getting your interfaces to talk

Configuring Network Interface Cards

Lance Spitzner

Last Modified: 17 August, 1999

This article is the first of a two part series. In this first article we will cover how to configure, troubleshoot, and modify system interfaces. The second article will cover static routing tables for systems with two or more interfaces. In both articles I will be focusing on TCP/IP in a Ethernet environment.

Interfaces

Network Interface Cards are what allow your system to talk to the network. When they don?t work, neither do you. I will cover how to configure, troubleshoot, and modify your interfaces. I will not be covering routing issues, that will follow in the next article. My goal here is to get your interface up and properly running.

The first place to start is installing and testing the hardware. Once you have installed the hardware, SPARC systems can be tested at the EPROM level to verify the network interface cards. Use the manual that accompanies the interface card on how to test that specific card. Solaris x86 is a little different, as there is no true EPROM, and the drivers are different. However, Solaris x86 2.6 is Plug and Play compatible, and I have had fairly good luck adding network interface cards.

Once you have confirmed at the hardware and driver level that everything works, the fun can begin. The place to start is the ifconfig command. This powerful command allows you configure and modify your interfaces in real time. However, any modifications made with ifconfig are not permanent. When the system reboots, it will default to its previous configuration. I will first show you how to make all modifications with the ifconfig command. The second half of this article will cover making these modifications permanent by modifying the proper configuration files.

ifconfig

ifconfig -a will show you which interfaces are currently installed and active. Remember, just because you added the physical network interface card does NOT mean it is active. If you do an ifconfig before you have configured the device, the interface will not show up. Once configured however, the typical output of the ifconfig -a command would look like this:

```
lo0: flags=849<UP,LOOPBACK,RUNNING,MULTICAST> mtu 8232
            inet 127.0.0.1 netmask ff000000
hme0: flags=863<UP,BROADCAST,NOTRAILERS,RUNNING,MULTICAST> mtu 1500
          inet 192.168.1.132 netmask ffffff00 broadcast 192.168.1.255
          ether 8:0:20:9c:6b:2d
```

Here we see two interfaces, lo0 and hme0. lo0 is the standard loopback interface found on all systems. hme0 is a 10/100 Mbps interface. All hme interfaces are 10/100 Mbps, all le interfaces are 10 Mbps, all qe interface are quad 10 Mbps, and qfe interfaces are quad 10/100 Mbps. There are three lines of information about the interface. The first line is about the TCP/IP stack. For the interface hme0, we see the system is up, running both broadcast and multicast, with a mtu (maximum transfer unit) of 1500 bytes, standard for an Ethernet LAN. Notrailers is a flag no longer used, but kept for backwards compatibility reasons.

The second line is about the IP addressing. Here we see the IP address, netmask in hexadecimal format, and the broadcast address. The third line is the MAC address. Unlike most interfaces, Sun Microsystems?s interfaces derive the MAC addressing from the NVRAM, not the interface itself. Thus, all the interfaces on a single SPARC system will have the same MAC address. This does not cause a problem in routing, since most NICs are always on a different network. Note, you must be root to see the MAC address with the ifconfig command, any other user will only see the first two lines of information.

The first step in bringing up an interface is "plumbing" the interface. By plumbing, we are implementing the TCP/IP stack. We will use the above interface, hme0, as an example. Lets say we had just physically added this network interface card and rebooted, now what? First, we plumb the device with the plumb command.

```
ifconfig hme0 plumb
```

This sets up the streams needed for TCP/IP to use the device. However, the stack has not been configured as you can see below.

```
hme0: flags=842<BROADCAST,RUNNING,MULTICAST> mtu 1500
           inet 0.0.0.0 netmask 0
           ether 8:0:20:9c:6b:2d
```

The next step is to configure the TCP/IP stack. We configure the stack by adding the IP address, netmask, and then telling the device it is up. All this can be down in one command, as seen below.

```
 homer #ifconfig hme0 192.168.1.132 netmask 255.255.255.0 up
```

This single command configures the entire device. Notice the up command, which initializes the interface. The interface can be in one of two states, up or down. When an interface is down, the system does not attempt to transmit messages through that interface. A down interface will still show with the ifconfig command, however it will not have the word "up" on the first line.

Virtual Interfaces

Before moving on to the configuration files, I would first like to cover virtual interfaces. A virtual interface is one or more logical interfaces assigned to an already existing interface. Solaris can have up to 255 virtual interfaces assigned to a single interface.

Once again, lets take the interface hme0 as an example. We have already covered how to configure this device. However, lets say the device is on a VLAN (virtual LAN) with several networks sharing the same wire. We can configure the device hme0 to answer to another IP address, say 172.20.15.4. To do so, the command would be the same as used for hme0, except the virtual interface is called hme0:*, where * is the number you assign to the virtual interface. For example, virtual interface one would be hme0:1. The command to configure it looks as follows.

```
ifconfig hme0:1 172.20.15.4 netmask 255.255.0.0 up
```

Once you have configured the virtual interface, you can compare hme0 and hme0:1 with the ifconfig command.

```
hme0: flags=843<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
          inet 192.168.1.132 netmask ffffff00 broadcast 192.168.1.255
          ether 8:0:20:9c:6b:2d
hme0:1: flags=842<BROADCAST,RUNNING,MULTICAST> mtu 1500
          inet 172.20.15.4 netmask ffff0000 broadcast 172.20.255.255
```

Here you see the two devices, both of which are on the same physical device. Notice how the virtual interface hme0:1 has no MAC address, as this is the same device as hme0. We can repeat this process all the way up to hme0:255. The operating system and most applications will treat these virtual devices as totally independent devices.

Note, Matthew A. Domurat has identified a "bug" with Solaris 2.6. When working with virtual interfaces, Solaris 2.6 will randomly select one of the interfaces as its source address for every packet sent. These are the patches to fix this:
105786-05: SunOS 5.6: /kernel/drv/ip patch
105787-04: SunOS 5.6_x86: /kernel/drv/ip patch

Configuration Files

Now you know how to configure your network interface cards. Unfortunately, any modifications, additions, or deletions you make with ifconfig are only temporary, you will lose these configurations when you reboot. I will now discuss what files you have to configure to make these changes permanent.

The place to start is the file /etc/hostname.*, where * is the name of the interface. In the case of hme0, the file name is /etc/hostname.hme0. The virtual interface hme0:1 would have the file name /etc/hostname.hme0:1. This file has a single entry, the name of the interface. This name is used in the /etc/hosts file to resolve name to IP address.

The file /etc/hostname.* is critical, this is what causes the device to be plumbed. During the boot process, the /etc/rcS.d/network.sh file reads all the /etc/hostname.* files and plumbs the devices. Once plumbed, the devices are configured by reading the /etc/hosts and the /etc/netmasks file. By reading these two files, the device is configured for the proper IP and netmask, and brought to an up state. Lets take the device hme0 as an example. During the boot process, /etc/rcS.d/network.sh looks for any /etc/hostname.* files. It finds /etc/hostname.hme0, which contains the following entry.

```
homer
```

/etc/rcS.d/rootusr.sh looks in /etc/hosts and resolves the name homer with an IP address of 192.168.1.132. The device hme0 is now assigned this IP address. The script then looks at /etc/netmasks to find the netmask for that IP address. With this information, the startup script brings up interface hme0 with an IP address of 192.168.1.132 and a netmask of 255.255.255.0. It may seem redundant having the script review the netmask of a class C address. However, do not forget that, starting with 2.6, Solaris supports both classless routing and VLSM (Variable Length Subnet Masks), both of which I will discuss in my next article.

As you have seen in this example, there are three files that must be modified for every interface. The first is /etc/hostname.*, this is the file you create to designate the interface?s name. The second file is /etc/hosts, here you resolve the IP to the interface name. Last is /etc/netmasks, this is where you define the netmask of the IP address.

Troubleshooting

Once you have mastered the tricks to modifying your interfaces, troubleshooting should be easier. Several things I always look for when troubleshooting an interface. First, when working with an unfamiliar machine, often you do not know how many physical interfaces are on the machine. A quick way to tell is use dmesg, this will give you information on the physical hardware. Look for le0, qfe0, hme0, or qe0. These are the names assigned to the physical devices.

If an interface is not responding to the network, check to be sure it is the correct IP address and netmask. The ifconfig command is a quick and temporary way to change IP and netmask information for troubleshooting purposes. Mtu (maximum transfer unit) is another possibility. Some systems may have problems communicating due to fragmented packets. Changing the mtu size may solve that problem. You?ll notice that you did not have to set the mtu size in the examples above, these are set to defaults automatically, such as 1500 for Ethernet interfaces.

If that fails, try bringing the face down, then reinitializing it with the up command. If nothing else works, unplumb the device, then plumb it again. Basically, this reinstalls the TCP/IP stack.

Conclusion

Network Interface Cards are critical to your systems networking capability. Understanding the configuration of your interface(s) ensures your system?s productivity. Next month we will look at routing tables, and ensure that once your interfaces are configured and up, your packets will know where to go.

Author?s bio
Lance Spitzner enjoys learning by blowing up his Unix systems at home. Before this, he was an Officer in the Rapid Deployment Force, where he blew up things of a different nature. You can reach him at lance@honeynet.org .

Whitepapers / Publications