12.7.3 Using Network Troubleshooting Tools

Click one of the buttons to take you to that part of the video.

Use Network Troubleshooting Tools 0:00-0:15

In this demonstration, we're going to introduce you to several tools that you can use to troubleshoot network problems. First we'll look at ping, then we'll look at netcat, and then we'll end this demonstration by looking at traceroute and tracepath.

'ping' Command 0:16-4:44

The first tool we want to look at is the ping utility. Before we go any further, I want to switch to my root user account on this system. The ping utility is very, very useful.

Let's go ahead and 'ping' our Fedora system. And as you can see, our Fedora system has an IP address of 10.0.0.136. Let's enter its IP address: '10.0.0.136'. And it begins sending ICMP echo requests, and the Fedora system responds with ICMP echo responses.

I'm going to press CTRL + C to stop it. The result of each ping request is displayed on a single line. The first field specifies the size of the echo response packet--the ICMP echo response packet--that it received from the destination system, and as you can see, they were all 64 bytes in size.

Then the next field displays the IP address that the ICMP echo response packet was received from, and in this case it was 10.0.0.136. We also have a time-to-live parameter here, a ttl. This is very important. You can see that the ttl value on each of these packets is set to 64.

The ttl parameters specifies the number of routers that the echo response packet is allowed to cross before it's going to be discarded. Every time that packet crosses a router, this value right here is decremented by 1. In this example, it wasn't decremented at all, because we didn't cross any routers.

The Fedora system that we pinged is on the same network segment as this openSUSE system, hence the ttl value was left alone. However, if we did have to go across a router to get to the Fedora system, the ttl value would have been decremented by 1.

The idea here is that we want to drop unnecessary traffic from the network segment. If you have a packet and it has crossed so many routers that the ttl value reaches 0, then the next router that receives it refuses to forward it any further.

It says, "This packet obviously has no idea where it's going. It's not being delivered, let's just drop it." And then, finally, we have the round-trip time. This tells us how long it took to send the ICMP echo request, and then to receive the ICMP echo response. This parameter can help you identify situations where you might have some latency in the network.

Remember I said earlier that on Linux, the ping command will continue to send ping requests over and over and over until you manually stop it. There is a way to change this behavior, and that is to include the -c option.

We enter '-c', for count, and then a number, such as '4'. This will mimic the way Windows works. This time, the ping command sent just four ICMP echo requests, it received ICMP echo responses, and then it automatically stopped for us.

In this example, I pinged by IP address, and this is actually a best practice when you're troubleshooting network communication problems. You should always ping by IP address first, because this eliminates the dependency that ping would have on a name resolution service if you used a hostname.

If you need to resolve a hostname into an IP address before ping can work, then you need to make sure that that name resolution server, such as your DNS server, is up, working, and reachable. Sometimes that may not be the case.

If it isn't, then the ping may fail and that may lead to you to a false conclusion: that there's a problem in your network infrastructure somewhere between this system and the destination system. When, in fact, you simply might have misconfigured your DNS server address such that the DNS name that you're trying to ping can't be resolved into an IP address. In other words, nothing's wrong; you just put the wrong number when you were configuring DNS resolver address.

What I always do is ping by IP address first to establish that packets can make it to the destination system, and then you can ping by hostname. Notice that the ping command this time resolved the hostname of the destination system into its associated IP address and then used that IP address to ping.

This has a lot of benefits. By doing it this way, I not only know that a basic level of connectivity is there between the system and the destination system, but now I also know that my name resolution system is working correctly as well.

'ping6' Command 4:45-6:18

Notice in the examples we've been working with, the ping command has been using IPv4 packets to test communications. You can also ping using IPv6 packets to test IPv6 communications.

If I run the 'ifconfig' command here, you'll notice that we have a unique local IPv6 address assigned on this system of fc00::1. And if I go to our Fedora system, it has an equivalent unique local IPv6 address of fc00::2. If I want to ping the Fedora system this time using IPv6 packets instead of IPv4, I type 'ping6' instead of ping, and then the IPv6 address of the host that I want to ping: 'fc00::2'.

Remember when you're dealing with IPv6 addresses, if you see :: like this, it basically means this is really actually a much longer address but it was all full of zeros. Everything from here to here was a 0, and so we just use 2 colons to specify that we're shortening the address by eliminating all the redundant zeros, and it makes life a lot easier when you're dealing with very long IPv6 addresses.

I'm going to go ahead and press Enter, and notice that just as with the regular ping command, the ping6 command is able to contact the remote host--just using the IPv6 protocol this time instead of IPv4. And you'll notice that the output of ping6 is pretty much that same as that for ping. We have the packet size, the responding host, ttl, and the round-trip time.

'netcat' Command 6:19-9:17

At this point, we need to shift gears a little bit and now look at the netcat command. netcat, or nc, is a very useful tool for testing network communications between hosts, and this is because it goes one step beyond what ping is able to do.

What netcat does is actually establish a TCP or a UDP connection between the two network hosts. Instead of just sending an ICMP echo request and getting ICMP echo response, we actually establish a network communication session between the two hosts, using TCP or UDP.

Let's take a look at how this works. On this system, on my openSUSE system, I'm going to open up a TCP socket and tell it to start listening for a request. To do this, I enter 'nc' and then I use the '-l' parameter to specify that it listened for incoming connections, and then I need to specify which port I want to listen on.

Let's do '2388'. Because I'm not specifying explicitly a particular protocol, it's going to use TCP by default. If I wanted to use UDP instead of TCP, then I would include the -u option as well. But we're not going to do that today; we're just going to use TCP.

At this point, I have opened a TCP socket on port 2388 and it's listening. It's waiting for somebody to connect to it. Let's go ahead and use nc on our Fedora system to actually establish a connection on port 2388.

I use the 'nc' command again, and now I need to know what the IP address is of our openSUSE system. I forgot to check that before we went over there, so I'm going to open up a new terminal session. Run the 'ifconfig' command, and see that our IP address is 10.0.0.228.

Go back to our Fedora system and we specify the IP address of the openSUSE system, which is '10.0.0.228', and then we have to specify the port that we want to connect to, '2388'. Notice that I did not use the -l option this time because I'm not opening up a listening connection. I'm establishing a connection to a socket that's already in a listening state, waiting for someone to connect to it.

All right. We have established a TCP connection, and we can verify that this is the case by actually entering text here, and seeing whether or not it's displayed on the openSUSE system. Let's just type 'Hello world'. Let's switch back over to the openSUSE system, and we see that Hello world is displayed here.

Be aware that in order for this to work, you do have to go into the firewall in both systems and open up whatever port it is that you want to connect through. In this case, I had to go in and open up port 2388 TCP on both systems, otherwise it won't work. We can break out by pressing CTRL + C, and we'll break out on this side too. Closing the socket.

'traceroute' Command 9:18-12:11

The last thing we're going to do in this demonstration is look at the traceroute and the tracepath utilities. The purpose of traceroute is to identify the network route that your packets have to take to get from one system to another.

It's an excellent troubleshooting tool, because it can help you identify any routers that are causing problems along the way. The syntax for using traceroute is to just type traceroute, followed by the IP address or DNS name of the host that you want to trace the route to.

For our purposes here, I actually can't run it from this system. This is because my openSUSE and my Fedora systems are behind a very, very stringent firewall, and it actually filters out all of my traceroute traffic and doesn't work.

What I'm going to have to do here is actually open an SSH session with another Linux system that is in my DMZ, my demilitarized zone, on the other side of my firewall so that we can actually run the command. I'm going to connect to it right now. I'm going to connect my 'rtracy'

user on that system.

And its hostname is 'linux-gipq'. I entered my rtracy user's password, and I am now connected. Now I'm connected to the system that's in the DMZ, so I can actually run the 'traceroute' command. And let's trace the route to www.google.com.

Oops, I have to switch to my root user account, sorry about that. Let's trace the route to 'www.google.com'. Every single router in the path from that system that's sitting in my DMZ through the internet to www.google.com is listed here.

You will see a couple of routes where we have asterisks. And a lot of people freak out when they see that. It's actually not that big a deal. That doesn't mean that this router didn't respond or isn't functioning properly. All it means is that traceroute was unable to resolve its IP address into a hostname.

As you can see with some of these routers, the traceroute command was able to resolve their IP address into a DNS name. These stars just mean that it wasn't able to do that. The router's still there, it's still doing its job. The packets were able to make it from the source here to the destination here.

You will notice that the traceroute command used the IPv4 protocol to trace the route from this host to google.com. You can do the same thing with the IPv6 protocol. To do this, you simply type, 'traceroute6' instead of traceroute, then specify the hostname that you're trying to connect to.

We can't actually do that here because my ISP does not support IPv6 yet, and hence I can't trace the route outside my local network.

'tracepath' Command 12:12-12:46

The last command we're going to look at in this demonstration is almost identical to traceroute. It does basically the same thing. It's called tracepath. The syntax you use is exactly the same: 'tracepath' instead of traceroute, and we'll put the destination host 'www.google.com'. Press Enter.

And just like traceroute, it traces the path to the google.com web server. And just as you could use traceroute with either IPv4 or IPv6, you can also use tracepath with IPv4 or IPv6. If you want to use IPv6, then you use the tracepath6 command instead of tracepath.

Summary 12:47-12:59

That's it for this demonstration. In this demo, we looked at several useful network troubleshooting tools that you can use to troubleshoot network problems. We first looked at the ping command, then we looked at ping6. We looked at the netcat command, and we ended this demonstration by looking at traceroute and tracepath.
