Quick HOWTO : Ch04 : Simple Network Troubleshooting

From Linux Home Networking

Contents

Other Linux Home Networking Topics

- Introduction to Networking
- Linux Networking
- Simple Network Troubleshooting
- Troubleshooting Linux with Syslog
- Installing Linux Software
- The Linux Boot Process
- Configuring the DHCP Server
- Linux Users and sudo
- Windows, Linux and Samba
- Sharing Resources with Samba
- Samba Security and Troubleshooting
- Linux Wireless Networking
- Linux Firewalls Using iptables
- Linux FTP Server Setup
- Telnet, TFTP and xinetd
- Secure Remote Logins and File Copying
- Configuring DNS
- Dynamic DNS
- The Apache Web Server
- Configuring Linux Mail Servers
- Monitoring Server Performance
- Advanced MRTG For Linux
- The NTP Server
- Network-Based Linux Installation
- Linux Software RAID
- Expanding Disk Capacity
- Managing Disk Usage with Quotas
- Remote Disk Access with NFS
- Configuring NIS
- Centralized Logins Using LDAP and RADIUS
- Controlling Web Access with Squid
- Modifying the Kernel to Improve Performance
- Basic MySQL Configuration

Introduction

You will eventually find yourself trying to fix a network related problem which usually appears in one of two forms. The first is slow response times from the remote server, and the second is a complete lack of connectivity. These symptoms can be caused by:

Sources of Network Slowness

- NIC duplex and speed incompatibilities
- Network congestion
- Poor routing
- Bad cabling
- Electrical interference
- An overloaded server at the remote end of the connection
- Misconfigured DNS (Covered in Chapter 18, "Configuring DNS" and Chapter 19, "Dynamic DNS")

Sources of a Lack of Connectivity

All sources of slowness can become so severe that connectivity is lost. Additional sources of disconnections are:

- Power failures
- The remote server or an application on the remote server being shut down.

We discuss how to isolate these problems and more in the following sections.

Doing Basic Cable and Link Tests

Your server won't be able to communicate with any other device on your network unless the NIC's "link" light is on. This indicates that the connection between your server and the switch/router is functioning correctly.

In most cases a lack of link is due to the wrong cable type being used. As described in Chapter 2, "Introduction to Networking", there are two types of Ethernet cables crossover and straight-through. Always make sure you are using the correct type.

Other sources of link failure include:

- The cables are bad.
- The switch or router to which the server is connected is powered down.
- The cables aren't plugged in properly.

If you have an extensive network, investment in a battery-operated cable tester for basic connectivity testing is invaluable. More sophisticated models in the market will be able to tell you the approximate location of a cable break and whether an Ethernet cable is too long to be used.

Testing Your NIC

It is always a good practice in troubleshooting to be versed in monitoring the status of your NIC card from the command line. The following sections introduce a few commands that will be useful.

Viewing Your Activated Interfaces

The ifconfig command without any arguments gives you all the active interfaces on your system. Interfaces will not appear if they are shut down:

```
[root@bigboy tmp]# ifconfig
```

Note: Interfaces will appear if they are activated, but have no link. We'll soon discuss how to determine the link status using commands.

Viewing All Interfaces

The ifconfig -a command provides all the network interfaces, whether they are functional or not. Interfaces that are shut down by the systems administrator or are nonfunctional will not show an IP address line and the word UP will not show in the second line of the output. This can be seen in the next examples:

- Shut Down Interface

```
wlan0   Link encap:Ethernet  HWaddr 00:06:25:09:6A:D7
        BROADCAST MULTICAST  MTU:1500  Metric:1
        RX packets:2924 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2287 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:100
        RX bytes:180948 (176.7 Kb)  TX bytes:166377 (162.4 Kb)
        Interrupt:10 Memory:c88b5000-c88b6000
```

- Active Interface

```
wlan0     Link encap:Ethernet  HWaddr 00:06:25:09:6A:D7
          inet addr:216.10.119.243  Bcast:216.10.119.255
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:2924 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2295 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:180948 (176.7 Kb)  TX bytes:166521 (162.6 Kb)
          Interrupt:10 Memory:c88b5000-c88b6000
```

DHCP Considerations

DHCP clients automatically give their NICs and IP address starting with 169.254.x.x until they can make contact with their DHCP server. When contact is made they reconfigure their IP addresses to the values provided by the DHC server. An interface with a 169.254.x.x address signifies a failure to communicate with the DHCP server. Check your cabling, routing and DHCP server configuration to rectify such a problem.

Testing Link Status from the Command Line

Both the mii-tool and ethtool commands command will provide reports on the link status and duplex settings for supported NICs.

When used without any switches, the mii-tool gives a very brief report. Use it with the -v switch because it provides more information on the supported autonegotiation speeds of the NIC and this can be useful in troubleshooting speed and duplex issues.

The ethtool command provides much more information than mii-tool and should be your command of choice, especially because mii-tool will be soon deprecated in Linux. In both of the following examples the NICs are operating at 100Mbps, full duplex and the link is ok.

Link Status Output from mii-tool

```
[root@bigboy tmp]# mii-tool -v

eth0: 100 Mbit, full duplex, link ok
  product info: Intel 82555 rev 4
  basic mode:   100 Mbit, full duplex
  basic status: link ok
  capabilities: 100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD
  advertising:  100baseTx-FD 100baseTx-HD 10baseT-FD 10baseT-HD flow-control
  link partner: 100baseTx-HD

[root@bigboy tmp]#
```

Link Status Output from ethtool

```
[root@bigboy tmp]# ethtool eth0

Settings for eth0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
        Advertised auto-negotiation: No
        Speed: 100Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 1
        Transceiver: internal
        Auto-negotiation: off
        Supports Wake-on: g
        Wake-on: g
        Current message level: 0x00000007 (7)
        Link detected: yes
[root@bigboy tmp]#
```

Viewing NIC Errors

Errors are a common symptom of slow connectivity due to poor configuration or excessive bandwidth utilization. They should always be corrected whenever possible. Error rates in excess of 0.5% can result in noticeable sluggishness.

Ifconfig Error Output

The ifconfig command also shows the number of overrun, carrier, dropped packet and frame errors.

```
wlan0     Link encap:Ethernet  HWaddr 00:06:25:09:6A:D7
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:2924 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2287 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:180948 (176.7 Kb)  TX bytes:166377 (162.4 Kb)
          Interrupt:10 Memory:c88b5000-c88b6000
```

ethtool Error Output

The ethtool command can provide a much more detailed report when used with the -S switch.

```
[root@probe-001 root]# ethtool -S eth0
NIC statistics:
     rx_packets: 1669993
     tx_packets: 627631
     rx_bytes: 361714034
     tx_bytes: 88228145
```

```
    rx_errors: 0
    tx_errors: 0
    rx_dropped: 0
    tx_dropped: 0
    multicast: 0
    collisions: 0
    rx_length_errors: 0
    rx_over_errors: 0
    rx_crc_errors: 0
    rx_frame_errors: 0
    rx_fifo_errors: 0
    rx_missed_errors: 0
    tx_aborted_errors: 0
    tx_carrier_errors: 0
    tx_fifo_errors: 0
    tx_heartbeat_errors: 0
    tx_window_errors: 0
    tx_deferred: 0
    tx_single_collisions: 0
    tx_multi_collisions: 0
    tx_flow_control_pause: 0
    rx_flow_control_pause: 0
    rx_flow_control_unsupported: 0
    tx_tco_packets: 0
    rx_tco_packets: 0
[root@probe-001 root]#
```

netstat Error Output

The netstat command is very versatile and can provide a limited report when used with the -i switch. This is useful for systems where mii-tool or ethtool are not available.

```
[root@bigboy tmp]# netstat -i
Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500   0 18976655      2      0      0 21343152    142      0      3 BMRU
eth1       1500   0   855154      0      0      0 15196620      0      0      0 BMRU
lo        16436   0  1784272      0      0      0  1784272      0      0      0 LRU
[root@bigboy tmp]#
```

Possible Causes of Ethernet Errors

Collisions: Signifies when the NIC card detects itself and another server on the LAN attempting data transmissions at the same time. Collisions can be expected as a normal part of Ethernet operation and are typically below 0.1% of all frames sent. Higher error rates are likely to be caused by faulty NIC cards or poorly terminated cables.

Single Collisions: The Ethernet frame went through after only one collision

Multiple Collisions: The NIC had to attempt multiple times before successfully sending the frame due to collisions.

CRC Errors: Frames were sent but were corrupted in transit. The presence of CRC errors, but not many collisions usually is an indication of electrical noise. Make sure that you are using the correct type of cable, that the cabling is undamaged and that the connectors are securely fastened.

Frame Errors: An incorrect CRC and a non-integer number of bytes are received. This is usually the result of collisions or a bad Ethernet device.

FIFO and Overrun Errors: The number of times that the NIC was unable of handing data to its memory buffers because the data rate the capabilities of the hardware. This is usually a sign of excessive traffic.

Length Errors: The received frame length was less than or exceeded the Ethernet standard. This is most frequently due to incompatible duplex settings.

Carrier Errors: Errors are caused by the NIC card losing its link connection to the hub or switch. Check for faulty cabling or faulty interfaces on the NIC and networking equipment.

How to See MAC Addresses

There are times when you lose connectivity with another server that is directly connected to your local network. Taking a look at the ARP table of the server from which you are troubleshooting will help determine whether the remote server's NIC is responding to any type of traffic from your Linux box. Lack of communication at this level may mean:

1. Either server might be disconnected from the network.
2. There might be bad network cabling.
3. A NIC might be disabled or the remote server might be shut down
4. The remote server might be running firewall software such as iptables or the Windows XP built in firewall. Typically in this case, you can see the MAC address, the server is running the correct software, but the desired communication doesn't appear to be occurring to the client on the same network.

Here is a description of the commands you may use to determine ARP values:

- The ifconfig -a command shows you both the NIC's MAC address and the associated IP addresses of the server that you are currently logged in to. Here you can see the wlan0 interface has two IP addresses 192.168.1.100 and 192.168.1.99 tied to the NIC hardware MAC address of 00:06:25:09:6A:B5

```
[root@bigboy tmp]# ifconfig -a

wlan0 Link encap:Ethernet HWaddr 00:06:25:09:6A:B5
inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
RX packets:47379 errors:0 dropped:0 overruns:0 frame:0
TX packets:107900 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:100
RX bytes:4676853 (4.4 Mb) TX bytes:43209032 (41.2 Mb)
Interrupt:11 Memory:c887a000-c887b000

wlan0:0 Link encap:Ethernet HWaddr 00:06:25:09:6A:B5
inet addr:192.168.1.99 Bcast:192.168.1.255 Mask:255.255.255.0
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
Interrupt:11 Memory:c887a000-c887b000

[root@bigboy tmp]#
```

- The arp -a command will show you the MAC addresses in your server's ARP table and all the other servers on the directly connected network. Here we see we have some form of connectivity with the router at address 192.168.1.1

```
 [root@bigboy tmp]# arp -a
bigboypix (192.168.1.1) at 00:09:E8:9C:FD:AB [ether] on wlan0
? (192.168.1.101) at 00:06:25:09:6A:D7 [ether] on wlan0
[root@bigboy tmp]#
```

Note: Make sure the IP addresses listed in the ARP table match those of servers expected to be on your network. If they don't, your server might be plugged into the wrong switch or router port.

You should also check the ARP table of the remote server to see whether it is populated with acceptable values.

Using ping to Test Network Connectivity

Whether or not your troublesome server is connected to your local network it is always a good practice to force a response from it.

One of the most common methods used to test connectivity across multiple networks is the ping command. ping sends ICMP echo packets that request a corresponding ICMP echo-reply response from the device at the target address. Because most servers will respond to a ping query it becomes a very handy tool. A lack of response could be due to:

1. A server with that IP address doesn't exist
2. The server has been configured not to respond to pings
3. A firewall or router along the network path is blocking ICMP traffic
4. You have incorrect routing. Check the routes and subnet masks on both the local and remote servers and all routers in between. A classic symptom of bad routes on a server is the ability to ping servers only on your local network and nowhere else. Use traceroute to ensure you're taking the correct path.
5. Either the source or destination device having an incorrect IP address or subnet mask.

There are a variety of ICMP response codes which can help in further troubleshooting. See Appendix I, "Miscellaneous Linux Topics", for a full listing of them.

The Linux ping command will send continuous pings, once a second, until stopped with a Ctrl-C. Here is an example of a successful ping to the server bigboy at 192.168.1.100

```
[root@smallfry tmp]# ping 192.168.1.101

PING 192.168.1.101 (192.168.1.101) from 192.168.1.100 : 56(84) bytes of data.
64 bytes from 192.168.1.101: icmp_seq=1 ttl=128 time=3.95 ms
64 bytes from 192.168.1.101: icmp_seq=2 ttl=128 time=7.07 ms
64 bytes from 192.168.1.101: icmp_seq=3 ttl=128 time=4.46 ms
64 bytes from 192.168.1.101: icmp_seq=4 ttl=128 time=4.31 ms

--- 192.168.1.101 ping statistics ---
4 packets transmitted, 4 received, 0% loss, time 3026ms
rtt min/avg/max/mdev = 3.950/4.948/7.072/1.242 ms

[root@smallfry tmp]#
```

You may get a "Destination Host Unreachable" message. There message is caused by your router or server knowing that the target IP address is part of a valid network, but is getting no response from the target server. There are a number of reasons for this:

If you are trying to ping a host on a directly connected network:

1. The client or server might be down, or disconnected for the network.
2. Your NIC might not have the correct duplex settings; you may verify this with the mii-tool command.
3. You might have the incorrect type of cable connecting your Linux box to the network. There are two basic types, straight through and crossover.
4. In the case of a wireless network, your SSID or encryption keys might be incorrect.

If you are trying to ping a host on remote network:

The network device doesn't have a route in its routing table to the destination network and sends an ICMP reply type 3 which triggers the message. The resulting message might be Destination Host Unreachable or Destination Network Unreachable.

```
[root@smallfry tmp]# ping 192.168.1.105
PING 192.168.1.105 (192.168.1.105) from 192.168.1.100 : 56(84) bytes of data.
From 192.168.1.100 icmp_seq=1 Destination Host Unreachable
From 192.168.1.100 icmp_seq=2 Destination Host Unreachable
From 192.168.1.100 icmp_seq=3 Destination Host Unreachable
From 192.168.1.100 icmp_seq=4 Destination Host Unreachable
From 192.168.1.100 icmp_seq=5 Destination Host Unreachable
From 192.168.1.100 icmp_seq=6 Destination Host Unreachable
--- 192.168.1.105 ping statistics ---
8 packets transmitted, 0 received, +6 errors, 100% loss, time 7021ms, pipe 3
[root@smallfry tmp]#
```

Using telnet to Test Network Connectivity

An easy way to tell if a remote server is listening on a specific TCP port is to use the telnet command. By default, telnet will try to connect on TCP port 23, but you can specify other TCP ports by typing them in after the target IP address. HTTP uses TCP port 80, HTTPS uses port 443.

Here is an example of testing server 192.168.1.102 on the TCP port 22 reserved for SSH:

```
[root@bigboy tmp]# telnet 192.168.1.102 22
```

When using telnet troubleshooting, here are some useful guidelines to follow that will help to isolate the source of the problem:

- Test connectivity from the remote PC or server.
- Test connectivity on the server itself. Try making the connection to the loopback address as well as the NIC IP address. If the server is running a firewall package such as the Linux iptables software, all loopback connectivity is allowed, but connectivity to desired TCP ports on the NIC interface might be blocked sometimes. Further discussion of the Linux iptables package is covered in a later section.
- Test connectivity from another server on the same network as the target server. This helps to eliminate the influence of any firewalls protecting the entire network from outside.

Linux telnet Troubleshooting

The following sections the use of telnet troubleshooting from a Linux box.

Note: Always remember that many Linux servers have the iptables firewall package installed by default. This is often the cause of many connectivity problems and the firewall rules should be correctly updated. In some cases where the network is already protected by a firewall, iptables might be safely turned off. You can use the /etc/init.d/iptables status command on the target server to determine whether iptables is running.

Successful Connection

With Linux a successful telnet connection is always greeted by a Connected to message like the one seen below when trying to test connectivity to server 192.168.1.102 on the SSH port (TCP 22).

```
[root@bigboy tmp]# telnet 192.168.1.102 22
Trying 192.168.1.102...
Connected to 192.168.1.102.
Escape character is '^]'.
SSH-1.99-OpenSSH_3.4p1
^]
telnet> quit
Connection closed.
[root@ bigboy tmp]#
```

To break out of the connection you have to press the Ctrl and] keys simultaneously, not the usual Ctrl-C.

Note: In many cases you can successfully connect on the remote server on the desired TCP port, yet the application doesn't appear to work. This is usually caused by there being correct network connectivity but a poorly configured application.

Connection Refused Messages

You will get a connection refused message for one of the following reasons:

- The application you are trying to test hasn't been started on the remote server.
- There is a firewall blocking and rejecting the connection attempt

Here is some sample output:

```
[root@bigboy tmp]# telnet 192.168.1.100 22
Trying 192.168.1.100...
telnet: connect to address 192.168.1.100: Connection refused
[root@bigboy tmp]#
```

telnet Timeout or Hanging

The telnet command will abort the attempted connection after waiting a predetermined time for a response. This is called a timeout. In some cases, telnet won't abort, but will just wait indefinitely. This is also known as hanging. These symptoms can be caused by the one of the following reasons:

- The remote server doesn't exist on the destination network. It could be turned off.
- A firewall could be blocking and not rejecting the connection attempt, causing it to timeout instead of being quickly refused.

```
[root@bigboy tmp]# telnet 216.10.100.12 22
Trying 216.10.100.12...
telnet: connect to address 216.10.100.12: Connection timed out
[root@bigboy tmp]#
```

telnet Troubleshooting Using Windows

Sometimes you have to troubleshoot Linux servers from a Windows PC. The telnet commands are the same, but the results are different. Go to the command line and type the same telnet command as you would in Linux.

Screen Goes Blank - Successful Connection

If there is connectivity, your command prompt screen will go blank. Using the Ctrl-C key sequence enables you to exit the telnet attempt.

"Connect Failed" Messages

The Connect failed messages are the equivalent of the Linux Connection refused messages explained above and are caused for the same reasons.

```
C:\>telnet 172.16.1.102 256
Connecting To 172.16.1.102...Could not open connection to the host, on port 256:
Connect failed
C:\>
```

telnet Timeout or Hanging

As explained previously, if there is no connectivity, the session will appear to hang or timeout. This is usually caused by the target server being turned off or by a firewall blocking the connection.

```
C:\>telnet 216.10.100.12 22
Connecting To 216.10.100.12...
```

Testing Web sites with the curl and wget Utilities

Testing a Web site's performance using a Web browser alone is sometimes insufficient to get a good idea of the source of slow Web server performance. Many useful HTTP error codes are often not displayed by browsers making troubleshooting difficult. A much better combination of tools is to use telnet to test your site's TCP port 80 response time in conjunction with data from the Linux curl and wget HTTP utilities.

Rapid TCP response times, but slow curl and wget response times usually point, not to a network issue, but to slowness in the configuration of the Web server or any supporting application or database servers it may use to generate the Web page.

Using curl

The curl utility acts like a text based Web browser in which you can select to see either the header or complete body of a Web page's HTML code displayed on your screen.

A good start is to use the curl command with the -I flag to view just the Web page's header and HTTP status code. By not using the -I command you will see all the Web page's HTML code displayed on the screen. Either method can provide a good idea of your server's performance.

```
[root@ bigboy tmp]# curl -I www.linuxhomenetworking.com
HTTP/1.1 200 OK
Date: Tue, 19 Oct 2004 05:11:22 GMT
Server: Apache/2.0.51 (Fedora)
Accept-Ranges: bytes
Vary: Accept-Encoding,User-Agent
Connection: close
Content-Type: text/html; charset=UTF-8
[root@bigboy tmp]#
```

In this case the Web server appears to be working correctly because it returns a 200 OK code. Please refer to Chapter 20, "The Apache Web Server", for a more complete listing of possibilities.

Using wget

You can use wget to recursively download a Web site's Web pages, including the entire directory structure of the Web site, to a local directory.

By not using recursion, and activating the timestamping feature (the -N switch), you view not only the HTML content of the Web site's index page in your local directory, but also the download speed, file size and precise start and stop times for the download. This can be very helpful in providing a simple way to obtain snapshots of your server's performance.

```
[root@bigboy tmp]# wget -N www.linuxhomenetworking.com
--23:07:22--  http://www.linuxhomenetworking.com/
           => `index.html'
Resolving www.linuxhomenetworking.com... done.
Connecting to www.linuxhomenetworking.com[65.115.71.34]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Last-modified header missing -- time-stamps turned off.
--23:07:22--  http://www.linuxhomenetworking.com/
           => `index.html'
Connecting to www.linuxhomenetworking.com[65.115.71.34]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]

   [   <=>                                     ] 122,150      279.36K/s

23:07:22 (279.36 KB/s) - `index.html' saved [122150]
[root@bigboy tmp]#
```

The netstat Command

Like curl and wget, netstat can be very useful in helping to determine the source of problems. Using netstat, with the -an option lists all the TCP ports on which your Linux server is listening including all the active network connections to and from your server. This can be very helpful in determining whether slowness is due to high traffic volumes:

```
[root@bigboy tmp]# netstat -an
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 :::80                   :::*                    LISTEN
tcp        0   1124 ::ffff:65.115.71.34:80 ::ffff:24.4.97.110:2955  ESTABLISHED
...
...
...
[root@bigboy tmp]#
```

Most TCP connections create permanent connections, HTTP is different because the connections are shut down on their own after a pre defined inactive timeout or time_wait period on the Web server. It is therefore a good idea to focus on these types of short-lived connections too. You can determine the number of established and time_wait TCP connections on your server by using the netstat command filtered by the grep and egrep commands, with the number of matches being counted by the wc command which, in this case shows 14 connections.

```
[root@bigboy tmp]# netstat -an | grep tcp | egrep -i 'established|time_wait' | wc -l
14
[root@bigboy tmp]#
```

The Linux iptables Firewall

An unexpected source of server connectivity issues for brand new servers is frequently the iptables firewall. This is installed by default under Fedora and RedHat and usually allows only a limited range of traffic. You may want to try stopping iptables while you troubleshoot.

Managing iptables is easy to do, but the procedure differs between Linux distributions. Here are some things to keep in mind.

- Firstly, different Linux distributions use different daemon management systems. Each system has its own set of commands to do similar operations. The most commonly used daemon management systems are SysV and Systemd.
- Secondly, the daemon name needs to be known. In the case of iptables the name is iptables.

Armed with this information you can know how to:

- Start your daemons automatically on booting
- Stop, start and restart them later on during troubleshooting or when a configuration file change needs to be applied.

For more details on this, please take a look at the "Managing Daemons" section of Chapter 6 "Installing Linux Software"

How to Configure iptables Rules

Stopping iptables may not be a good permanent solution especially if your network isn't protected by a firewall. You can read more about configuring iptables from Chapter 14, "Linux Firewalls Using iptables".

Using traceroute to Test Connectivity

Another tool for network troubleshooting is the traceroute command. It gives a listing of all the router hops between your server and the target server. This helps you verify that routing over the networks in between is correct.

The traceroute command works by sending a UDP packet destined to the target with a TTL of 0. The first router on the route recognizes that the TTL has already been exceeded and discards or drops the packet, but also sends an ICMP time exceeded message back to the source. The traceroute program records the IP address of the router that sent the message and knows that that is the first hop on the path to the final destination. The traceroute program tries again, with a TTL of 1. The first hop, sees nothing wrong with the packet, decrements the TTL to 0 as expected, and forwards the packet to the second hop on the path. Router 2, sees the TTL of 0, drops the packet and replies with an ICMP time exceeded message. traceroute now knows the IP address of the second router. This continues around and around until the final destination is reached.

> **Note:** In Linux the traceroute command is traceroute. In Windows it is tracert.

> **Note:** You will receive traceroute responses only from functioning devices. If a device responds it is less likely to be the source of your problems.

Sample traceroute Output

Here is a sample output for a query to 144.232.20.158. Notice that all the hop times are under 50 milliseconds (ms) which is acceptable.

```
[root@bigboy tmp]# traceroute -I 144.232.20.158

traceroute to 144.232.20.158 (144.232.20.158), 30 hops max, 38 byte packets
1 adsl-67-120-221-110.dsl.sntc01.pacbell.net (67.120.221.110) 14.408 ms 14.064 ms 13.111 ms
2 dist3-vlan50.sntc01.pbi.net (63.203.35.67) 13.018 ms 12.887 ms 13.146 ms
3 bb1-g1-0.sntc01.pbi.net (63.203.35.17) 12.854 ms 13.035 ms 13.745 ms
4 bb2-p11-0.snfc21.pbi.net (64.161.124.246) 16.260 ms 15.618 ms 15.663 ms
5 bb1-p14-0.snfc21.pbi.net (64.161.124.53) 15.897 ms 15.785 ms 17.164 ms
6 sl-gw11-sj-3-0.sprintlink.net (144.228.44.49) 14.443 ms 16.279 ms 15.189 ms
7 sl-bb25-sj-6-1.sprintlink.net (144.232.3.133) 16.185 ms 15.857 ms 15.423 ms
8 sl-bb23-ana-6-0.sprintlink.net (144.232.20.158) 27.482 ms 26.306 ms 26.487 ms
```

```
[root@bigboy tmp]#
```

Possible traceroute Messages

There are a number of possible message codes traceroute can give, these are listed in Table 4-1.

Table 4-1: traceroute Return Code Symbols

Traceroute Symbol	Description
***	Expected 5 second response time exceeded. Could be caused by: ■ A router on the path not sending back the ICMP "time exceeded" messages ■ A router or firewall in the path blocking the ICMP "time exceeded" messages ■ The target IP address not responding
!H, !N, or !P	Host, network or protocol unreachable
!X or !A	Communication administratively prohibited. A router Access Control List (ACL) or firewall is in the way
!S	Source route failed. Source routing attempts to force traceroute to use a certain path. Failure might be due to a router security setting

traceroute Time Exceeded False Alarms

If there is no response within a 5-second timeout interval an asterisk (*) is printed for that probe as seen in the following example.

```
[root@bigboy tmp]# traceroute 144.232.20.158
traceroute to 144.232.20.158 (144.232.20.158), 30 hops max, 38 byte packets
1 adsl-67-120-221-110.dsl.sntc01.pacbell.net (67.120.221.110) 14.304 ms 14.019 ms 16.120 ms
2 dist3-vlan50.sntc01.pbi.net (63.203.35.67) 12.971 ms 14.000 ms 14.627 ms
3 bb1-g1-0.sntc01.pbi.net (63.203.35.17) 15.521 ms 12.860 ms 13.179 ms
4 bb2-p11-0.snfc21.pbi.net (64.161.124.246) 13.991 ms 15.842 ms 15.728 ms
5 bb1-p14-0.snfc21.pbi.net (64.161.124.53) 16.133 ms 15.510 ms 15.909 ms
6 sl-gw11-sj-3-0.sprintlink.net (144.228.44.49) 16.510 ms 17.469 ms 18.116 ms
7 sl-bb25-sj-6-1.sprintlink.net (144.232.3.133) 16.212 ms 14.274 ms 15.926 ms
8 * * *
9 * * *
[root@bigboy tmp]#
```

Some devices will prevent traceroute packets directed at their interfaces, but will allow ICMP packets. Using traceroute with a -I flag forces traceroute to use ICMP packets that may go through. In this case the * * *, status messages disappear:

```
[root@bigboy tmp]# traceroute -I 144.232.20.158
traceroute to 144.232.20.158 (144.232.20.158), 30 hops max, 38 byte packets
1 adsl-67-120-221-110.dsl.sntc01.pacbell.net (67.120.221.110) 14.408 ms 14.064 ms 13.111 ms
2 dist3-vlan50.sntc01.pbi.net (63.203.35.67) 13.018 ms 12.887 ms 13.146 ms
3 bb1-g1-0.sntc01.pbi.net (63.203.35.17) 12.854 ms 13.035 ms 13.745 ms
4 bb2-p11-0.snfc21.pbi.net (64.161.124.246) 16.260 ms 15.618 ms 15.663 ms
5 bb1-p14-0.snfc21.pbi.net (64.161.124.53) 15.897 ms 15.785 ms 17.164 ms
6 sl-gw11-sj-3-0.sprintlink.net (144.228.44.49) 14.443 ms 16.279 ms 15.189 ms
7 sl-bb25-sj-6-1.sprintlink.net (144.232.3.133) 16.185 ms 15.857 ms 15.423 ms
8 sl-bb23-ana-6-0.sprintlink.net (144.232.20.158) 27.482 ms 26.306 ms 26.487 ms
[root@bigboy tmp]#
```

traceroute Internet Slowness False Alarm

The following traceroute gives the impression that a Web site at 80.40.118.227 might be slow because there is congestion along the way at hops 6 and 7 where the response time is over 200ms:

```
C:\>tracert 80.40.118.227
  1     1 ms     2 ms     1 ms  66.134.200.97
  2    43 ms    15 ms    44 ms  172.31.255.253
  3    15 ms    16 ms     8 ms  192.168.21.65
  4    26 ms    13 ms    16 ms  64.200.150.193
  5    38 ms    12 ms    14 ms  64.200.151.229
  6   239 ms   255 ms   253 ms  64.200.149.14
  7   254 ms   252 ms   252 ms  64.200.150.110
  8    24 ms    20 ms    20 ms  192.174.250.34
  9    91 ms    89 ms    60 ms  192.174.47.6
 10    17 ms    20 ms    20 ms  80.40.96.12
 11    30 ms    16 ms    23 ms  80.40.118.227
Trace complete.
C:\>
```

This indicates only that the devices on hops 6 and 7 were slow to respond with ICMP TTL exceeded messages, but not and indication of congestion, latency, or packet loss. If any of those conditions existed all points past the problematic link would show high latency.

Many Internet routing devices give very low priority to traffic related to traceroute in favor of revenue generating traffic.

traceroute Dies At The Router Just Before The Server

In this case the last device to respond to the traceroute just happens to be the router that acts as the default gateway of the server. The problem is not with the router, but with the server. Remember, you will only receive traceroute responses from functioning devices.

Possible causes of this problem include the following:

- A server has a bad default gateway
- The server is running some type of firewall software that blocks traceroute
- The server is shut down, or disconnected from the network, or it has an incorrectly configured NIC.

```
C:\>tracert 80.40.100.18

Tracing route to 80.40.100.18 over a maximum of 30 hops
 1    33 ms    49 ms    28 ms  192.168.1.1
 2    33 ms    49 ms    28 ms  65.14.65.19
 3    33 ms    32 ms    32 ms  81.25.68.252
 4    47 ms    32 ms    31 ms  80.40.97.1
 5    29 ms    28 ms    32 ms  80.40.96.114
 6     *        *        *     Request timed out.
 7  ^C
C:\>
```

Always Get a Bidirectional traceroute

It is always best to get a traceroute from the source IP to the target IP and also from the target IP to the source IP. This is because the packet's return path from the target is sometimes not the same as the path taken to get there. A high traceroute time equates to the round trip time for both the initial traceroute query to each hop and the response of each hop.

Here is an example of one such case, using disguised IP addresses and provider names. There was once a routing issue between telecommunications carriers FastNet and SlowNet. When a user at IP address 40.16.106.32 did a traceroute to 64.25.175.200, a problem seemed to appear at the 10th. hop with OtherNet. However, when a user at 64.25.175.200 did a traceroute to 40.16.106.32, latency showed up at hop 7 with the return path being very different.

In this case, the real traffic congestion was occurring where FastNet handed off traffic to SlowNet in the second trace. The latency appeared to be caused at hop 10 on the first trace not because that hop was slow, but because that was the first hop at which the return packet traveled back to the source via the congested route. Remember, traceroute gives the packet round trip time.

Trace route to 40.16.106.32 from 64.25.175.200

```
1  0    ms 0    ms 0    [64.25.175.200]
2  0    ms 0    ms 0    [64.25.175.253]
3  0    ms 0    ms 0    border-from-40-tesser.boulder.co.coop.net [207.174.144.169]
4  0    ms 0    ms 0    [64.25.128.126]
5  0    ms 0    ms 0    p3-0.dnvtco1-cr3.othernet.net [4.25.26.53]
6  0    ms 0    ms 0    p2-1.dnvtco1-br1.othernet.net [4.24.11.25]
7  0    ms 0    ms 0    p15-0.dnvtco1-br2.othernet.net [4.24.11.38]
8  30   ms 30   ms 30   p15-0.snjpca1-br2.othernet.net [4.0.6.225]
9  30   ms 30   ms 30   p1-0.snjpca1-cr4.othernet.net [4.24.9.150]
10 1252 ms 1212 ms 1202 h0.webhostinc2.othernet.net [4.24.236.38]
11 1252 ms 1212 ms 1192 [40.16.96.11]
12 1262 ms 1212 ms 1192 [40.16.96.162]
13 1102 ms 1091 ms 1092 [40.16.106.32]
```

Trace route to 64.25.175.200 from 40.16.106.32

```
1  1    ms 1    ms 1    ms [40.16.106.3]
2  1    ms 1    ms 1    ms [40.16.96.161]
3  2    ms 1    ms 1    ms [40.16.96.2]
4  1    ms 1    ms 1    ms [40.16.96.65]
5  2    ms 2    ms 1    ms border8.p4-2.webh02-1.sfj.fastnet.net [216.52.19.77]
6  2    ms 1    ms 1    ms core1.ge0-1-net2.sfj.fastnet.net [216.52.0.65]
7  993  ms 961  ms 999  ms sjo-edge-03.inet.slownet.net [208.46.223.33]
8  1009 ms 1008 ms 971  ms sjo-core-01.inet.slownet.net [205.171.22.29]
9  985  ms 947  ms 983  ms svl-core-03.inet.slownet.net [205.171.5.97]
10 1028 ms 1010 ms 953  ms [205.171.205.30]
11 989  ms 988  ms 985  ms p4-3.paix-bi1.othernet.net [4.2.49.13]
12 1002 ms 1001 ms 973  ms p6-0.snjpca1-br1.othernet.net [4.24.7.61]
13 1031 ms 989  ms 978  ms p9-0.snjpca1-br2.othernet.net [4.24.9.130]
14 1031 ms 1017 ms 1017 ms p3-0.dnvtco1-br2.othernet.net [4.0.6.226]
15 1027 ms 1025 ms 1023 ms p15-0.dnvtco1-br1.othernet.net [4.24.11.37]
16 1045 ms 1037 ms 1050 ms p1-0.dnvtco1-cr3.othernet.net [4.24.11.26]
17 1030 ms 1020 ms 1045 ms p0-0.cointcorp.othernet.net [4.25.26.54]
18 1038 ms 1031 ms 1045 ms gw234.boulder.co.coop.net [64.25.128.99]
19 1050 ms 1094 ms 1034 ms [64.25.175.253]
20 1050 ms 1094 ms 1034 ms [64.25.175.200]
```

ping and traceroute Troubleshooting Example

In this example, a ping to 186.9.17.153 gave a TTL timeout message. Ping TTLs will usually timeout only if there is a routing loop in which the packet bounces between two routers on the way to the target. Each bounce causes the TTL to decrease by a count of 1 until the TTL reaches 0 at which point you get the timeout.

The routing loop was confirmed by the traceroute in which the packet was proven to be bouncing between routers at 186.40.64.94 and 186.40.64.93:

```
G:\>ping 186.9.17.153

Pinging 186.9.17.153 with 32 bytes of data:

Reply from 186.40.64.94: TTL expired in transit.
Reply from 186.40.64.94: TTL expired in transit.
Reply from 186.40.64.94: TTL expired in transit.
Reply from 186.40.64.94: TTL expired in transit.

Ping statistics for 186.9.17.153:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum =  0ms, Average =  0ms

G:\>tracert 186.9.17.153

Tracing route to lostserver.confusion.net [186.9.17.153]

over a maximum of 30 hops:
 1   <10 ms   <10 ms   <10 ms  186.217.33.1
 2    60 ms    70 ms    60 ms  rtr-2.confusion.net [186.40.64.94]
 3    70 ms    71 ms    70 ms  rtr-1.confusion.net [186.40.64.93]
 4    60 ms    70 ms    60 ms  rtr-2.confusion.net [186.40.64.94]
 5    70 ms    70 ms    70 ms  rtr-1.confusion.net [186.40.64.93]
 6    60 ms    70 ms    61 ms  rtr-2.confusion.net [186.40.64.94]
 7    70 ms    70 ms    70 ms  rtr-1.confusion.net [186.40.64.93]
 8    60 ms    70 ms    60 ms  rtr-2.confusion.net [186.40.64.94]
 9    70 ms    70 ms    70 ms  rtr-1.confusion.net [186.40.64.93]
...
```

```
...
...
Trace complete.
G:\>
```

This problem was solved by resetting the routing process on both routers. The problem was initially triggered by an unstable network link that caused frequent routing recalculations. The constant activity eventually corrupted the routing tables of one of the routers.

traceroute Web sites

Many ISPs will provide their subscribers with the facility to do a traceroute from purpose built servers called looking glasses. A simple web search for the phrase Internet looking glass will provide a long list of alternatives. Doing a traceroute form a variety of locations can help identify whether the problem is with the ISP of your Web server or the ISP used at home/work to provide you with Internet access. A more convenient way of doing this is to use a site like traceroute.org which provides a list of looking glasses sorted by country.

Possible Reasons For Failed Traceroutes

A traceroute can fail to reach its intended destination for a number of reasons, including the following:

- traceroute packets are being blocked or rejected by a router in the path. The router immediately after the last visible one is usually the culprit. It's usually good to check the routing table and/or other status of this next hop device.
- The target server doesn't exist on the network. It could be disconnected or turned off. (!H or !N messages might be produced.)
- The network on which you expect the target host to reside doesn't exist in the routing table of one of the routers in the path. (!H or !N messages might be produced.)
- You may have a typographical error in the IP address of the target server
- You may have a routing loop in which packets bounce between two routers and never get to the intended destination.
- The packets don't have a proper return path to your server. The last visible hop being the last hop in which the packets return correctly. The router immediately after the last visible one is the one at which the routing changes. It's usually good to do the following:
  - Log on to the last visible router.
  - Look at the routing table to determine what the next hop is to your intended traceroute target.
  - Log on to this next hop router.
  - Do a traceroute from this router to your intended target server.
  - **If this works:** Routing to the target server is OK. Do a traceroute back to your source server. The traceroute will probably fail at the bad router on the return path.
  - **If it doesn't work:** Test the routing table and/or other status of all the hops between it and your intended target.

Note: If there is nothing blocking your traceroute traffic, then the last visible router of an incomplete trace is either the last good router on the path, or the last router that has a valid return path to the server issuing the traceroute.

Using MTR To Detect Network Congestion

Matt's Traceoute is an application you can use to do a repeated traceroute in real time; it dynamically shows the round-trip time to reach each hop along the traceroute path. The constant updates enable you not only to visually determine which hops are slow, but also to determine when they appear to be slow. It is a good tool to use whenever you suspect there is some intermittent network congestion.

You type in the word mtr followed by the target IP address to get output similar to the following:

```
[root@bigboy tmp]# mtr 192.168.25.26

                   Matt's traceroute  [v0.52]

Bigboy                                          Fri Feb 20 17:19:17 2004
Keys:  D - Display mode    R - Restart statistics    Q - Quit
                                  Packets               Pings
Hostname                        %Loss  Rcv  Snt  Last Best  Avg  Worst
  1. 192.168.1.1                   0%    17   17    32   10   15     32
  2. 192.168.2.254                 0%    17   17    12   11   18     41
  3. 192.168.3.15                  0%    17   17    23   14   18     25
  4. 192.168.18.35                 0%    16   16    24   23   29     42
  5. 192.168.25.26                 0%    16   16    23   21   26     37
^C
[root@bigboy tmp]#
```

One of the nice features of MTR is that it gives you the best, worst and average roundtrip times in milliseconds for the probe packets between each hop along the way to the final destination. The advantage of this is that you can let MTR run for an extended period of time, acting as a constant monitor of communication path quality. The constant refreshing of the screen also enables you to instantaneously spot transient changes in quality fairly easily, making it much more convenient than a regular traceroute.

MTR is automatically installed as part of Fedora Linux. If MTR isn't installed on your system, you can download the RPM software installation package from many of the Fedora download sites. The installation of RPMs is covered in Chapter 6, "Installing Linux Software". There is even a free Windows version called WinMTR.

Viewing Packet Flows with tcpdump

The tcpdump command is one of the most popular packages for viewing the flow of packets through your Linux box's NIC card. It is installed by default on RedHat/Fedora Linux and has very simple syntax, especially if you are doing simpler types of troubleshooting.

One of the most common uses of tcpdump is to determine whether you are getting basic two-way communication. Lack of communication could be due to the following:

- Bad routing
- Faulty cables, interfaces of devices in the packet flow
- The server not listening on the port because the software isn't installed or started
- A network device in the packet path is blocking traffic; common culprits are firewalls, routers with access control lists and even your Linux box running iptables.

Analyzing tcpdump in much greater detail is beyond the scope of this section.

Like most Linux commands, tcpdump uses command-line switches to modify the output. Some of the more useful command-line switches are listed in Table 4-2.

Table 4-2 : Possible TCPdump Switches

tcpdump command switch	Description
`-c`	Stop after viewing *count* packets.
`-i`	Listen on *interface*. If this is not specified, then the command will use the lowest numbered interface that is UP
`-w`	Dump the output to a specially formatted TCPdump dump file
`-C`	Specifies the size the dump file must reach before a new one with a numeric extension is created.
`-t`	Don't print a timestamp at the beginning of each line

You can also add expressions after all the command-line switches. These act as filters to limit the volume of data presented on the screen. You can also use keywords such as and or or between expressions to further fine-tune your selection criteria. Some useful expressions are listed in Table 4-3.

Table 4-3 : Useful tcpdump Expressions

tcpdump command expression	Description
host host-address	View packets from the IP address host-address
icmp	View icmp packets
tcp port port-number	View TCP packets with packets with either a source or destination TCP port of port-number
udp port port-number	View UDP packets with either a source or destination UDP port of port-number

The following is an example of tcpdump being used to view ICMP ping packets going through interface wlan0:

```
[root@bigboy tmp]# tcpdump -i wlan0 icmp
tcpdump: listening on wlan0
21:48:58.927091 smallfry > bigboy.my-site.com: icmp: echo request (DF)
21:48:58.927510 bigboy.my-site.com > smallfry: icmp: echo reply
21:48:58.928257 smallfry > bigboy.my-site.com: icmp: echo request (DF)
21:48:58.928365 bigboy.my-site.com > smallfry: icmp: echo reply
21:48:58.943926 smallfry > bigboy.my-site.com: icmp: echo request (DF)
21:48:58.944034 bigboy.my-site.com > smallfry: icmp: echo reply
21:48:58.962244 bigboy.my-site.com > smallfry: icmp: echo reply
21:48:58.963966 bigboy.my-site.com > smallfry: icmp: echo reply
21:48:58.968556 bigboy.my-site.com > smallfry: icmp: echo reply

9 packets received by filter
0 packets dropped by kernel
[root@bigboy tmp]#
```

In this example:

- The first column of data is a packet timestamp.
- The second column of data shows the packet source and then the destination IP address or server name of the packet.
- The third column shows the packet type.
- Two-way communication is occurring as each echo gets an echo reply.

The following example shows tcpdump being used to view packets on interface wlan0 to/from host 192.168.1.102 on TCP port 22 with no timestamps in the output (-t switch).

```
[root@bigboy tmp]# tcpdump -i wlan0 -t host 192.168.1.102 and tcp port 22

tcpdump: listening on wlan0
smallfry.32938 > bigboy.my-site.com.ssh: S 2013297020:2013297020(0) win 5840 <mss    1460,sackOK,timestamp 75227931 0,nop,wsca
bigboy.my-site.com.ssh > smallfry.32938: R 0:0(0) ack 2013297021 win 0 (DF) [tos 0x10]
smallfry.32938 > bigboy.my-site.com.ssh: S 2013297020:2013297020(0) win 5840 <mss  1460,sackOK,timestamp 75227931 0,nop,wscale
bigboy.my-site.com.ssh > smallfry.32938: R 0:0(0) ack 1 win 0 (DF) [tos 0x10]
smallfry.32938 > bigboy.my-site.com.ssh: S 2013297020:2013297020(0) win 5840 <mss 1460,sackOK,timestamp 75227931 0,nop,wscale
7 packets received by filter
0 packets dropped by kernel
```

```
[root@bigboy tmp]#
```

In this example:

- The first column of data shows the packet source and then the destination IP address or server name of the packet
- The second column shows the TCP flags within the packet
- The client named bigboy is using port 32938 to communicate with the server named smallfry on the TCP SSH port 22.
- Two way communication is occurring

Analyzing tcpdump files

By using the -w filename option you can send the entire Ethernet frame, not just a brief IP information that normally goes to the screen, to a file. This can then be analyzed by graphical analysis tools such as Wireshark, which is available in both Windows and Linux, with customized filters, colorization of packet records based on criteria deemed interesting, and the capability of automatically highlighting certain error conditions such as data retransmissions:

```
tcpdump -i eth1 -w /tmp/packets.dump tcp port 22
```

Covering Wireshark is beyond the scope of this book but that shouldn't discourage you from using it. The application is part of the Fedora RPM suite, and a Windows version is also available.

Common Problems with tcpdump

By default tcpdump will attempt to determine the DNS names of all the IP addresses it sees while logging data. This can slow down tcpdump so much that it appears not to be working at all. The -n switch stops DNS name lookups and will make tcpdump work more reliably.

The following are examples of how the -n switch affects the output:

Without the -n switch

```
[root@bigboy tmp]# tcpdump -i eth1 tcp port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
02:24:34.818398 IP 192-168-1-242.my-site.com.1753 > bigboy-100.my-site.com.ssh: . ack 318574223 win 65471
02:24:34.818478 IP bigboy-100.my-site.com.ssh > 192-168-1-242.my-site.com.1753: P 1:165(164) ack 0 win 6432
02:24:35.019042 IP 192-168-1-242.my-site.com.1753 > bigboy-100.my-site.com.ssh: . ack 165 win 65307
02:24:35.019118 IP bigboy-100.my-site.com.ssh > 192-168-1-242.my-site.com.1753: P 165:401(236) ack 0 win 6432
02:24:35.176299 IP 192-168-1-242.my-site.com.1753 > bigboy-100.my-site.com.ssh: P 0:20(20) ack 401 win 65071
02:24:35.176337 IP bigboy-100.my-site.com.ssh > 192-168-1-242.my-site.com.1753: P 401:629(228) ack 20 win 6432

6 packets captured
7 packets received by filter
0 packets dropped by kernel
[root@bigboy tmp]#
```

With the -n switch

```
[root@bigboy tmp]# tcpdump -i eth1 -n tcp port 22
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 96 bytes
02:25:53.068511 IP 192.168.1.242.1753 > 192.168.1.100.ssh: . ack 318576011 win 65163
02:25:53.068606 IP 192.168.1.100.ssh > 192.168.1.242.1753: P 1:165(164) ack 0 win 6432
02:25:53.269152 IP 192.168.1.242.1753 > 192.168.1.100.ssh: . ack 165 win 64999
02:25:53.269205 IP 192.168.1.100.ssh > 192.168.1.242.1753: P 165:353(188) ack 0 win 6432
02:25:53.408556 IP 192.168.1.242.1753 > 192.168.1.100.ssh: P 0:20(20) ack 353 win 64811
02:25:53.408589 IP 192.168.1.100.ssh > 192.168.1.242.1753: P 353:541(188) ack 20 win 6432

6 packets captured
7 packets received by filter
0 packets dropped by kernel
[root@bigboy tmp]#
```

Viewing Packet Flows with tshark

The tshark program is of the Fedora Linux wireshark RPM. It used to be called tethereal and came as part of the tethereal package, and many texts refer to it by its old name.

The tshark command-line options and screen output mimic that of tcpdump in many ways but tshark has a number of advantages.

The tshark command has the ability of dumping data to a file like tcpdump and creating new files with new filename extensions when a size limit has been reached. It can additionally limit the total number of files created before overwriting the first one in the queue, which is also known as a ring buffer.

The tshark screen output is also more intuitive to read, though the dump file format is identical to tcpdump.

Table 4-4 and Table 4-5 show some popular command switches and expressions that can be used with tshark.

Table 4-4 : Possible tshark Switches

tshark command switch	Description
-c	Stop after viewing *count* packets.

`-i`	Listen on *interface*. If this is not specified, then the command will use the lowest numbered interface that is UP
`-w`	Dump the output to a specially formatted TCPdump dump file
`-C`	Specifies the size the dump file must reach before a new one with a numeric extension is created.
`-b`	The size of the ring buffer when the -C switch is selected.

Table 4-5 : Useful tshark Expressions

tshark command expression	**Description**
host host-address	View packets from the IP address host-address
icmp	View icmp packets
tcp port port-number	View TCP packets with packets with either a source or destination TCP port of port-number
udp port port-number	View UDP packets with either a source or destination UDP port of port-number

In the next example we're trying to observe an HTTP (TCP port 80) packet flow between server smallfry at address 192.168.1.102 and bigboy at IP address 192.168.1.100. The tshark output groups the IP addresses and TCP ports together and then provides the TCP flags, followed by the sequence numbering. It may not be apparent on this page, but the formatting lines up in neat columns on your screen, making analysis much easier. Also notice how the command line mimics that of tcpdump.

```
[root@smallfry tmp]# tshark -i eth0 tcp port 80 and host 192.168.1.100
Capturing on eth0
  0.000000 192.168.1.102 -> 192.168.1.100 TCP 1442 > http [SYN] Seq=3325831828 Ack=0 Win=5840 Len=0
  0.000157 192.168.1.100 -> 192.168.1.102 TCP http > 1442 [SYN, ACK] Seq=3291904936 Ack=3325831829 Win=5792 Len=0
  0.000223 192.168.1.102 -> 192.168.1.100 TCP 1442 > http [ACK] Seq=3325831829 Ack=3291904937 Win=5840 Len=0
  2.602804 192.168.1.102 -> 192.168.1.100 TCP 1442 > http [FIN, ACK] Seq=3325831829 Ack=3291904937 Win=5840 Len=0
  2.603211 192.168.1.100 -> 192.168.1.102 TCP http > 1442 [ACK] Seq=3291904937 Ack=3325831830 Win=46 Len=0
  2.603356 192.168.1.100 -> 192.168.1.102 TCP http > 1442 [FIN, ACK] Seq=3291904937 Ack=3325831830 Win=46 Len=0
  2.603398 192.168.1.102 -> 192.168.1.100 TCP 1442 > http [ACK] Seq=3325831830 Ack=3291904938 Win=5840 Len=0
[root@smallfry tmp]#
```

Covering Wireshark is beyond the scope of this book but that shouldn't discourage you from using it. The application is part of the Fedora RPM suite and a Windows version is also available.

Basic DNS Troubleshooting

Sometimes the source of problems can be due to misconfigured DNS rather than poor network connectivity. As mentioned before, DNS is the system that helps map an IP address to your Web site's domain name and your site may suddenly become unavailable if the mapping is incorrect.

Using nslookup to Test DNS

The nslookup command can be used to get the associated IP address for your domain and vice versa. The nslookup command is very easy to use; you just need to type the command followed by the IP address or Web site name you want to query.

The command actually queries your DNS server for a response, which is then displayed on the screen. Failures can be caused by your server not having the correct value set in the /etc/resolv.conf file as explained in Chapter 18, "Configuring DNS", poor connectivity to your DNS server, or an incorrect configuration on the DNS server.

Using nslookup to Check Your Web site Name

Here we see nslookup returning the IP address 216.151.193.92 for the site www.linuxhomenetworking.com.

```
[root@bigboy tmp]# nslookup www.linuxhomenetworking.com
...
...
Name:   www.linuxhomenetworking.com
Address: 216.151.193.92

[root@bigboy tmp]#
```

Using nslookup To Check Your IP Address

The nslookup command can operate in the opposite way in which a query against the address 216.151.193.92 returns the Web site named www.linuxhomenetworking.com:

```
[root@bigboy tmp]# nslookup 216.151.193.92
...
...
Non-authoritative answer:
92.193.151.216.in-addr.arpa     name = extra193-92.myisp.net.

Authoritative answers can be found from:
193.151.216.in-addr.arpa        nameserver = dns1.myisp.net.
193.151.216.in-addr.arpa        nameserver = dns2.myisp.net.
dns1.myisp.net   internet address = 216.151.192.1

[root@bigboy tmp]#
```

Using nslookup to Query a Specific DNS Server

Sometimes you might want to test the DNS mapping against a specific DNS server, this can be achieved by adding the DNS server's IP address immediately after the IP address of the Web site name you intend to query.

```
[root@bigboy tmp]# nslookup www.linuxhomenetworking.com 68.87.96.3
...
...
Server:         68.87.96.3
Address:        68.87.96.3#53

Name:   www.linuxhomenetworking.com
Address: 216.151.193.92

[root@bigboy tmp]#
```

Note: The nslookup command will probably be removed from future releases of Linux, but can still be used with Windows. The Linux host command can be used as a good replacement.

Using the host Command to Test DNS

More recent versions of Linux have started to use the host command for basic DNS testing. Fortunately syntax is identical to that of nslookup and the resulting output is very similar.

```
[root@bigboy tmp]# host 216.151.193.92
92.193.151.216.in-addr.arpa domain name pointer extra193-92.myisp.net.
[root@bigboy tmp]#

[root@bigboy tmp]# host www.linuxhomenetworking.com
www.linuxhomenetworking.com has address 216.151.193.92
[root@bigboy tmp]#

[root@zippy root]# host www.linuxhomenetworking.com 68.87.96.3
Using domain server:
Name: 68.87.96.3
Address: 68.87.96.3#53
Aliases:

www.linuxhomenetworking.com has address 65.115.71.34
[root@zippy root]#
```

Using nmap

You can use nmap to determine all the TCP/IP ports on which a remote server is listening. It isn't usually an important tool in the home environment, but it can be used in a corporate environment to detect vulnerabilities in your network, such as servers running unauthorized network applications. It is a favorite tool of malicious surfers and therefore should be used to test external as well as internal servers under your control.

Whenever you are in doubt, you can get a list of available nmap options by just entering the command without arguments at the command prompt.

```
[root@bigboy tmp]# nmap

Nmap V. 3.00 Usage: nmap [Scan Type(s)] [Options] <host or net list>
Some Common Scan Types ('*' options require root privileges)
* -sS TCP SYN stealth port scan (default if privileged (root))
  -sT TCP connect() port scan (default for unprivileged users)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
...
...
[root@bigboy tmp]#
```

Some of the more common nmap options are listed in Table 4-6, but you should also refer to the nmap man pages for full descriptions of them all.

Table 4-6 Commonly Used NMAP Options

Argument	Description
`-P0`	Nmap first attempts to ping a host before scanning it. If the server is being protected from ping queries, then you can use this option to force it to scan anyway.
`-T`	Defines the timing between the packets set during a port scan. Some firewalls can detect the arrival of too many non-standard packets within a predetermined time frame. This option can be used to send them from 60 seconds apart with a value of "5" also known as insane mode to 0.3 seconds with a value of "0" in paranoid mode.
`-O`	This will try to detect the operating system of the remote server based on known responses to various types of packets.
`-p`	Lists the TCP/IP port range to scan.
`-s`	Defines a variety of scan methods that use either packets that comply with the TCP/IP standard or are in violation of it.

Here is an example of us trying to do a scan using valid TCP connections (-sT) in the extremely slow "insane" mode (-T 5) from ports 1 to 5000.

```
[root@bigboy tmp]# nmap -sT -T 5 -p 1-5000 192.168.1.153

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on whoknows.my-site-int.com (192.168.1.153):
(The 4981 ports scanned but not shown below are in state: closed)
Port       State       Service
21/tcp     open        ftp
25/tcp     open        smtp
139/tcp    open        netbios-ssn
199/tcp    open        smux
```

```
2105/tcp   open        eklogin
2301/tcp   open        compaqdiag
3300/tcp   open        unknown

Nmap run completed -- 1 IP address (1 host up) scanned in 8 seconds
[root@bigboy tmp]#
```

Full coverage of the possibilities on nmap as a security scanning tool are beyond the scope of this book, but you should go the extra mile and purchase a text specifically on Linux security to help protect you against attempts at malicious security breaches.

Using netcat to Test Network Bandwidth

Most Linux distributions contain the netcat or nc packages which can be used to create a TCP socket over which you can transfer data. The syntax can also vary between distributions so you should refer to your system's man pages if you have any questions.

The netcat server can be easily created with the -l switch that signifies the program should listen, and not talk. The desired TCP port then follows. In this case the server is listening on TCP port 7777.

```
[root@smallfry tmp]# nc -l 7777
```

The netcat client only needs to specify the server's IP address followed by server's the TCP listener port.

```
[root@bigboy ~]# nc 192.168.2.50 7777
```

Any text typed to the console screen of the client;

```
[root@bigboy ~]# nc 192.168.2.50 7777
This is a test of the NetCat program!
[root@bigboy ~]#
```

will also be visible on the server's console.

```
[root@smallfry tmp]# nc -l 7777
This is a test of the NetCat program!
[root@smallfry tmp]#
```

If you want to transfer a file, you only need to use some simple command line redirection. In this case, the server will output all data it receives on port 7777 to a file called FC-6-i386-disc1.iso, and the client pipes the output of the cat command to the netcat client that points to our server.

```
[root@smallfry tmp]# nc -l 7777 > FC-6-i386-disc1.iso

[root@bigboy ~]# cat /tmp/FC-6-i386-disc1.iso | nc 192.168.2.50 7777
```

All Linux systems have a black hole file named /dev/null which automatically discards any data written to it. If you want to test file transfers without filling your disk storage, or having the server's disk I/O be a bottleneck, then use this as your output file instead.

```
[root@smallfry tmp]# nc -l 7777 > /dev/null
```

All Linux systems also have a have a continuous random data source located at /dev/random. Instead of using a file in your tests, you can use this instead for a data stream or infinite duration.

```
[root@bigboy ~]# cat /dev/random | nc 192.168.2.50 7777
```

The netcat program is a good bandwidth tester as it can dominate the capacity of your NIC. Unfortunately, it doesn't provide data transfer statistics, you will have to use some other tool such as MRTG, covered in Chapter 22 " Monitoring Server Performance", to give that information.

Determining the Source of an Attack

Sometimes you realize that your system is under a denial-of-service type attack. This could be either malicious or simply someone rapidly downloading all the pages of your Web site with the Linux wget command. Symptoms include a large numbers of established connections when viewed with the netstat command or an excessive number of entries in your firewall or Web server logs.

Sometimes the attack isn't in the form of a constant bombardment that your server can't handle, but of the type that you can't handle, such as e-mail SPAM. ISPs are usually very sensitive to complaints about SPAM, but though you may have the IP address, a traceroute won't provide any contact information for the ISP.

Sometimes DNS lookups aren't enough to determine who owns an offending IP address. You need another tool.

One of the better ones to use is the whois command. Use it with an IP address or DNS domain as its sole argument and it will provide you with all the administrative information you need to start your hunt. Here is an example for the yahoo.com domain:

```
[root@bigboy tmp]# whois yahoo.com
...
...
   Administrative Contact:
       Domain Administrator
       (NIC-1382062)
       Yahoo! Inc.
       701 First Avenue
       Sunnyvale
       CA
       94089
```

```
        US
        domainadmin@yahoo-inc.com
        +1.4083493300
        Fax- +1.4083493301
...
...
[root@bigboy tmp]#
```

Who Has Used My System?

It is always important to know who has logged into your Linux box. This isn't just to help track the activities of malicious users, but mostly to figure out who made the mistake that crashed the system or blew up Apache with a typographical error in the httpd.conf file.

The last Command

The most common command to determine who has logged into your system is last which lists the last users who logged into the system. Here are some examples:

```
[root@bigboy tmp]# last -100
root     pts/0        reggae.my-site.co Thu Jun 19 09:26   still logged in
root     pts/0        reggae.my-site.co Wed Jun 18 01:07 - 09:26 (1+08:18)
reboot   system boot   2.4.18-14         Wed Jun 18 01:07         (1+08:21)
root     pts/0        reggae.my-site.co Tue Jun 17 21:57 - down    (03:07)
root     pts/0        reggae.my-site.co Mon Jun 16 07:24 - 00:35  (17:10)
wtmp begins Sun Jun 15 16:29:18 2003
[root@bigboy tmp]#
```

In this example someone from reggae.my-site.com logged into bigboy as user root. I generally prefer not to give out the root password and let all the systems administrators log in with their own individual logins. They can then get root privileges by using sudo. This makes it easier to track down individuals rather than groups of users.

The who Command

The who command is used to see who is currently logged in to your computer. Here we see a user logged as root from server reggae.my-site.com.

```
[root@bigboy tmp]# who
root     pts/0        Jun 19 09:26 (reggae.my-site.com)
[root@bigboy tmp]#
```

Conclusion

One of the greatest sources of frustration for any systems administrator is to try to isolate whether poor server performance is due to a network issue or problems with an application or database. The worry can be amplified especially as network instability is often under the control of network engineers who need evidence pointing to problems in their domain of expertise before they will be convinced to act.

These tips should help provide you with a definitive answer by enabling you to isolate the source of most network problems and helping you make their resolution much faster.

The next chapter builds on this new knowledge and expands your troubleshooting skills to include the reading of Linux error log files to assist in the diagnosis of unexpected Linux application behavior.