202/24 vhid 1 pass password1 advbase 1 advskew 100
minnie# ifconfig carp1 172.16.240.202/24 vhid 2 pass password2 advbase 1 advskew 100
```

We have just created the interfaces, assigned them an IP address, a virtual host ID (1 on the LAN, 2 on the DMZ) and a password (probably not the most secure...) for authentication. We also decided that, whenever possible, Mickey will be the master; this is done by giving Minnie a higher `advskew` value (100), thus making the interval between its advertisements ($1 + 100 / 255$) higher than the interval between Mickey's advertisements ($1 + 0 / 255$). And, as we've seen [above](#), the host that's able to advertise most frequently becomes master.

Furthermore, by setting `net.inet.carp.preempt` to "1" on Mickey, we ensure that Mickey will always try to become the master:

```
mickey# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1
```

To make these settings permanent after reboot, we just need to edit the `/etc/hostname.carp*` and

/etc/sysctl.conf files on Mickey:

```
/etc/hostname.carp0
```

```
inet 172.16.0.202 255.255.255.0 172.16.0.255 vhid 1 pass password1 advbase 1 advskew
0
```

```
/etc/hostname.carp1
```

```
inet 172.16.240.202 255.255.255.0 172.16.240.255 vhid 2 pass password2 advbase 1
advskew 0
```

```
/etc/sysctl.conf
```

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

and on Minnie:

```
/etc/hostname.carp0
```

```
inet 172.16.0.202 255.255.255.0 172.16.0.255 vhid 1 pass password1 advbase 1 advskew
100
```

```
/etc/hostname.carp1
```

```
inet 172.16.240.202 255.255.255.0 172.16.240.255 vhid 2 pass password2 advbase 1
advskew 100
```

Note: to make the adoption of CARP easier on pre-existing networks, CARP allows using the physical address of a host as the virtual address of the whole redundancy group.

4.2.2 Active/active configuration

Now let's get on to Donald and Daisy and start by configuring their DMZ interfaces. As before, we will create the `carp0` device on each machine, but this time, to enable load balancing, we will use the `carpnodes` option to assign two different Virtual Host IDs to the interface (VHIDs 3 and 4).

On VHID 3, we will set the `advskew` of Donald and Daisy to 0 and 100 respectively: this will ensure that Donald becomes master for that VHID; on VHID 4, instead, we will do the opposite, by setting the `advskew` of Donald and Daisy to 100 and 0 respectively, in order to force Daisy to become master for VHID 4:

```
donald# ifconfig carp0 172.16.240.102/24 balancing ip carpnodes 3:0,4:100 \
> pass password3
donald# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1
```

```
daisy# ifconfig carp0 172.16.240.102/24 balancing ip carpnodes 3:100,4:0 \
> pass password3
daisy# sysctl net.inet.carp.preempt=1
net.inet.carp.preempt: 0 -> 1
```

We now have two redundancy groups with the same IP address, but each with a different master:

```
donald# ifconfig carp0
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    lladdr 01:00:5e:00:01:01
    carp: carpdev r11 advbase 1 balancing ip
```

```

state MASTER vhid 3 advskew 0
state BACKUP vhid 4 advskew 100
groups: carp
inet 172.16.240.102 netmask 0xffffffff broadcast 172.16.240.255
inet6 fe80::2c0:a8ff:fe8e:b112%carp0 prefixlen 64 scopeid 0x5

```

```

daisy# ifconfig carp0
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
lladdr 01:00:5e:00:01:01
carp: carpdev r11 advbase 1 balancing ip
state BACKUP vhid 3 advskew 100
state MASTER vhid 4 advskew 0
groups: carp
inet 172.16.240.102 netmask 0xffffffff broadcast 172.16.240.255
inet6 fe80::219:d2ff:fe02:6469%carp0 prefixlen 64 scopeid 0x5

```

To make these settings permanent across reboots, we need to edit the startup files on Donald:

```
/etc/hostname.carp0
```

```
inet 172.16.240.102 255.255.255.0 172.16.240.255 balancing ip carpnodes 3:0,4:100
pass password3
```

```
/etc/sysctl.conf
```

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

and Daisy:

```
/etc/hostname.carp0
```

```
inet 172.16.240.102 255.255.255.0 172.16.240.255 balancing ip carpnodes 3:100,4:0
pass password3
```

```
/etc/sysctl.conf
```

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

Now we just have to do the same on the external network interfaces, with another two Virtual Host IDs (VHIDs 5 and 6):

```

donald# ifconfig carp1 172.16.250.102/24 balancing ip carpnodes 5:0,6:100 \
> pass password5

```

```

daisy# ifconfig carp1 172.16.250.102/24 balancing ip carpnodes 5:100,6:0 \
> pass password5

```

and edit the startup files on Donald:

```
/etc/hostname.carp1
```

```
inet 172.16.250.102 255.255.255.0 172.16.250.255 balancing ip carpnodes 5:0,6:100
pass password5
```

and Daisy:

```
/etc/hostname.carp1
```

```
inet 172.16.250.102 255.255.255.0 172.16.250.255 balancing ip carpnodes 5:100,6:0
```

```
pass password5
```

Though the above configuration involves only a couple of machines, it can be easily extended to up to 32 hosts. *One last note:* load sharing won't probably achieve a perfect 50/50 distribution between the two machines, since CARP uses a hash of the source and destination IP addresses to determine which system should accept a packet, not the actual load.

5. The pfsync protocol

Pfsync is the protocol used by [Packet Filter](#) to manage and update state tables, which allow for stateful inspection and NAT. By default, state change messages are sent out on the synchronization interface using IP multicast packets. The protocol is IP protocol 240 and the multicast group used is 224.0.0.240. We will use it to synchronize state tables among firewalls of the same redundancy group and, in the event of a failover, allow network traffic to flow uninterrupted through the new master firewall.

[pfsync\(4\)](#) is also the name of the pseudo-device on which PF state table changes pass (except states created by rules marked with the `no-sync` keyword or by [pfsync\(4\)](#) packets). [pfsync\(4\)](#) can be configured with a physical synchronization interface, in order to merge the state tables of multiple firewalls.

The physical synchronization interface can be set through [ifconfig\(8\)](#), using the `syncdev` parameter; for example, on our firewalls, we can write:

```
# ifconfig pfsync0 syncdev r12
```

assuming that the `r12` interface is, on each host (see [picture](#)), the interface on the 192.168.1.0/24 subnet (for Mickey and Minnie) or 192.168.2.0/24 (for Donald and Daisy) and cross-cabled to the "beloved" firewall.

Crossover cables are recommended because the pfsync protocol doesn't provide any cryptography or authentication mechanism; if you don't use a secure network, like a crossover cable, an attacker may use spoofed pfsync packets to alter the firewalls state tables and bypass filter rules.

Alternatively, you can use the `syncpeer` keyword to specify the address of the firewall to synchronize with. The system will use this address, instead of multicast, as the destination of [pfsync\(4\)](#) messages, allowing the use of [IPsec](#) to protect the communication. In this case, `syncdev` must be set to the [enc\(4\)](#) pseudo-device, which encapsulates/decapsulates [ipsec\(4\)](#) traffic. E.g.:

```
# ifconfig pfsync0 syncpeer 192.168.1.101 syncdev enc0
```

To make these settings permanent after reboot, we need to edit the `/etc/hostname.pfsync0` file on each firewall:

```
/etc/hostname.pfsync0
```

```
up syncdev r12
```

6. PF rules!

The impact of CARP and [pfsync\(4\)](#) on Packet Filter rules is really minimal. First, you need to let the PFSYNC and CARP protocols pass on their own interfaces:

```
pass quick on rl2 proto pfsync keep state (no-sync)
pass on { rl0, rl1 } proto carp keep state (no-sync)
```

Then, when writing firewall rules, keep in mind that, from [pf\(4\)](#)'s point of view, all traffic passes through the physical interface. Thus, in cases like:

```
pass in on $ext_if [...]
```

you can keep referring to the physical, not the virtual interface.

On the other hand, the virtual address is associated to the CARP interface; thus, you need to refer to it if the firewall offers any services on its virtual address:

```
# SSH on the virtual interface
pass in on $int_if inet proto tcp from $int_if:network to carp0 port ssh
```

or on a NATed server, through traffic redirection:

```
# Mail server accessible from the internet
pass in on $ext_if inet proto tcp from any to carp2 port $mail_ports \
    rdr-to $mail_srv
```

In all other cases, CARP is perfectly transparent to [pf\(4\)](#), as for services offered by the firewall on its physical addresses:

```
# SSH on the physical address
pass in on $int_if inet proto tcp from $int_if:network to $int_if port ssh
```

or for normal filtering:

```
# External DNS
pass in on $int_if inet proto { tcp, udp } from $int_if:network to $dns_srv \
    port domain
pass out on $ext_if inet proto { tcp, udp } from $ext_if to $dns_srv \
    port domain
```

As an example, let's see a basic PF ruleset for our external firewalls, Donald and Daisy:

```
/etc/pf.conf
```

```
#####
# Macros and lists
#####

ext_if = rl0                # External interface
int_if = rl1                # DMZ interface
pfs_if = rl2                # Pfsync interface
carp_if = carp1            # External CARP interfaces

mail_srv = "mail.kernel-panic.it"           # Mail server
web_srv = "{ www1.kernel-panic.it, www2.kernel-panic.it }" # Web servers
dns_srv = "{ dns1.isp.com, dns2.isp.com }"  # DNS servers
int_fw = "{ mickey.kernel-panic.it, minnie.kernel-panic.it }" # Internal fw

mail_ports = "{ smtp, imap, imaps }"        # Mail server ports
```

```

web_ports = "{ www, https }"           # Web server ports
# Allowed incoming ICMP types
icmp_types = "{ echoreq, timex, paramprob, unreachable code needfrag }"

# Private networks (RFC 1918)
priv_nets = "{ 127.0.0.0/8, 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 }"

#####
# Options, scrub and NAT                                     #
#####

set block-policy drop
set loginterface $ext_if
set skip on lo

# NAT outgoing connections
match out on $ext_if from !$ext_if to any nat-to $ext_if

# Redirect web services (with load balancing)
match in on $ext_if inet proto tcp from any to $carp_if port $web_ports \
    rdr-to $web_srv round-robin sticky-address

# Redirect mail services
match in on $ext_if inet proto tcp from any to $carp_if port $mail_ports \
    rdr-to $mail_srv

#####
# Filtering rules                                           #
#####

block all                # Default deny
block in quick from urpf-failed # Spoofed address protection

# Scrub incoming packets
match in all scrub (no-df)

pass quick on $pfs_if proto pfsync           # Enable pfsync
pass quick on { $int_if, $ext_if } proto carp # Enable CARP

block in  quick on $ext_if from $priv_nets to any
block out quick on $ext_if from any to $priv_nets

# Mail server
pass in  on $ext_if inet proto tcp from any to $mail_srv port $mail_ports
pass out on $int_if inet proto tcp from any to $mail_srv port $mail_ports
pass in  on $int_if inet proto tcp from $mail_srv to any port smtp
pass out on $ext_if inet proto tcp from $ext_if to any port smtp modulate state

# Web servers
pass in  on $ext_if inet proto tcp from any to $web_srv port $web_ports \
    synproxy state
pass out on $int_if inet proto tcp from any to $web_srv port $web_ports

# ICMP
pass in  inet proto icmp all icmp-type $icmp_types
pass out inet proto icmp all

# DNS
pass in  on $int_if inet proto { tcp, udp } from $int_if:network to $dns_srv \
    port domain
pass out on $ext_if inet proto { tcp, udp } from $ext_if to $dns_srv \
    port domain

# Internet web servers

```

```
pass in on $int_if inet proto tcp from $int_fw to any port $web_ports
pass out on $ext_if inet proto tcp from $ext_if to any port $web_ports \
modulate state
```

7. Appendix

7.1 References

- [MISC17] - *Filtrage applicatif : le cas des clients web, mail et p2p* (MISC N°17)
- [PUIS] - *Practical UNIX and Internet Security*, Simson Garfinkel and Gene Spafford, O'Reilly, 2003
- [ABSO] - *Absolute OpenBSD*, Michael W. Lucas, No Starch Press, 2003
- [CARP] - "CARP License" and "Redundancy must be free"
- [PFFAQ] - PF: The OpenBSD Packet Filter
- [RFC3768] - RFC 3768, Virtual Router Redundancy Protocol (VRRP)
- [RFC2281] - RFC 2281, Cisco Hot Standby Router Protocol (HSRP)

7.2 Bibliography

- [The Common Address redundancy Protocol \(CARP\)](#)
- [Firewall Failover with pfsync and CARP](#)
- [Firewalling with OpenBSD's PF packet filter](#), Peter N. M. Hansteen, 2008
- [The Book of PF, 2nd Edition](#), Peter N. M. Hansteen, No Starch Press, 2010
- [The OpenBSD PF Packet Filter Book](#), Jeremy C. Reed, Reed Media Services, 2006