6

HEARTBEAT INTRODUCTION AND THEORY

This chapter introduces a software package called Heartbeat that gives you the ability to failover a resource from one computer to another. The following three chapters will explore Heartbeat and high-availability techniques in detail. Part III of this book will build on the techniques described here, and it will explain how to build a highly available cluster. Once you know how to use Heartbeat properly, you can deploy just about any service and make it highly available.

NOTE *A highly available system has no single points of failure. A single point of failure is a single system component that upon failing causes the whole system to fail.*

Heartbeat works like this: you tell Heartbeat which computer owns a particular resource (which computer is the *primary server*), and the other computer will automatically be the *backup server*. You then configure the

Heartbeat daemon running on the backup server to listen to the "heartbeats" coming from the primary server. If the backup server does not hear the primary server's heartbeat, it initiates a failover and takes ownership of the resource.

The Physical Paths of the Heartbeats

The Heartbeat program running on the backup server can check for heartbeats coming from the primary server over the normal Ethernet network connection, but normally Heartbeat is configured to work over a separate physical connection between the two servers. This separate physical connection can be either a serial cable or another Ethernet network connection (via a crossover cable[1] or mini hub, for example).

Heartbeat will work over one or more of these physical connections at the same time and will consider the primary node active as long as heartbeats are received on at least one of the physical connections. Figure 6-1 shows three physical connections, or paths, between the servers. The first path, the normal Ethernet network used to connect systems to each other on the network, is the least preferred for sending the heartbeats, because it will add extra traffic to your network (though this is a trivial load under normal circumstances). Your choice of whether to use one or more new serial or Ethernet connections will depend on your situation.

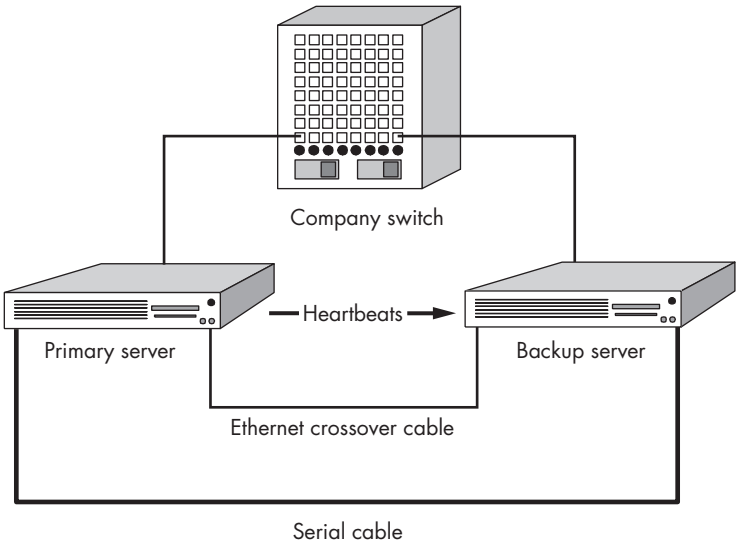

Figure 6-1: Physical paths for heartbeats

[1] A crossover cable is simpler and more reliable than a mini hub because it does not require external power.

NOTE *High-availability best practices dictate that heartbeats should travel over multiple independent communication paths.*[2] *This helps eliminate the communications path from being a single point of failure.*

Serial Cable Connection

A serial connection is slightly more secure than an Ethernet connection, because a hacker will not be able to run telnet, ssh, or rlogin over the serial cable if they break into one of the systems. (The serial cable is a simple crossover cable connected to the COM port on each system.) However, because serial cables are short,[3] the servers must be located near each other, usually in the same computer room.

Ethernet Cable Connection

Using a new Ethernet network (or Ethernet crossover cable) eliminates any distance limitation between the servers. It also allows you to synchronize the filesystems on the two servers (as described in Chapter 4) without placing any extra network traffic on your normal Ethernet network.

Using two physical paths to connect the primary and backup servers provides redundancy for heartbeat control messages and is therefore a requirement of a no-single-point-of-failure configuration. The two physical paths between the servers need not be of the same type; an Ethernet and a serial connection can be used together in the same configuration.

Partitioned Clusters and STONITH

For true redundancy, two physical connections should carry heartbeat control messages between the primary and backup server. These two physical connections will help prevent a situation where a network or cable failure causes both nodes to try and assume ownership of the same resources. This condition is known as a *split-brain* or *partitioned* cluster,[4] and it can have dire consequences if you are using two heartbeat nodes to control one physical device (such as a shared SCSI or Fibre Channel disk drive). To avoid this situation, take the following precautions:

- Create a redundant, reliable physical connection between heartbeat nodes (preferably using both a serial connection and an Ethernet connection) to carry heartbeat control messages.
- Allow for the ability to forcibly shut down one of the heartbeat nodes when a partitioned cluster is detected.

---

[2] In *Blueprints for High Availability,* Evan Marcus and Hal Stern define three different types of networks used in failover configurations: the Heartbeat network, the production network (for client access to cluster resources), and an administrative network (for system administrators to access the servers and do maintenance tasks).

[3] The original EIA-232 specification did not specify a distance limitation, but 50 feet has become the industry's de facto distance limit for normal serial communication. See the Serial HOWTO for more information.

[4] Sometimes the term *cluster fencing* or *i/o fencing* is used to describe what should happen when a cluster is partitioned. It means that the cluster must be able to build a fence between partitions and decide on which side of the fence the cluster resources should reside.

This second precaution has been dubbed "shoot the other node in the head," or STONITH. Using a special hardware device that can power off a node through software commands (sent over a serial or network cable), Heartbeat can implement a Stonith[5] configuration designed to avoid cluster partitioning. (See Chapter 9 for more information.)

NOTE *It is difficult to guarantee exclusive access to resources and avoid split-brain conditions when the primary and backup heartbeat servers are not in close proximity. You will very likely reduce resource reliability and increase system administration headaches if you try to use Heartbeat as part of your disaster recovery or business resumption plan over a wide area network (WAN). In the cluster solution described in this book, no Heartbeat pairs of servers need communicate over a WAN.*

Heartbeat Control Messages

In this chapter we will look at the three most basic heartbeat control messages[6] (three kinds of packets, if you are using an Ethernet network):

- Heartbeats or status messages
- Cluster transition messages
- Retransmission requests

Heartbeats

Heartbeats (sometimes called *status messages*) are broadcast, unicast, or multicast packets that are only about 150 bytes long. You control how often each computer broadcasts its heartbeat and how long the `heartbeat` daemon running on another node should wait before assuming something has gone wrong.

Cluster Transition Messages

The two most prevalent cluster transition messages are ip-request and ip-request-resp. These messages are relatively rare and contain the conversation between `heartbeat` daemons when they want to move a resource from one computer to another.

When you repair the primary server and it comes back online, it uses ip-request to ask the backup server to release the resource it took ownership of when the primary server failed. The backup server then shuts off the service and replies with an ip-request-resp message to inform the primary server that it no longer owns the resource. When the primary server receives this ip-request-resp, it starts up the service and offers it to the client computers again (it takes back ownership of the resource).

[5] Although an acronym at birth, this term has moved up in stature to become, by all rights, a word—it will be used as such throughout the remainder of this book.

[6] For a complete list of heartbeat control message types see this website: http://www.linux-ha.org/comm/Heartbeat-msgfmt.html.

Retransmission Requests

The rexmit-request (or ns_rexmit) message—a request for a retransmission of a heartbeat control message—is issued when one of the servers running heartbeat notices that it is receiving heartbeat control messages that are out of sequence. (Heartbeat daemons use sequence numbers to ensure packets are not dropped or corrupted.) Heartbeat daemons will only ask for a retransmission of a heartbeat control message (with no sequence number, hence the "ns" in ns_rexmit) once every second, to avoid flooding the network with these retry requests when something goes wrong.

Ethernet Heartbeat Control Messages

All three of these heartbeat control messages are sent using the UDP protocol to either the port number specified in the `/etc/ha.d/ha.cf` file, or to the multicast address specified in this same configuration file (when using Ethernet).

Currently Heartbeat does not support more than two nodes.[7] More than one pair of heartbeat servers can share the same Ethernet network connection and exchange heartbeats and heartbeat control messages, but each of these pairs of heartbeat servers must use a unique UDP port number as specified in the `/etc/ha.d/ha.cf` file, or a unique unicast or multicast address.

Security and Heartbeat Control Messages

In addition to using a numbering sequence to recover from dropped or corrupted packets, Heartbeat digitally signs each packet using either a 128-bit hashing algorithm called MD5 (see RFC[8] 1321), or the even more secure 160-bit HMAC-SHA1 (see RFC 2104). (You enter the same encryption password for either of these methods on both the primary and the backup heartbeat nodes.)

NOTE *The Heartbeat developers recommend that you use one of these encryption methods even on private networks to protect Heartbeat from an attacker (spoofed packets or a packet replay attack).*

How Client Computers Access Resources

Normally client computers know the name of the server offering the resource they want to use (mail.mydomain.com or www.mydomain.com, for example), and they use the Domain Name System (DNS) to look up the proper IP

[7] To be more accurate, Heartbeat will only allow two nodes to share haresources entries (see Chapter 8 for more information about the proper use of the haresources file). In this book we are focusing on building a pair of high-availability LVS-DR directors to achieve high-availability clustering.

[8] The Requests for Comments (RFCs) can be found at http://www.rfc-editor.org.

address of the server. Once the routing of the packet through the Internet or WAN is done, however, this IP address has to be converted into a physical network card address called the Media Access Control (MAC) address.

At this point, the router or the locally connected client computers use the Address Resolution Protocol (ARP) to ask: "Who owns this IP address?" When the computer using the IP address responds, the router or client computer adds the IP address and its corresponding MAC address to a table in its memory called the ARP table so it won't have to ask each time it needs to use the IP address. After a few minutes, most computers let unused ARP table entries expire, to make sure they do not hold unused (or inaccurate) addresses in memory.[9]

Failover Using IP Address Takeover (IPAT)

In order to move a resource (a service or daemon and its associated IP address) from one computer to another, we need a way to move the IP address from the primary computer to the backup computer. The method normally used by Heartbeat is called IP address takeover (sometimes called IPAT). To accomplish IPAT, Heartbeat uses secondary IP addresses (formerly called *IP aliases* in older Linux kernels) and gratuitous ARP broadcasts.

FAILING OVER MULTIPLE IP ALIASES

In theory, you can add as many secondary IP addresses (or IP aliases) as your network IP address numbering scheme will allow (depending upon your kernel version). However, if you want Heartbeat to move more than a few IP addresses from a primary server to a backup server when the primary server fails, you may want to use a different method for IP address failover than the method described in this chapter. In fact, Heartbeat versions prior to version 0.4.9 were limited to eight IP addresses per network interface card (NIC) unless you used this method. (See the file `/usr/share/doc/packages/heartbeat/faqntips.html` after installing the Heartbeat RPM for instructions on how to configure your router for this alternative method.)

Secondary IP Addresses and IP Aliases

Secondary IP addresses and *IP aliases* are two different methods for adding multiple IP addresses to the same physical network card. The first IP address (also called the *primary address*) is added to the NIC at boot time. Additional IP addresses are then added by Heartbeat based on entries in the haresources configuration file (we'll discuss this file in detail in Chapter 8)—additional IP addresses are either IP aliases or secondary IP addresses. As of this writing, the Linux kernel supports both IP aliases and secondary IP addresses, though IP aliases are deprecated in favor of secondary IP addresses.

[9] For more information see RFC 826, "Ethernet Address Resolution Protocol."

NOTE IP aliasing *(sometimes called* network interface aliasing*) is a standard feature of the Linux kernel when standard IPv4 networking is configured in the kernel. Older versions of the kernel required you to configure IP alias support as an option in the kernel.*

IP ALIASES AND SOURCE ADDRESS

Whether you use secondary IP addresses or IP aliases, the source address used in packets that originate from the server (connections created from the server) will use the primary IP address on the NIC. You can force the kernel to make the source address appear to be one of the secondary IP addresses or IP aliases associated with the NIC by using the `ip` command and the undocumented `src` option. For example:

```
#ip route add default via 209.100.100.254 src 209.100.100.3 dev eth0
```

In this example, `209.100.100.254` is the gateway, and `209.100.100.3` is the secondary IP address (or IP alias) you want to use as a source address for outgoing packets. This method is used by the `ipsrcaddr` resource script included with the Heartbeat package. (You can also use the command `ip route get 209.100.100.10` to see which interface and source address is returned by the kernel's routing table for a particular IP address.)

Note: Forcing the kernel to use a secondary IP address or IP alias for the return address in the packet header does not work if both sides of an active/active Heartbeat pair of servers use the ipsrcaddr resource, or if different resources need to use different source addresses.

By using secondary IP addresses or IP aliases you can offer a service such as sendmail on one IP address and offer another service, like HTTP, on another IP address, even though these two IP addresses are really owned by the same computer (one physical network card at one MAC address).

When you use Heartbeat to offer services on a secondary IP address (or IP alias) the service is owned by the server (meaning the server is the active, or primary, node) and the server also owns the IP addresses used to access the service. The backup node must not be using this secondary IP address (or IP alias). When the primary node fails, and the service should be offered by the backup server, the backup server will not only need to start the service or daemon, it will also need to add the proper secondary IP address or IP alias to one of its network cards.

A diagram of this two-node Heartbeat cluster configuration using an Ethernet network to carry the heartbeat packets is shown in Figure 6-2.

In Figure 6-2 the IP address `209.100.100.2` is the primary IP address of the primary server, and it never needs to move to the backup server. The backup server's primary IP address is `209.100.100.5`, and this IP address will likewise never need to move to another network card. However, the IP addresses `209.100.100.3` and `209.100.100.4` are each associated with a particular service running on the primary server. If the primary server goes down, these IP addresses need to move to the backup server, as shown in Figure 6-3.

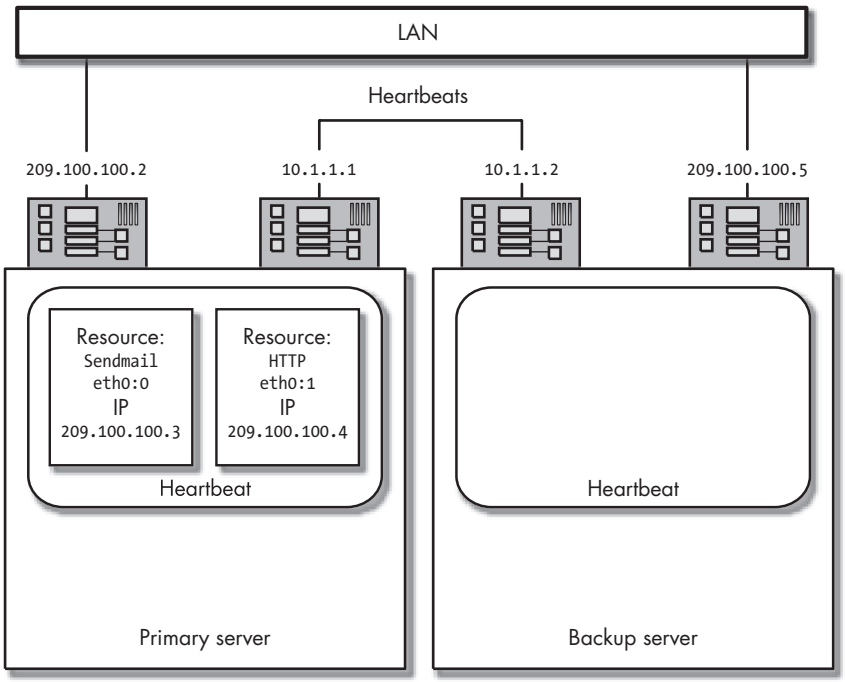

Figure 6-2: A basic Heartbeat configuration

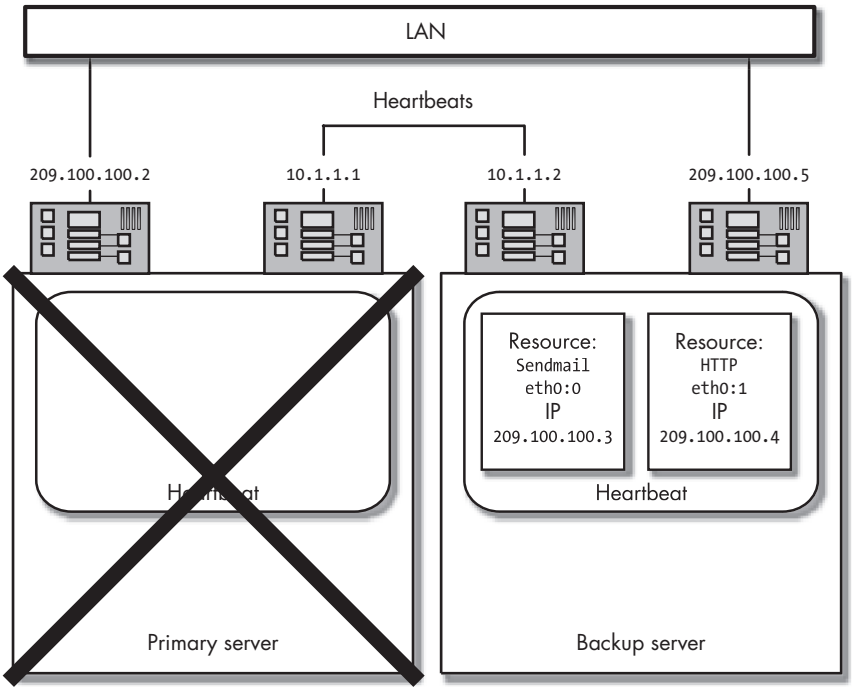

Figure 6-3: The same basic Heartbeat configuration after failure of the primary server

Ethernet NIC Device Names

The Linux kernel assigns the physical Ethernet interfaces names such as `eth0`, `eth1`, `eth2`, and so forth. These names are assigned either at boot time or when the Ethernet driver for the interface is loaded (if you are using a modular kernel), and they are based on the configuration built during the system installation in the file `/etc/modules.conf` (or `/etc/conf.modules` on older versions of Red Hat Linux). You can use the following command to see the NIC driver, interrupt (IRQ) address, and I/O (base) address assigned to each PCI network interface card (assuming it is a PCI card) that was recognized during the boot process:

```
#lspci -v | less
```

You can then check this information against the interface configuration using the `ifconfig` command:

```
#ifconfig -a | less
```

This should help you determine which `eth` number is assigned to a particular physical network card by the kernel `eth` numbering scheme. (See Appendix C for more information about NICs.)

Secondary IP Address Names

Secondary IP addresses are added after a primary IP address has already been configured for a NIC. The primary IP address can be assigned to a NIC using the `ifconfig` command (this is normally how Linux distributions assign IP addresses at boot time) or by using the `ip` command. Secondary IP addresses, however, can only be added using the `ip` command. When Heartbeat needs to add a secondary IP address to a NIC it uses the script `IPaddr2` (included with the Heartbeat distribution) to run the proper `ip` command.

Both the primary and the secondary IP addresses can be viewed with this command:

```
#ip addr sh
```

NOTE *Secondary IP addresses are not shown by the* `ifconfig` *command.*

Creating and Deleting Secondary IP Addresses with the ip Command

Creating and deleting secondary IP addresses in Linux is easy. To create (add) a secondary IP address for the `eth0` NIC, use this command:

```
#ip addr add 209.100.100.3/24 broadcast 209.100.100.255 dev eth0
```

In this example, we are assuming that the eth0 NIC already has an IP address on the 209.100.100.0 network, and we are adding 209.100.100.3 as an additional (secondary) IP address associated with this same NIC. To view the IP addresses configured for the eth0 NIC with the previous command, enter this command:

```
#ip addr sh dev eth0
```

To remove (delete) this secondary IP address, enter this command:

```
#ip addr del 209.100.100.3/24 broadcast 209.100.100.255 dev eth0
```

NOTE *The* ip *command is provided as part of the IProute2 package. (The RPM package name is iproute.)*

Fortunately, you won't need to enter these commands to configure your secondary IP addresses—Heartbeat does this for you automatically when it starts up and at failover time (as needed) using the IPaddr2 script.

IP Aliases

As previously mentioned, you only need to use one method for assigning IP addresses under Heartbeat's control: secondary IP addresses or IP aliases. If you are new to Linux, you should use secondary IP addresses as described in the previous sections and skip this discussion of IP aliases.

You add IP aliases to a physical Ethernet interface (that already has an IP address associated with it) by running the ifconfig command. The alias is specified by adding a colon and a number to the interface name. The first IP alias associated with the eth0 interface is called eth0:0, the second is called eth0:1, and so forth. Heartbeat uses the IPaddr script to create IP aliases.

Creating and Deleting IP Aliases with the ifconfig Command

In our previous example, we were using IP address 209.100.100.2 with a network mask of 255.255.255.0 on the eth0 Ethernet interface. To manually add IP alias 209.100.100.3 to the same interface, use this command:

```
#ifconfig eth0:0 209.100.100.3 netmask 255.255.255.0 up
```

You can then view the list of IP addresses and IP aliases by typing the following:

```
#ifconfig –a
```

or simply

```
#ifconfig
```

This command will produce a listing that looks like the following:

```
eth0      Link encap:Ethernet su HWaddr 00:99:5F:0E:99:AB
          inet addr:209.100.100.2  Bcast:209.100.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:976 errors:0 dropped:0 overruns:0 frame:0
          TX packets:730 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x1400

eth0:0    Link encap:Ethernet  HWaddr 00:99:5F:0E:99:AB
          inet addr:209.100.100.3  Bcast:209.100.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          Interrupt:11 Base address:0x1400
```

From this report you can see that the MAC addresses (called `HWaddr` in this report) for `eth0` and `eth0:0` are the same. (The interrupt and base addresses also show that these IP addresses are associated with the same physical network card.) Client computers locally connected to the computer can now use an ARP broadcast to ask "Who owns IP address `209.100.100.3`?" and the primary server will respond with "I own IP `209.100.100.3` on MAC address `00:99:5F:0E:99:AB`."

NOTE *The* `ifconfig` *command does not display the secondary IP addresses added with the* `ip` *command.*

IP aliases can be removed with this command:

```
#ifconfig eth0:0 down
```

This command should not affect the primary IP address associated with `eth0` or any additional IP aliases (they should remain up).

NOTE *Use the preceding command to see whether your kernel can properly support IP aliases. If this command causes the primary IP address associated with this network card to stop working, you need to upgrade to a newer version of the Linux kernel. Also note that this command will not work properly if you attempt to use IP aliases on a different subnet.*[10]

Offering Services

Once you are sure a secondary IP address or IP alias can be added to and removed from a network card on your Linux server without affecting the primary IP address, you are ready to tell Heartbeat which services it should offer, and which secondary IP address or IP alias it should use to offer the services.

[10] The "secondary" flag will not be set for the IP alias if it is on a different subnet. See the output of the command `ip addr` to find out whether this flag is set. (It should be set for Heartbeat to work properly.)

NOTE *Secondary IP addresses and IP aliases used by highly available services should always be controlled by Heartbeat (in the* `/etc/ha.d/haresouces` *file as described later in this chapter and in the next two chapters). Never use your operating system's ability to add IP aliases as part of the normal boot process (or a script that runs automatically at boot time) on a Heartbeat server. If you do, your server will incorrectly claim ownership of an IP address when it boots. The backup node should always be able to take control of a resource along with its IP address and then reset the power to the primary node without worrying that the primary node will try to use the secondary IP address as part of its normal boot procedure.*

Gratuitous ARP (GARP) Broadcasts

As mentioned previously, client computers normally use the Address Resolution Protocol (ARP) to figure out which hardware address owns a particular IP address, and then they store this address in an ARP table. The Heartbeat program uses a little trick, called Gratuitous ARP (GARP) broadcasts, to forcibly update these client computer ARP tables with a new hardware (MAC) addresses when the primary server fails, effectively convincing the client computers to talk to the backup server.[11]

GARP broadcasts[12] are just sneaky ARP broadcasts (broadcasts, remember, are only seen by locally connected nodes). The GARP broadcast asks every node connected to the network, "Who owns this IP address?" when, in fact, the ARP request packet header has a source (or reply) IP address equal to the requested IP address. This forces all nodes connected to the network to update their ARP tables with the new source address.

NOTE *As of Heartbeat version 0.4.9.1 the* `send_arp` *program included with Heartbeat uses both ARP request and ARP reply packets when sending GARPs (*`send_arp` *version 1.6). If you experience problems with IP address failover on older versions of Heartbeat, try upgrading to the latest version of Heartbeat.*

Heartbeat uses the `/usr/lib/heartbeat/send_arp` program (formerly `/etc/ha.d/resource.d/send_arp`) to send these specially crafted GARP broadcasts. You can use this same program to build a script that will send GARPs. The following is an example script (called `iptakeover`) that uses `send_arp` to do this.

[11] To use the Heartbeat IP failover mechanism with minimal downtime and minimal disruption of the highly available services you should set the ARP cache timeout values on your switches and routers to the shortest time possible. This will increase your ARP broadcast traffic, however, so you will have to find the best trade-off between ARP timeout and increased network traffic for your situation. (Gratuitous ARP broadcasts should update the ARP table entries even before they expire, but if they are missed by your network devices for some reason, it is best to have regular ARP cache refreshes.)

[12] Gratuitous ARPs are briefly described in RFC 2002, "IP Mobility Support." Also see RFC 826, "Ethernet Address Resolution Protocol."

```
#!/bin/bash
#
# iptakeover script
#
#  Simple script to take over an IP address.
#
#  Usage is "iptakeover {start|stop|status}"
#
# SENDARP is the program included with the Heartbeat program that
# sends out an ARP request. Send_arp usage is:
#
#
SENDARP="/usr/lib/heartbeat/send_arp"
#
# REALIP is the IP address for this NIC on your LAN.
#
REALIP="299.100.100.2"
#
# ROUTERIP is the IP address for your router.
#
ROUTER_IP="299.100.100.1"
#
# SECONDARYIP is the first IP alias for a service/resource.
#
SECONDARYIP1="299.100.100.3"
# or
#IPALIAS1="299.100.100.3"

#
# NETMASK is the netmask of this card.
#
NETMASK="24"
# or
# NETMASK="255.255.255.0"
#
BROADCAST="299.100.100.0"
#
# MACADDR is the hardware address for the NIC card.
# (You'll find it using the command "/sbin/ifconfig")
#
MACADDR="091230910990"
case $1 in
start)
    # Make sure our primary IP is up
    /sbin/ifconfig eth0 $REALIP up
```

```
    # Associate the virtual IP address with this NIC
    /sbin/ip addr add $SECONDARYIP1/$NETMASK broadcast $BROADCAST dev eth0
    # Or, to create an IP alias instead of secondary IP address, use the
    command:
    # /sbin/ifconfig eth0:0 $IPALIAS1 netmask $NETMASK up

    # Create a new default route directly to the router
    /sbin/route add default gw $ROUTER_IP

    # Now send out 5 Gratuitous ARP broadcasts (ffffffffffff)
    # at two second intervals to tell the local computers to update
    # their ARP tables.

    $SENDARP -i 2000  -r 5 eth0 $SECONDARYIP1 $MACADDR $SECONDARYIP1 ffffffffffff
;;
stop)
    # Take down the secondary IP address for the service/resource.
    /sbin/ip addr del $SECONDARYIP1/$NETMASK broadcast $BROADCAST dev eth0
    # or
    /sbin/ifconfig eth0:0 down
;;
status)
    # We check to see if we own the IPALIAS.
    OWN_ALIAS=`ifconfig | grep $SECONDARYIP1`
    if [ "$OWN_ALIAS" != "" ]; then
    echo "OK"
    else
    echo "DOWN"
    fi
;;
# End of the case statement.
    esac
```

NOTE *You do not need to use this script. It is included here (and on the CD-ROM) to demonstrate exactly how Heartbeat performs GARPs.*

The important line in the above code listing is marked in boldface.

This command runs `/usr/lib/heartbeat/send_arp` and sends an ARP broadcast (to hexadecimal IP address `ffffffffffff`) with a source and destination address equal to the secondary IP address to be added to the backup server.

You can use this script to find out whether your network equipment supports IP address failover using GARP broadcasts. With the script installed on two Linux computers (and with the MAC address in the script changed to the appropriate address for each computer), you can move an IP address back and forth between the systems to find out whether Heartbeat will be able to do the same thing when it needs to failover a resource.

NOTE *When using Cisco equipment, you should be able to log on to the router or switch and enter the command* `show arp` *to watch the MAC address change as the IP address moves between the two computers. Most routers have similar capabilities.*

Resource Scripts

All of the scripts under Heartbeat's control are called resource scripts. These scripts may include the ability to add or remove a secondary IP address or IP alias, or they may include packet-handling rules (see Chapter 2) in addition to being able to start and stop a service. Heartbeat looks for resource scripts in the Linux Standards Base (LSB) standard init directory (`/etc/init.d`) and in the Heartbeat `/etc/ha.d/resource.d` directory.

Heartbeat should always be able to start and stop your resource by running your resource script and passing it the `start` or `stop` argument. As illustrated in the `iptakeover` script, this can be accomplished with a simple `case` statement like the following:

```
#!/bin/bash
case $1 in
start)
    commands to start my resource
    ;;
stop)
    commands to stop my resource
    ;;
status)
    commands to test if I own the resource
    ;;
*)
    echo "Syntax incorrect. You need one of {start|stop|status}"
    ;;
esac
```

The first line, `#!/bin/bash`, makes this file a bash (Bourne Again Shell) script. The second line tests the first argument passed to the script and attempts to match this argument against one of the lines in the script containing a closing parenthesis. The last match, `*)`, is a wildcard match that says, "If you get this far, match on anything passed as an argument." So if the program flow ends up executing these commands, the script needs to complain that it did not receive a `start`, `stop`, or `status` argument.

Status of the Resource

Heartbeat should always know which computer owns the resource or resources being offered. When you write a script to start or stop a resource, it is important to write the script in such a way that it will accurately determine whether the service is currently being offered by the system. If it is, the script

should respond with the word OK, Running, or running when passed the status argument—Heartbeat requires one of these three responses if the service is running. What the script returns when the service is not running doesn't matter to Heartbeat—you can use DOWN or STOPPED, for example. However, do *not* say not running or not OK for the stopped status. (See the "Using init Scripts as Heartbeat Resource Scripts" section later in this chapter for more information on proposed standard exit codes for Linux.)

NOTE *If you want to write temporary scratch files with your resource script and have Heartbeat always remove these files when it first starts up, write your files into the directory* /var/lib/heartbeat/rsctmp *(on a standard Heartbeat installation on Linux).*

Resource Ownership

So how do you write a script to properly determine if the machine it is running on currently owns the resource? You will need to figure out how to do this for your situation, but the following are two sample tests you might perform in your resource script.

Testing for Resource Ownership—Is the Daemon Running?

Red Hat Linux and SuSE Linux both ship with a program called pidof that can test to see if a daemon is running. The pidof program finds the process identifier (PID) of a running daemon when you give it the name of the program file that was used to start the daemon.

When you run the pidof script from within your resource script you should always pass it the full pathname of the daemon you want to search for in the process table. For example, to determine the PID of the sendmail program, you would use this command:

```
#pidof /usr/sbin/sendmail
```

If sendmail is running, its PID number will be printed, and pidof exits with a return code of zero. If sendmail is not running, pidof does not print anything, and it exits with a nonzero return code.

NOTE *See the* /etc/init.d/functions *script distributed with Red Hat Linux for an example use of the pidof program.*

A sample bash shell script that uses pidof to determine whether the sendmail daemon is running (when given the status argument) might look like this:

```
#!/bin/bash
#
case $1 in
status)
```

```
    if /sbin/pidof /usr/sbin/sendmail >/dev/null; then
        echo "OK"
    else
        echo "DOWN"
    fi
;;
esac
```

If you place these commands in a file, such as `/etc/ha.d/resource.d/myresource`, and `chmod` the file to 755, you can then run the script with this command:

```
#/etc/ha.d/resource.d/myresource status
```

If sendmail is running, the script will return the word `OK`; if not, the script will say `DOWN`. (You can stop and start the `sendmail` daemon for testing purposes by running the script `/etc/init.d/myresource` and passing it the `start` or `stop` argument.)

Testing for Resource Ownership—Do You Own the IP Address?

If you use Heartbeat's built-in ability to automatically failover secondary IP addresses or IP aliases, you do not need to worry about this second test (see the discussion of the `IPaddr2` resource script and the discussion of resource groups in Chapter 8). However, if you are using the `iptakeover` script provided on the CD-ROM, or if you are building a custom configuration that makes changes to network interfaces or the routing table, you may want to check to see if the system really does have the secondary IP address currently configured as part of your script's status test.

To perform this check, the script needs to look at the current secondary IP addresses configured on the system. We can modify the previous script and add the `ip` command to show the secondary IP address for the `eth0` NIC as follows:

```
#!/bin/bash
#
case $1 in
status)
        if /sbin/pidof /usr/sbin/sendmail >/dev/null; then
        ipaliasup=`/sbin/ip addr show dev eth0 | grep 200.100.100.3`
        if [ "$ipaliasup" != "" ]; then
        echo "OK"
        exit 0
        fi
        fi
        echo "DOWN"
;;
esac
```

Note that this script will only work if you use secondary IP addresses. To look for an ip alias, modify this line in the script:

```
ipaliasup=`/sbin/ip addr show dev eth0 | grep 200.100.100.3`
```

to match the following:

```
ipaliasup=`/sbin/ifconfig eth0 | grep 200.100.100.3`
```

This changed line of code will now check to see if the IP alias `200.100.100.3` is associated with the interface `eth0`. If so, this script assumes the interface is up and working properly, and it returns a status of `OK` and exits from the script with a return value of `0`.

NOTE *Testing for a secondary IP address or an IP alias should only be added to resource scripts in special cases; see Chapter 8 for details.*

Testing a Resource Script

You can test to make sure the Heartbeat system will work properly with your newly created resource script by running the following command:

```
#/usr/lib/heartbeat/ResourceManager status <resource-name>
```

where <*resource-name*> is the name of your newly created resource script in the `/etc/rc.d/init.d` or `/etc/ha.d/resource.d` directory. Then tell your shell to display the value of your resource script's return code with this command:

```
#echo $?
```

The `$?` must be the next command you type after running the script—otherwise the return value stored in the shell variable `?` will be overwritten.

If the return code is `3`, it means your script returned something other than `OK`, `Running`, or `running`, which means that Heartbeat thinks that it does not own the resource. If this command returns `0`, Heartbeat considers the resource active.

Once you have built a proper resource script and placed it in the `/etc/rc.d/init.d` directory or the `/etc/ha.d/resource.d` directory and thoroughly tested its ability to handle the `start`, `stop`, and `status` arguments, you are ready to configure Heartbeat.

Using init Scripts as Heartbeat Resource Scripts

Most of the `/etc/init.d` scripts (the scripts used at system boot time by the init process) will already properly handle the `start`, `stop`, and `status` arguments. In fact, the Linux Standard Base Specification (see http://www.linuxbase.org)

states that in future releases of Linux, all Linux init scripts running on all versions of Linux that support LSB must implement the following: `start`, `stop`, `restart`, `reload`, `force-reload`, and `status`.

The LSB project also states that if the `status` argument is sent to an init script it should return `0` if the program is running.

For the moment, however, you need only rely on the convention that a `status` command should return the word `OK`, or `Running`, or `running` to standard output (or "standard out").[13] For example, Red Hat Linux scripts, when passed the `status` argument, normally return a line such as the following example from the `/etc/rc.d/init.d/sendmail status` command:

```
sendmail (pid 4511) is running...
```

Heartbeat looks for the word `running` in this output, and it will ignore everything else, so you do not need to modify Red Hat's init scripts to use them with Heartbeat (unless you want to also add the test for ownership of the secondary IP address or IP alias).

However, once you tell Heartbeat to manage a resource or script, you need to make sure the script does not run during the normal boot (init) process. You can do this by entering this command:

```
#chkconfig --del <scriptname>
```

where <*scriptname*> is the name of the file in the `/etc/init.d` directory. (See Chapter 1 for more information about the `chkconfig` command.)

The init script must not run as part of the normal boot (init) process, because you want Heartbeat to decide when to start (and stop) the daemon. If you start the Heartbeat program but already have a daemon running that you have asked Heartbeat to control, you will see a message like the following in the `/var/log/messages` file:

```
WARNING: Non-idle resources will affect resource takeback.
```

If you see this error message when you start Heartbeat, you should stop all of the resources you have asked Heartbeat to manage, remove them from the init process (with the `chkconfig --del` <*scriptname*> command), and then restart the Heartbeat daemon (or reboot) to put things in a proper state.

[13] Programs and shell scripts can return output to standard output or to standard error. This allows programmers to control, through the use of redirection symbols, where the message will end up—see the "REDIRECTION" section of the bash man page for details.

Heartbeat Configuration Files

Heartbeat uses three configuration files:

`/etc/ha.d/ha.cf`

Specifies how the `heartbeat` daemons on each system communicate with each other.

`/etc/ha.d/haresources`

Specifies which server should normally act as the primary server for a particular resource and which server should own the IP address that client computers will use to access the resource.

`/etc/ha.d/authkeys`

Specifies how the Heartbeat packets should be encrypted.

NOTE *The* `ha` *in these scripts stands for high availability. (The name "haresources" is not related to bunny rabbits.)*

In Conclusion

This chapter has introduced the Heartbeat package and the theory behind properly deploying it. To ensure client computers always have access to a resource, use the Heartbeat package to make the resource highly available. A highly available resource does not fail even if the computer it is running on crashes.

In Part III of this book I'll describe how to make the cluster load-balancing resource highly available. A cluster load-balancing resource that is highly available is the cornerstone of a cluster that is able to support your enterprise.

In the next few chapters I'll focus some more on how to properly use the Heartbeat package and avoid the typical pitfalls you are likely to encounter when trying to make a resource highly available.