

# Failover

Jason Healy, Director of Networks and Systems

Last Updated Mar 18, 2008



# Contents

<b>1</b>	<b>Failover and High Availability</b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Terms and Definitions . . . . .	5
1.3	Mac OS X IPFailover . . . . .	6
1.3.1	Configuring Failover . . . . .	7
1.3.2	Failover Transition Scripting . . . . .	9
1.3.3	Peering Failover . . . . .	10
1.3.4	Using the Suffield Configs . . . . .	10



# Chapter 1

## Failover and High Availability

Last updated 2008/03/18

### 1.1 Introduction

Network infrastructure occasionally requires maintenance, no matter how well-designed it is. However, many organizations have come to depend on network services to such a degree that downtime adversely affects the entire organization. Thus, we must find a way to increase the **availability** of service, even when parts of the infrastructure are not operating.

In the network world, increased availability usually requires duplicate hardware, and additional expense. We don't have a lot of money to throw at the problem, so the approaches outlined in this document use a minimum of hardware and expense to accomplish their task. Obviously, you must decide what level of downtime your organization can withstand, and balance this with the added cost and complexity of a suitable failover system.

### 1.2 Terms and Definitions

Before we begin, some quick discussion of terminology:

- When a network service is "up and running", we say that it is **available**. The goal is to achieve the highest **availability** possible, so that users of

the service are not disrupted. When special steps are taken to prevent outages, we aim for **high availability** (or **HA**).

- High availability usually requires **redundant hardware** (so as to tolerate the loss of a particular system). Several strategies exist for using this redundant hardware.
- **Load balancing** involves having service requests routed to all redundant hardware. If one server dies, the balancer leaves it out of rotation until it returns. This provides maximum hardware utilization, but requires special load-balancing hardware and/or software.
- **Hot-spare failover** involves setting up two duplicate servers, but only one is active at any given time. The two servers constantly communicate, and in the event that the primary becomes unavailable, the secondary server takes over. This setup is relatively simple to configure, but can involve a large amount of "wasted" resources for the backup machine, as it is not frequently used.
- **Warm-spare failover** is similar to a hot-spare, except that the spare might do something else in addition to serve as a backup. When the spare detects a fault in the primary, it reconfigures itself to take over the primary's services in addition to its own.
- **Cold-spare failover** basically means that a duplicate machine is available to take over, but it must be booted and/or configured by hand before services resume. Cold-spare requires no up-front configuration, and can be inexpensive (especially if a single machine is a spare for many others), but it involves the largest amount of downtime while the spare is configured.

### 1.3 Mac OS X IPFailover

Mac OS X Server 10.4 ships with built-in IP-based failover support. Out of the box, it's a **warm-spare failover** setup; the backup server will take over the IP address of the failed server automatically, but it is up to the administrator to configure the server to start any additional services that may be required.

Failover is implemented using two system-level daemons:

1. **heartbeated** runs on the primary server, and broadcasts "availability" packets at regular intervals. These packets serve as an announcement that the machine is up and running.
2. **failoverd** runs on the secondary server, and listens for the broadcast packets from the primary. In the event that the broadcasts are not received, the secondary takes over the primary's IP address.

Apple requires the following to implement IP-based failover:

- Both machines be on the same **public** subnet
- Both machines are connected via an independent **private** subnet
- Each machine have its own unique IP address on both subnets

Again, Apple's solution only transfers the IP address from one server to another; you are responsible for ensuring that the secondary server contains the data and configuration necessary to actually take over the network services.

To help with this, Apple uses a customizable scripting framework to help you launch processes during a failover situation (more on this below).

### 1.3.1 Configuring Failover

For the purposes of this document, we'll be using the following sample names. **You must replace the names and IP addresses with those of your actual equipment.**

- The **primary** server has public IP address 10.0.0.100 and private IP address 192.168.0.1.
- The **secondary** server has public IP address 10.0.0.200 and private IP address 192.168.0.2.

#### Public Network

First, ensure that both machines are on the same public subnet, and are able to reach each other. In our example, both machines are on the 10.0.0.0/24 subnet.

If you have enabled firewall software, ensure that it allows UDP traffic destined for port 1694 to reach the server. **Note:** Apple's built-in firewall entry for "IP Failover" incorrectly defaults to TCP traffic for this rule; you must also enable UDP.

#### Private Network

Next, connect both servers together via an independent private network. This can be over a second ethernet connection, firewire cable, or other network.

Ensure that the secondary network interface appears **below** the primary interface in the **Network Settings** preference pane; this ensures that the machine will only use the private network when the public network is down.

Also ensure that the private network has **no DNS information** specified. All DNS information should be obtained from the public network.

If you have enabled firewall software, ensure that it allows UDP traffic destined for port 1694 to reach the server. **Note:** Apple's built-in firewall entry for "IP Failover" incorrectly defaults to TCP traffic for this rule; you must also enable UDP.

### Configuring the Primary Server

On the primary server, edit the `/etc/hostconfig` file and add a line containing the `**broadcast addresses*` of both the public and private subnets. Using our sample IPs from above, our line looks like:

```
FAILOVER_BCAST_IPS="192.168.0.255 10.0.0.255"
```

Save the file and reboot the primary server (or manually start the `IPFailover` service using `SystemStarter`).

The primary should now have a `heartbeatd` process running, and a `tcpdump` listening for port 1694 should show regular traffic from the server on both its public and private interfaces. If this is happening, move on to the next step.

### Configuring the Secondary Server

On the secondary server, edit the `/etc/hostconfig` file and add lines defining the primary server's IP address, and the interface that should assume the address in the event of a failover. Additionally, you may specify an e-mail address to send notifications to when a failover occurs:

```
FAILOVER_PEER_IP="10.0.0.100"  
FAILOVER_PEER_IP_PAIRS="en0:10.0.0.100"  
FAILOVER_EMAIL_RECIPIENT="root@example.org"
```

The second line's syntax says that the address 10.0.0.100 should be added to the `en0` interface. If your machine has multiple interfaces, specify the one that should take over the primary server's address.

Save the file and reboot the secondary server (or manually start the `IPFailover` service using `SystemStarter`).

You should now have a `failoverd` process running on the secondary server, ready to take over from the primary. You may test this by unplugging **both** network interfaces from the primary (or simply shutting it down). The secondary server should notice that the server is unavailable and take over the IP address. Additionally, you should receive an e-mail notification about the takeover.

If you reconnect the primary server, the secondary should notice and relinquish its address within 15 seconds. Again, an e-mail notification is sent to confirm the change.

### 1.3.2 Failover Transition Scripting

The process described above handles the takeover of an IP address from a primary server to a secondary one. However, that's all it does; if your secondary server is not already running all the services that the primary uses, then the takeover won't help you.

For this reason, Apple has provided a scriptable framework so you can take specific actions whenever a secondary server takes over (or gives up) the primary's address.

#### File Locations

Apple's IPFailover scripts look for a directory in `/Library/IPFailover` named after the public IP address of the primary server. In our example, the secondary server would have a directory named:

```
/Library/IPFailover/10.0.0.100
```

This directory can contain several scripts, outlined below:

- **Test**, which is run before any takeover is attempted. If the script returns with zero status, the takeover continues; if it returns non-zero status the takeover is aborted. This allows for conditional takeover depending on other external factors.
- **PreAcq.\*** scripts get run before the primary address is added to the secondary. Any script starting with the prefix **PreAcq** is run, and the ordering is determined by the name of the file (*e.g.*, **PreAcq-1** would run before **PreAcq-2**).
- **PostAcq.\*** scripts get run after the primary address has been acquired by the secondary server. Execution rules are the same as with **PreAcq** scripts above.

- `PreRel.*` scripts get run just before the secondary server gives up the primary address. Execution rules are the same as with `PreAcq` scripts above.
- `PostRel.*` scripts get run after the primary address has been relinquished by the secondary server. Execution rules are the same as with `PreAcq` scripts above.

You may have as many of each script as you want, and they may perform any scriptable tasks. For example, you might use the scripts to start a service after acquiring the primary's address, and then stop the service when the primary comes back.

We've built a template directory containing simple scripts that are easily customized:

[Suffield IPFailover Config Template](#)

### 1.3.3 Peering Failover

It is possible to set pairs of machines up in a peering configuration such that each acts as the backup for the other. Simply add the requisite lines to `/etc/hostconfig` on both machines and they'll act as a backup for each other.

### 1.3.4 Using the Suffield Configs

We keep all of our script directories under version control. To use them, you must check out the repository for the primary host you're interested in onto the secondary server that backs it up.

**Note:** This document assumes you've set up basic IPFailover as described above. That means you've edited `/etc/hostconfig` and "vanilla" failover is working.

Check out the config from our Subversion repository, substituting the actual primary IP address for `<PRIMARY_IP>`:

```
cd /Library/IPFailover/

sudo svn checkout \
svn://svn.suffieldacademy.org/netadmin/trunk/software/failover/host_configs/<PRIMARY_IP>
```

This will check out the configuration directory named after the IP address and place a working copy in the `/Library/IPFailover` directory. You may change into this directory at any time and run `svn up` to merge in any updates to the configuration.