Table of Contents

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Chapter 9. Basic Tools

This chapter presents a number of tools that do not warrant a full chapter of their own. While these tools are not as complicated as the others listed in this book, they are the the tools most frequently used in network administration. When something goes awry, they will likely be the first tools you use, even if only to rule out certain problems before continuing with a more detailed examination.

The tools listed here include the ping program, which performs basic tests for network reachability to a host; the telnet and Netcat programs, which allow you to test application level protocol problems; traceroute and MTR, which determine network routing paths; and netstat, which prints information about network connections to a workstation.

9.1. Ping

Perhaps the first tool that most administrators reach for when debugging a network problem is the ping program. It can tell you if a machine is alive on the network, and it can print statistics on the network conditions from your machine to another. Though the ping program is relatively simple, it has a few subtleties that are often overlooked.

Ping comes installed with every Unix operating system, so there is no need to build it yourself unless you wish to use a different version than the one you have installed. Be aware that the program tends to use different options and has different default behavior on different systems.

9.1.1. How Ping Works

The ping program operates by sending an Internet Control Message Protocol (ICMP)[1] echo message (an ICMP message whose type is **echo**) to a remote host. When a networked device, such as the remote host, receives the ICMP echo message, it responds with an ICMP **echo reply** to the sending host. The ping program waits to receive this ICMP echo reply message and uses the fact that it has arrived, the amount of time it took to arrive, and other data to report statistics back to the user.

[1] ICMP is a part of the Internet Protocol and is used for sending messages about errors and other control information at the IP layer.

Using a ping test on a device is similar to using sonar on a submarine. Your submarine sends out a loud ping and then waits to hear the response from the sound bouncing off other objects. You want to know if the sound returned at all (otherwise there's nothing out there), and if it does, you want to know how long it took to make the trip.

In the simplest case, the ping program can be used as a test to see if a machine is reachable on the network. One or more ICMP echo messages are sent, and if the device responds within a reasonable amount of time, the ping program will indicate that the machine is alive:

```
Solaris% ping workstation.example.com
workstation.example.com is alive
Solaris% ping client.example.com
no answer from client.example.com
```

Note that one difference in the ping program behavior on different platforms is already relevant. The Solaris version of ping performs this simple alive-or-dead test by default, while the Linux version will send continuous echo requests unless you specifically ask it not to.

If the test is successful, what have you learned? You know that the ICMP packet was sent from your workstation to the remote host and that the remote host was able to send an ICMP packet back to your workstation. If the test fails, however, you do not know exactly where the

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

problem is. It may be that your ICMP echo packets are not reaching the remote machine, or it may be that the remote machine is receiving the packets but the responses are not reaching your workstation. This may be or may not be expected behavior. Some sites administratively block ICMP traffic so that even if a host is on the network, you will not be able to ping it. On rare occasion, you may find a host where the operating system has been modified to ignore ICMP echo messages while other parts of the system will respond normally to network requests. Typically, however, you can expect a host operating under normal conditions to respond to pings, especially if you know it did at some point in the past.

One common problem that can be diagnosed with the ping program is a machine whose netmask or gateway is misconfigured. If either of these pieces of information is incorrect, you may not be able to successfully ping the machine from a different network, but you *can* ping it from a host (or router) that is on the same network.[2] Details on using a router to ping a host are presented later in this section.

[2] The definition of "network" is a little fuzzy here, and "subnet" might be a better word. It is essentially any set of machines that can communicate directly without going through an IP router.

You can diagnose other problems by running ping in a different mode. By repeatedly sending ICMP echo request packets, a continuous ping test can report results as they change in real time. Linux will use this behavior by default; on Solaris you must use the `-s` option:

```
Solaris% ping -s server1.example.com
PING server1.example.com: 56 data bytes
64 bytes from server1.example.com (192.0.2.3): icmp_seq=0. time=14. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=1. time=14. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=2. time=14. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=3. time=13. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=4. time=13. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=5. time=13. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=6. time=13. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=7. time=14. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=8. time=14. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=9. time=13. ms
64 bytes from server1.example.com (192.0.2.3): icmp_seq=10. time=14. ms
^C
----server1.example.com PING Statistics----
11 packets transmitted, 11 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 13/13/14
```

Once every second, the ping program sends an ICMP echo packet. For each ICMP echo reply the program receives, a single line of output is printed. When the user sends a break (by typing <ctrl>-C), the program terminates and reports the cumulative statistics. Included in the statistics is the **packet loss** rate, which is the percentage of ICMP packets sent for which there was never a corresponding response. Though applications will tolerate low levels of packet loss, any amount of packet loss indicates a network problem. A solid local network should have 0% packet loss.

The last field on each line printed in a continuous ping test indicates the time period from when an ICMP echo packet was sent and the corresponding ICMP echo reply was received. This is called the **round-trip-time** (**RTT**). In this case, the average RTT is 13 milliseconds, as you can see from the last line of the output. What is a normal value for the RTT? The answer depends on what you are testing. If the remote host is next door on the same physical network, you would expect a low RTT, say 0–3 ms. If instead the remote host is across many networks and on the other side of the world, you should not be surprised to see 150 ms or larger RTTs.

Because the RTT is a very high-level measurement of latency, a large RTT value can be the result of any number of factors and it does not immediately indicate that something is broken. For example, some transmission media such as satellite links are expected to have a high latency.[3] One common condition that causes high RTT times is when the CPU of the host being pinged is too busy to respond quickly to ICMP requests. Many routers will prioritize other tasks over responding to ICMP when the CPU becomes bogged down with tasks. If this happens, the RTTs to the router will be much higher than normal but not because of any problem with the network itself.

[3] It takes light a little while to make it all the way up to the satellite and back, of course.

You will also want to note whether the RTT values are consistent or erratic. Very large and erratic changes in RTTs can be a sign of congestion, high collision rate, route flapping, or other network problems.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

Options for Ping

Though each ping program is different, most will let you change the default options to facilitate more interesting testing. The most important option is the size of the ICMP packets sent. The default size of the data portion sent by most ping programs is 56 bytes. With 28 bytes added for the IP and ICMP headers, the full IP packet ends up being 84 bytes long, which is still a relatively small packet.[4] Since a number of network problems will not present themselves unless larger packets are used, you will frequently want to instruct the ping program to send more data in a single packet. On Solaris, the packet size is specified as an additional argument after the hostname; on Linux, you must use the `-s` option (not to be confused with the Solaris `-s` option, which requests a continuous ping).

[4] You will notice that the ping program reports 64 bytes received; this is referring to the combined ICMP header and data, but not the IP header. The IP header adds 20 bytes, for a total packet length of 84 bytes.

```
Solaris% ping -s client.example.com 1450
PING client.example.com: 1450 data bytes
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=0. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=1. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=2. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=3. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=4. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=5. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=6. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=7. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=8. time=1. ms
1458 bytes from CLIENT.EXAMPLE.COM (192.0.2.114): icmp_seq=9. time=1. ms
^C
----client.example.com PING Statistics----
10 packets transmitted, 10 packets received, 0% packet loss
round-trip (ms)  min/avg/max = 1/1/1
```

Because Ethernet can support packets as large as 1500 bytes, we choose to send packets with 1450 bytes of data, which leaves a little room for protocol headers. Note that if one of the links between us and client.example.com has a smaller maximum transmission unit (MTU) than 1500 bytes, the packets will need to be fragmented before transmission, which may have unexpected results on your test. If the problem you are debugging is possibly an MTU problem, the results will be relevant, but if it is not an MTU problem, they may confuse the issue. A thorough ping test will test each link at a variety of packet sizes.

Other options that many ping programs will allow include changing the time interval between pings, the IP time-to-live (TTL) value, the IP source address and the IP type of service field. While these are occasionally useful options, they are not often needed to help diagnose network problems.

Pinging from Network Devices

Most managed network devices have ping software built in, which allows you to run a ping test from many different places on your network. The IOS ping that runs on Cisco routers, for example, is illustrated here:

```
router# ping client
Translating "client"...domain server (192.0.2.160) [OK]

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.0.2.114, timeoout is 2 se...
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/maz = 1/1...
```

As with the other ping programs, the Cisco IOS ping will allow you to set more interesting options. If you run the ping command with no arguments, you will be prompted for the more advanced features:

```
router# ping
Protocol [ip]:
Target IP address: 192.0.2.114
```


Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
Repeat count [5]: 50
Datagram size [100]: 1450
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface:
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]: yes
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 50, 1450-byte ICMP Echos to 192.0.2.114, timeoout is 2 ...
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
Success rate is 100 percent (50/50), round-trip min/avg/max = 1...
```

Setting the don't-fragment (DF) bit in the IP header can help debug MTU and fragmentation problems. Validating that the ICMP data field is returned as sent can help detect data corruption on the wire. These and all of the other extended options are available only from enable mode on the router.

Running an Effective Ping Test

Using the ping program to effectively diagnose a problem frequently requires more thought than simply running the program once against a problematic host. A good ping test will explore different possible problems and attempt to eliminate anything unrelated to the true problem. If ping performance is poor to a particular host, is it also as bad to other hosts on the same network? What about hosts on networks between yours and that of the remote host? Does the problem exhibit itself for large packets only? Is the problem transient or consistent? Use these questions as a guide to begin your debugging.

9.2. Telnet

Telnet is another program that comes installed with every Unix operating system. Ordinarily, it is used to login to remote hosts, but it can also be a valuable tool for network administration. Many networking protocols are designed so that you can participate in the protocol using the telnet program. HTTP (for loading Web pages), IMAP (for retrieving email), and SMTP (for sending email) are a few services that fit into this category. For example, the following will retrieve the root Web page from www.example.com:

```
Solaris% telnet www.example.com 80
Trying 192.0.34.166...
Connected to www.example.com (192.0.34.166).
Escape character is '^]'.
GET / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 26 Feb 2003 13:47:13 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html

<HTML>
...
```

The only lines typed by the user are the telnet command and the `GET / HTTP/1.0` line, which must be followed by an extra blank line; that is, press enter a second time. The text after that is the response from the server, most of which has been removed here for the sake of brevity.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

The 80 on the end of the telnet line informs the program to connect to port 80 on the remote host, which is the port that Web servers listen on. The GET command is the HTTP command for retrieving a Web page; we ask it for the Web page named / and specify the version of HTTP we speak as version 1.0. If you replace the word GET with the word HEAD, the Web server will print only the HTTP headers, leaving off the actual HTML content.

In this example, everything worked as it should. But what kind of problems would telnet have alerted you to? First, it would have told you if it was not possible to contact the service:

```
Solaris% telnet www.example.com 80
Trying 192.0.34.166...
telnet: Unable to connect to remote host: Connection refused
```

This means that www.example.com is on the network,[5] but for some reason, it will not respond to requests on port 80. The most likely reason is that the Web server software is not running. However, you do know that the server is alive on the network because the operating system received our packet destined for port 80 and sent a response indicating that it will not accept connections on that port.

[5] As you can test with ping.

Under different circumstances, telnet might hang waiting for a response instead of coming back immediately with the connection refused message:

```
Solaris% telnet www.example.com 80
Trying 192.0.34.166...
```

Your prompt does not return until you break the process with <ctrl>-C. This means either that www.example.com is not on the network at all or that packets on port 80 are not making it to or from the machine, likely because of a filter or a firewall between you and the server.

If telnet makes it as far as the "Connected to ..." line but stalls after that, the problem is likely on the server. It could, for example, be experiencing an abnormally heavy load. However, this could also be an indication of a network problem such as a path MTU mismatch, so further debugging is required before the blame can be placed immediately on the server.

The telnet program can connect to only text-based TCP services and then only if the service is appropriately line oriented. The exact specifications for many of these protocols are listed in RFCs available from http://www.ietf.org/. HTTP version 1.1 is described in RFC 2068, SMTP is described in RFC 2821, and the base IMAP protocol is described in RFC 1730.

9.3. Netcat

Netcat is the next logical step up from telnet. Like telnet, it lets you send data to and from text-based TCP protocols, but it also has the following additional features:

- It can send and receive binary data.
- It can handle UDP protocols.
- It can be set up as a listener for incoming connections.
- Data can be piped to or from a file.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

9.3.1. Installing Netcat

Netcat sometimes comes installed with operating systems, but most often it does not. If you do have it, it will likely be called `nc` and may live at `/usr/bin/nc`. If you do not have it, it can be retrieved from http://www.atstake.com/research/tools/network_utilities/. Be sure to unpackage it in its own separate directory because it will not create one for you. For example:

```
Solaris% mkdir nc
Solaris% mv nc110.tgz nc
Solaris% cd nc
Solaris% gunzip -c nc110.tgz | tar xvf -
```

The build system for Netcat is somewhat nontraditional. There is no configure script, but instead, you supply the name of the operating system to the make command line:

```
Solaris% make solaris
```

or

```
Linux% make linux
```

You can find the names used for other systems by reading the `Makefile` in the distribution. When the build is complete, there will be a file called `nc`, which is the Netcat program.

9.3.2. Using Netcat

Running Netcat with the `-h` option will produce a help listing:

```
Solaris% nc -h
[v1.10]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
options:
        -g gateway    source-routing hop point[s], up to 8
        -G num        source-routing pointer: 4, 8, 12, ...
        -h            this cruft
        -i secs       delay interval for lines sent, ports scanned
        -l            listen mode, for inbound connects
        -n            numeric-only IP addresses, no DNS
        -o file       hex dump of traffic
        -p port       local port number
        -r            randomize local and remote ports
        -s addr       local source address
        -u            UDP mode
        -v            verbose [use twice to be more verbose]
        -w secs       timeout for connects and final net reads
        -z            zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive]
```

If you wish to test a Web server, just as you did with telnet:

```
Solaris% nc www.example.com 80
HEAD / HTTP/1.0

HTTP/1.1 200 OK
Date: Wed, 26 Feb 2003 13:58:14 GMT
Server: Apache/1.3.27 (Unix)  (Red-Hat/Linux)
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
```


Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html
```

Remember to type an extra newline after the request, regardless of whether it is a GET or a HEAD request.

Netcat behaves somewhat differently than telnet. For one thing, it does not print a message indicating that the connection was established unless you use the `-v` option on the command line, indicating that you want verbose output. In a similar manner, you will need to use `-v` to ask Netcat to explicitly inform you if the connection is refused.

Here is an example of how Netcat can be used to pipe data to or from a file:

```
Solaris% (echo "GET / HTTP/1.0" ; echo ) | nc www.example.com \
   80 > /var/tmp/out
```

This is something that could not easily be done with telnet. In this example, Netcat is both reading input from a pipe and redirecting its output to a file. Of course, you do not have to do both; you may choose to pipe input from a file and view the results on the screen, or you may send the results to a file while typing the input to Netcat yourself. Here the entire output is sent to the file `/var/tmp/out`. For the input, we use the trick of placing two echo statements in parenthesis so that the extra newline character will be sent.

Netcat can also store a hexadecimal dump of traffic using the `-o` option:

```
Solaris% nc -o /var/tmp/hexout www.example.com 80
GET / HTTP/1.0
```

You will still be able view the text traffic on the screen, but an additional copy will be stored in `/var/tmp/hexout` that contains both the hexadecimal and text representations of the data. Data sent from the client to the server will be preceded by a right angle bracket; data sent from the server to the client will be preceded by a left angle bracket:

```
Solaris% head /var/tmp/hexout
> 00000000 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 30 0a    # GET / HTTP/1.0.
> 0000000f 0a                                              # .
< 00000000 48 54 54 50 2f 31 2e 31 20 32 30 30 20 4f 4b 0d # HTTP/1.1 200 OK.
< 00000010 0a 44 61 74 65 3a 20 54 75 65 2c 20 32 35 20 46 # .Date: Tue, 25 F
< 00000020 65 62 20 32 30 30 33 20 32 32 3a 31 31 3a 31 31 # eb 2003 22:11:11
< 00000030 20 47 4d 54 0d 0a 53 65 72 76 65 72 3a 20 41 70 #  GMT..Server: Ap
< 00000040 61 63 68 65 2f 31 2e 33 2e 32 37 20 28 55 6e 69 # ache/1.3.27 (Uni
< 00000050 78 29 20 20 28 52 65 64 2d 48 61 74 2f 4c 69 6e # x)  (Red-Hat/Lin
< 00000060 75 78 29 0d 0a 4c 61 73 74 2d 4d 6f 64 69 66 69 # ux)..Last-Modifi
< 00000070 65 64 3a 20 57 65 64 2c 20 30 38 20 4a 61 6e 20 # ed: Wed, 08 Jan
```

Constructing a UDP example is slightly harder because most UDP protocols are not encoded in a simple text format. However, if you place the appropriate binary data for a UDP service into a file, Netcat will happily send it.

The following Perl script will create a valid DNS query in a binary format:

```
#!/usr/bin/perl
print pack("H*", "f5bc"."0100"."0001"."0000");
print pack("H*", "0000"."0000"."0377"."7777");
print pack("H*", "0765"."7861"."6d70"."6c65");
print pack("H*", "0363"."6f6d"."0000"."0100");
print pack("H*", "01");
```

Place the program in a file called `make-packet.pl` and give it execute permission with:

```
Solaris% chmod u+x make-packet.pl
```

You can then use it to place the packet data in a file:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
Solaris% ./make-packet.pl > /var/tmp/packet
```

Now you can pipe that packet data to Netcat and have Netcat send it to a root name server while capturing the response in `/var/tmp/hexout`. If you wish, you can also run tcpdump at the same time to watch for the response.

```
Solaris1% cat /var/tmp/packet | nc -o /var/tmp/hexout -u \
   198.41.0.4 53 > /dev/null
```

You will have to type <ctrl>-C to break the Netcat process. Since it is not making a TCP connection, it has no way of knowing when the conversation is over. Now examine the results in `/var/tmp/hexout`. You will see the packet you constructed sent to the server and the response from the server, which should list the address for www.example.com and the addresses of other top-level name servers as well.

Finally, Netcat can be directed to listen on a port and accept incoming connections. This is useful for all sorts of things, including examination of client behavior. Say you have a Web browser that is not behaving properly and you suspect it may be sending strange options to the Web servers it tries to contact. You can instruct Netcat to listen on a port while the user attempts to load a Web page from your workstation:

```
Solaris# nc -l -p 80
```

After you have started the program, it will sit and wait for a connection. Note that you must run Netcat from a root account if you wish to listen on port 80 or any other port below 1024. If you do not have root access to a machine, you can always use a higher numbered port and point the Web browser at that instead.

Now you, or the user whose Web browser is misbehaving, can attempt to open a Web page using your workstation as the serving host. If the name of your workstation is `workstation1.example.com`, the URL would be `http://workstation1.example.com/`. Once it is accessed, output like the following will appear from Netcat:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.78 [en] (X11; U; SunOS 5.8 sun4u)
Host: client1.example.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ima...
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
```

If necessary, you can continue the HTTP transaction by typing text into Netcat. Otherwise, if you have all the information you need, you can break the session with <ctrl>-C.

9.4. Traceroute

The traceroute program is a very useful tool that prints a list of the routers an IP packet travels through on its way to a particular destination. If there is trouble communicating with a machine and you suspect the problem may be due to misrouted packets or an intermediate network that is off the air, traceroute will help identify the problem.

9.4.1. How Traceroute Works

The specification for IP includes a mechanism for recording the path taken by a packet. Each router can add its address directly to a packet that has the appropriate option set. However, this mechanism is not commonly used for two reasons. One is that the design allows for a only a very small number of routers to store their addresses in the packet. The other is that routers may treat packets differently if they have special

options set. Since the goal is to determine what the router would do with *ordinary* traffic, it may defeat the purpose to have these packets given special treatment.

Instead of using the IP feature for recording route paths, traceroute uses an extremely clever hack[6] to figure things out. It does not rely on any special options at all but instead takes advantage of an unexpectedly useful but required behavior of IP.

[6] Here the word hack does not refer to anything malicious but is instead used in its older and more traditional sense of a clever and unexpected solution to a problem.

Every IP packet contains a field in its header called the TTL field. This is a number that can range from 0 to 255. When a packet is sent out from a machine, it starts with a relatively high TTL, usually 255, and each router the packet passes through along the way to its destination decrements the TTL value by one.[7] If in the course of decrementing the TTL value the router finds the new value will be zero, the packet is discarded and an ICMP error message is sent back to the original sender. The idea is that no packet should be able to live on the network forever. This helps keep a routing loop or other misconfiguration from becoming a catastrophic problem. Eventually, after being forwarded 255 times, a packet will just disappear from the network.

[7] Actually, the router may decrement the value by more than one but must always decrement the value by at least one. In practice, each router will decrement the value by one.

So how does this help determine the route to a particular destination? Say we want to know the path to www.example.com from client.example.com. Instead of sending out the first packet with the usual TTL value of 255, we send it with the TTL set to one. The first router that receives our packet will decrement the TTL value to zero, and as a result, it will send an ICMP error message back to client.example.com indicating the problem. So now we know the IP address of the first router: It is the source address of the ICMP error message! The router gives away its identity when it reports the problem. Next, we send a packet to www.example.com with the TTL value set to two. The packet will make it through the first router, which decrements the TTL to one, but the second router will decrement the TTL to zero and send an ICMP error message back to client.example.com. Now we have the address of the second hop router. We continue in this way, sending out packets with successively higher TTLs until we can reach the final destination host.

Traceroute uses this algorithm to collect information, as you can see from the sample output that follows. Each line represents one router, beginning with the nearest hop and ending with the destination host. Instead of sending one packet for each test, traceroute sends three; the numbers at the end of each line tell you how much time elapsed after the packet was sent and before the ICMP response for the attempt was received.

```
Solaris% traceroute server.example.com
traceroute to SERVER.EXAMPLE.COM (192.0.2.50), 30 hops max, 40...
 1  ROUTER-1.EXAMPLE.COM (192.0.2.1)  0.379 ms  0.273 ms  0.316 ms
 2  ROUTER-2.EXAMPLE.COM (192.0.2.2)  0.335 ms  0.365 ms  0.320 ms
 3  SERVER.EXAMPLE.COM (192.0.2.50)  69.641 ms  38.169 ms  39.9...
```

9.4.2. Installing Traceroute

Most modern versions of Unix come with traceroute installed by default. On Linux and Solaris, it lives in `/usr/sbin/traceroute`, which might be in your path only if you are logged in to a root account. If your system does not have traceroute installed, you can download it from `ftp://ftp.ee.lbl.gov/traceroute.tar.gz`. It will build easily on most systems:

```
Solaris% ./configure
Solaris% make
```

If you have a particularly old system, you may run into problems building traceroute. Read the `INSTALL` file in the distribution for additional help.

Traceroute needs to be run with root privileges. Typically, it is installed with root as the owner and the setuid bit enabled, which allows non-root users to run it with root privileges. If for some reason it is not installed this way on your system, you will either need to run the program from a root account or turn the setuid bit on yourself:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
Solaris# chown root /usr/local/bin/traceroute
Solaris# chmod u+s /usr/local/bin/traceroute
```

Of course, you must execute these commands from a root account. Do note that if someone else is maintaining your system, that person may have disabled the setuid bit on traceroute on purpose. Since bugs in setuid programs can occasionally let an attacker gain root access to the system from a user-level account, some administrators will disable the setuid bit from all non-essential programs.

9.4.3. Using Traceroute

Most of the time, the only argument given to traceroute is the name of the destination to which you wish to learn the path, though occasionally you may wish to use the `-n` flag to turn off DNS lookups for the router names. The traceroute man page lists a number of more fancy options that control the behavior of the program.

Traceroute will sometimes print special characters designating that a particular kind of unexpected response was received. The meaning of these characters is listed in Figure 9.1.

Figure 9.1. Special Traceroute Characters.

Character	Meaning
*	No response received
!H	Host unreachable
!N	Network unreachable
!P	Protocol unreachable
!S	Source route failed
!F	Fragmentation needed
!X	Administratively unreachable
!*number*	Other ICMP unreachable
!	Response TTL < 1

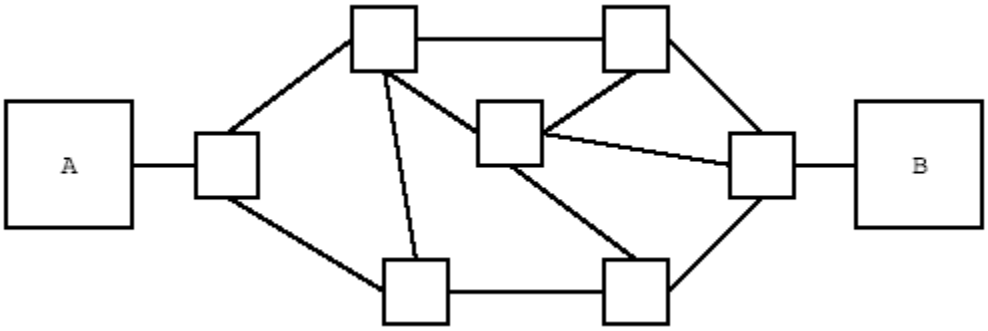
As with any diagnostic tool, it is important to consider what the tool is actually testing because the output is not always a direct representation of reality. Under abnormal conditions, for example, traceroute may present unexpected results. If your site or a site that you are probing is blocking ICMP traffic, traceroute will not work. The routers will send ICMP error messages when the TTL is decremented to zero, but when those messages are blocked from reaching your workstation, traceroute cannot collect the information.

Also remember that on the Internet, every packet sent from machine A to machine B does not have to take the same path. In Figure 9.2, there are many different paths available between the two hosts, and each packet may take a different path, even in the middle of downloading a

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

single Web page. If traceroute finds that more than one router responds to different probes of the same TTL value, it will print the responses from each router. But this does not guarantee that the paths listed by traceroute are the same as those that other traffic took. Perhaps traceroute was unlucky and did not happen to find the same path. Or perhaps an operator administratively changed the path between the time you experienced the problem you are attempting to debug and your attempts to run traceroute. This should not dissuade you from using traceroute as a diagnostic tool; in most cases, traceroute will display the same path other traffic would have taken. But do be aware that it is possible for the path to be different.

Figure 9.2. Different Paths Between Two Hosts.



The traceroute man page discusses a number of other interesting cases in which the output is unexpectedly affected by bugs in software, such as the destination host's operating system. Most of these bugs were corrected long ago, but reading these examples is a good way to understand how seemingly unrelated problems can affect traceroute's behavior.

9.5. MTR

Matt's traceroute (MTR) is a newer version of traceroute that combines the functionality of traceroute and ping in a single interactive session. If your system will support it, it will even use a fancy graphical interface to display its results.

9.5.1. Installing MTR

MTR is available from http://www.bitwizard.nl/mtr/. It will build easily on Linux:

```
Linux% gunzip -c mtr-0.53.tar.gz | tar xvf -
Linux% cd mtr-0.53
Linux% ./configure
Linux% make
```

But Solaris will be a bit more trouble. First, Solaris does not come installed with the ncurses package, which is required by MTR. You can download it from http://www.gnu.org/ and it should build cleanly:

```
Solaris% gunzip -c ncurses-5.3.tar.gz | tar xvf -
Solaris% cd ncurses-5.3
Solaris% ./configure
Solaris% make
```

Then set the `CFLAGS` and `LDFLAGS` environment variables appropriately before configuring MTR. If you use the bash, Korn, or Bourne shell, [8] it will be something like:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

[8] The Bourne shell is /bin/sh.

```
Solaris% CFLAGS=-I/var/tmp/ncurses-5.3/include export CFLAGS
Solaris% LDFLAGS=-L/var/tmp/ncurses-5.3/lib export LDFLAGS
```

But if you use csh or tcsh it will be:

```
Solaris% setenv CFLAGS -I/var/tmp/ncurses-5.3/include
Solaris% setenv LDFLAGS -L/var/tmp/ncurses-5.3/lib
```

In either case, the path should correspond to wherever you built the ncurses package.

Now you may configure and build MTR:

```
Solaris% gunzip -c mtr-0.53.tar.gz | tar xvf -
Solaris% cd mtr-0.53
Solaris% ./configure
Solaris% make
```

But it will likely fail at the last step because of a bug in the way MTR configures for Solaris. You can work around the problem as follows. Type:

```
Solaris% make >! /var/tmp/buildlog
```

Then open the file `/var/tmp/buildlog` in an editor. There will be a few lines, one much longer than the rest:

```
make  all-recursive
Making all in img
cc  -I/var/tmp/ncurses-5.3/include/ -L/var/tmp/ncurses-5.3/lib/...
*** Error code 1
*** Error code 1
*** Error code 1
```

Remove every line except for the one long line. Then, at the very end of that line, leave a space and append the text `-lncurses` so that the end of the line looks something like:

```
...-lm -ltermcap -lncurses
```

Now save the file, and execute:

```
Solaris% source /var/tmp/buildlog
```

When it is complete, there should be an executable file named `mtr` in the directory, indicating that the build was successful.

9.5.2. Using MTR

Just like traceroute, MTR requires root privileges to run correctly. You can either install MTR with setuid root privileges or simply run MTR from a root account.

Here is a capture of a text-based interactive session, which was started with the command `mtr -t www.example.com` on `workstation1.mit.edu`:

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

```
                        Matt's traceroute  [v0.52]
workstation1.mit.edu                                  Tue Feb 25 18:41:58 2003

Keys:  D - Display mode    R - Restart s    Packets               Pings
Hostname                                 %Loss  Rcv  Snt  Last Best  Avg  Worst
 1. ROUTER1.MIT.EDU                        0%    5    5     0    0    0       0
 2. EXTERNAL-RTR-2-BACKBONE.MIT.EDU        0%    5    5     0    0    0       0
 3. p4-0.bstnma1-cr5.bbnplanet.net         0%    4    4     0    0    0       0
 4. so-4-3-0.bstnma1-nbr2.bbnplanet.net    0%    4    4     0    0    0       0
 5. p9-0.nycmny1-nbr2.bbnplanet.net        0%    4    4     6    6    6       6
 6. p15-0.nycmny1-nbr1.bbnplanet.net       0%    4    4     6    6    7       7
 7. p1-0.nycmny1-cr11.bbnplanet.net        0%    4    4     6    6    6       6
 8. pos2-1.pr1.lga1.us.mfnx.net            0%    4    4     6    6    6       6
 9. so-3-0-0.cr2.lga1.us.mfnx.net          0%    4    4     7    7    7       7
10. so-1-0-0.cr2.iad1.us.mfnx.net          0%    4    4    11   11   11      11
11. so-1-0-0.cr2.dca2.us.mfnx.net          0%    4    4    11   11   11      11
12. so-5-3-0.mpr4.sjc2.us.mfnx.net         0%    4    4    73   73   76      81
13. pos8-0.mpr1.sjc2.us.mfnx.net           0%    4    4    73   73   73      74
14. pos0-0.mpr2.lax2.us.mfnx.net           0%    4    4    82   82   82      82
15. pos11-0-0.mpr1.lax1.us.mfnx.net        0%    4    4    83   83   83      83
16. 208.184.95.130.mdr-icann.zoo.icann.o   0%    4    4    83   83   83      83
17. www.example.com                        0%    4    4    83   83   83      83
```

The session continues to send packets and update the display until we break out of it by typing Q or <ctrl>-C. MTR uses essentially the same kind of traceroute algorithm as above to determine the path between you and the destination host but then uses pings (ICMP echo requests) to gather statistics about the reachability of each router. Remember that many routers will treat ICMP traffic differently from other traffic. As mentioned earlier, a busy router may decide that ICMP requests destined for the router itself are a lower priority than traffic passing through the router. For this reason, the statistics presented by MTR sometimes will not represent the same conditions that normal user traffic would encounter.

The MTR man page lists a number of options; those most commonly used are listed in Figure 9.3. As noted at the top of the display, you can also use the R key to reset the statistics and the D key to toggle into two other display modes for viewing packet statistics.

Figure 9.3. Commonly Used MTR Options.

Option	Meaning
-n	Do not perform DNS lookups
-g	Force use of the graphical interface
-t	Force use of the terminal interface
-h	Print help

Aside from the fancy display and the ping statistics, there is an additional advantage to using MTR. Unlike traceroute, MTR will not block when it hits a hop for which it cannot determine the router. That is, when traceroute sends out a packet with the TTL set to five, it will wait for the ICMP error response from the router five hops away before it goes on to send a packet with TTL six. If for some reason the router that is five hops away does not respond, traceroute will never learn about the routers later in the path. MTR, however, will try to access routers a few hops ahead, even if one of the earlier routers does not respond.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003

9.6. Netstat

The netstat program is a bit of a catch-all network reporting tool for Unix workstations. The default behavior is to print information about active network connections on a workstation, which it obtains from the operating system. This can be useful for examining problems on servers and client machines alike.

Netstat comes installed on every Unix operating system, though the options and behavior are very different from system to system. Running netstat with no arguments typically causes it to print a list of every open TCP connection on the machine and a list of active Unix domain sockets:[9]

[9] These are a network connection internal to a machine that programs use to communicate with each other.

```
Solaris% netstat

TCP: IPv4
   Local Address        Remote Address    Swind Send-Q Rwind Recv-Q  State
-------------------- -------------------- ----- ------ ----- ------ -------
workstation.63078    SERVER1.eklogin      24820      0 24820      0 ESTABLISHED
workstation.37318    SERVER2.telnet        2197      0 24820      0 ESTABLISHED
workstation.45954    ROUTER1.ssh           3953      0 25192      0 ESTABLISHED
workstation.telnet   CLIENT.61786          8460      0 25380      0 ESTABLISHED
workstation.47669    MAILSERVER.imap      64240      0 24820      0 TIME_WAIT

Active UNIX domain sockets
Address  Type          Vnode     Conn  Local Addr      Remote Addr
30002d95cb8 stream-ord 300027b6638 00000000 /tmp/.X11-unix/X0
```

In this example, you can see that this machine, workstation.example.com, has several established TCP connections.[10] One is an encrypted login (eklogin) session to server1.example.com, one is a telnet session to server2.example.com, and another is an ssh connection to a router. You can also see that the host client.example.com has a telnet session open to the workstation. Additionally, there is an Internet Message Access Protocol (IMAP) connection from the workstation that is in the TIME_WAIT state, which is a TCP state where the connection is all but terminated, pending a timeout to ensure that no extra data will be transmitted on the port.

[10] The actual netstat output is not usually as well formatted as this; the spacing has been cleaned up a bit for the sake of clarity.

The output on other operating systems may be organized in an entirely different fashion, but the general idea will be the same. This program is obviously a useful means to figuring out exactly what services a machine is trying to contact. On a server, it allows you to figure what clients are connecting and what state the connections are in.

Netstat is also a common way, and on some systems the only way, to view the routing table. Using the `-r` option:

```
Solaris% netstat -r

Routing Table: IPv4
  Destination           Gateway           Flags  Ref   Use   Interface
-------------------- -------------------- ----- ----- ------- ---------
192.0.2.0            workstation          U         1    15116  eri0
BASE-ADDRESS.MCAST.NET workstation        U         1        0  eri0
default              ROUTER1              UG        1   778357
localhost            localhost            UH        3 12485044  lo0
```

Other Netstat functionality varies so much from system to system that your best bet is to read the Netstat man page for details on which features are available. Most versions include an option that will allow you to view traffic statistics for each interface in real time. On Solaris, you can run Netstat as `netstat -i 1` and it will print second-by-second statistics for network traffic. This is an easy way to view the number of packets per second received and transmitted. A handy feature on Linux versions of Netstat is the `-p` flag, which causes it to print the program name and process ID associated with each network connection.

Open Source Network Administration By James Kretchmar
ISBN: 0-13-046210-1 Publisher: Prentice Hall Print Publication Date: 9/22/2003