12.1.1 IP Protocols

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

IP Protocols 0:00-0:24

In this lesson, we're going to review how the IP protocol can be used to enable communications between network systems.

Understand that before your Linux system can participate on a computer network, you first have to install some kind of network interface that connects it to the network. Then you have to configure that network interface to participate on your particular network.

Protocols 0:25-1:20

Let's begin by discussing what a protocol is. You might be asking, "Protocol, what is that?" Strictly speaking, a protocol is a set of rules. In the context of networking, a protocol is a set of rules that govern communications between two or more computer systems.

Really, a good analogy for how a protocol works is to look at human language. Think about it. If you have two people, and those two people need to communicate. They have to speak the same language. Otherwise, no information can be transferred between them. They can send information to each other, but if they don't understand each other's language, no communications occur.

The same thing holds true with computer systems. Before they can share information, they have to be configured to use the same protocol. This protocol specifies how the information is encoded and sent on the network so that the system that receives that information can interpret it and reconstruct the data that was originally sent.

IP Protocol 1:21-3:07

In today's networking environment, you have to be familiar with the IP protocol. That's because the IP protocol is the networking protocol used on the internet, and it's also the protocol that we use with most computer networks in most organizations.

The key thing that you need to understand is that IP is responsible for taking information on the sending system, formatting it in a way that can be transmitted on the network medium, and such that information can then be received by the receiving system, which will then reconstruct the data so that it can see exactly what it was that was sent from the sending system.

Let's suppose we have system A here and system B here, and they're connected over an IP network. Let's suppose system A has this piece of information that it wants to send to B. This piece of information is fairly large. Because it's fairly large, it can't all be sent at once over the network medium to system B.

The IP protocol is going to have to work in conjunction with other protocols in order to take this information and divide it up into digestible chunks. Let's say we have chunk 1, chunk 2, chunk 3, chunk 4, and chunk 5.

Understand that the IP protocol itself is not concerned with how we're going to tear this piece of information up into chunks. All the IP protocol is concerned with is where is this data coming from, system A, and where is it going to, system B.

In order to do this, the IP protocol assigns each of these hosts an address called an IP address. In this case, system A has an IP address of 10.0.0.1 and system B has an IP address of 10.0.0.2.

TCP 3:08-9:15

In order for this to work, the IP protocol is going to work in conjunction with another protocol called TCP, the transmission control protocol. It's TCP's job to break up this big piece of information into little chunks.

As the TCP protocol takes this big chunk of data and breaks it up into little chunks, the IP protocol is going to add two pieces of very important information to each one of those chunks.

It's going to specify the source address of that chunk where it's coming from, in this case 10.0.0.1, and it's also going to specify where that chunk is supposed to go to. In this case, it's supposed to go over the network to the system with an address of 10.0.0.2. Each and every chunk will be assigned these addresses. This will be done by the IP protocol.

The TCP protocol will work hand in hand with the IP protocol to break that data up into chunks. In order to make sure that these chunks get reassembled in the correct order, the TCP protocol is going to assign some sequencing information to it.

Once that's done, the information can be sent over the network from the source system to the destination system.

Because the data is broken up into these little chunks, it is possible for that data to arrive at the destination system out of order. Notice down here that chunk 1--called a packet--arrives first, but then packet 3 arrives before packet 2. Then packets 4 and 5 arrive at the end.

In order for the destination system B to properly reconstruct the data, it takes a look at the sequencing information provided by TCP and reconstructs the data in the correct order, even though it may have been received out of order.

It makes sure that packet 1 goes first, then packet 2, packet 3, packet 4, and then packet 5. By doing this, the exact same data appears over here on system B as was sent from system A.

An important thing to understand here is the relationship between the IP protocol and the TCP protocol. The IP protocol is what we call a connectionless protocol. The IP protocol is not concerned with this sequencing process or breaking the data up into chunks. All it's concerned with is how to get the data from host A over to host B.

The job of breaking this data down into chunks is the job of the TCP protocol. As we talked about, each of these packets is individually addressed. It's an individual piece of data as far as the IP protocol is concerned. The fragmentation of that data on the sending system and the resequencing of information on the receiving system is the job of the TCP protocol.

TCP plays a very important role in this process. Its job is to make sure that the data is reliably exchanged between host A and host B. One of the key things that TCP does to make this happen is require acknowledgements for every single fragment that is sent from host A to host B for each and every packet.

Each time host B receives one of these packets, it sends an acknowledgement back to host A and says, "Hey, I got number 1. Hey, I got number 2. Hey, I got number 3. Hey, I got number 4 too. Hey, I got number 5."

By doing this, A knows that B got all of the pieces of information. And therefore, the data on B should be identical to the data that was sent from A. It's a perfect copy.

It is theoretically possible for something to happen during this process. Let's say that the data transmission started and packets 1, 2, 3, and 4 were received properly from A to B, but 5 never made it. Who knows what happened. Maybe there was an electrical storm and 5 got toasted while it was being transmitted on the network wire. We don't know. Something happened. 5 didn't make it. A knows that it sent five. B knows that it received four.

The TCP protocol over on A is going to sit and wait for that acknowledgement of number 5. It's waiting and it's waiting, and it never comes. In this situation, A knows that it needs to resend 5 a second time. It finally arrives and B acknowledges that it got it. Then the TCP protocol on A is confident that the data that was received by B is in fact an exact copy of that which was sent original from A.

Using TCP with the IP protocol is kind of like using signature confirmation when you're shipping a package with a shipping company. When you send the package, the shipper requires the receiver to sign for the package. This allows the sender to verify that the package actually made it.

TCP works in much the same manner. Every single packet has to be acknowledged by the receiver to the sender. Because of this, the TCP protocol is used by many upper layer applications that require a high degree of data integrity, such as web servers, email servers, FTP servers, and so on.

Think about how mad you would be if someone sent you an email and a big chunk of it was missing because the data had never made it properly. TCP is designed to make sure that does not happen. Anything that's missing will be retransmitted.

Understand that this data fidelity that comes by using TCP comes so at a cost, primarily in the form of latency. This is because TCP requires a fair amount of network and processing overhead by sending all of those acknowledgements back and forth from the receiving system to the sending system.

There are actually some network applications that don't really require this high degree of data fidelity that's used by TCP. There are some upper layer applications that require less latency and can actually tolerate a little more unreliability. These applications use a different protocol with IP instead of TCP.

UDP 9:16-11:40

Instead of TCP they use the user datagram protocol, or UDP. UDP does basically the same job as TCP. It works in conjunction with the IP protocol. It also fragments the piece of data being sent into digestible chunks and it provides sequencing numbers with those chunks just like TCP does. Nothing's different.

Each chunk has a source address and each chunk has a destination address. Nothing has changed there. When we transmit the data over the network medium, it arrives on the receiving system, and then the receiving system uses the same protocol, UDP, to re-sequence the data in the right order.

In this case, 1, 2, 3, 4, 5 packets were sent from the sending system and they're re-sequenced as 1, 2, 3, 4, 5 on the receiving system--this time using the UDP protocol instead of the TCP protocol.

Here's what's different. With TCP, every single packet had to be acknowledged back to the sending system. UDP doesn't care. It does not acknowledge the receipt of the packets. This has some advantages and some disadvantages.

The key advantage is it makes UDP much faster than TCP. The disadvantage is you could lose a packet along the way. Say we lost packet number 5 in the process. UDP doesn't check to see if everything made it properly.

It's kind of like sending someone a postcard through the mail. A postcard doesn't contain a whole lot of data and the receiver doesn't have to sign for it. Essentially, the sender assumes that the mail carrier is reasonably reliable and that the information on the postcard isn't important enough to require the receiver to sign for it.

UDP works in much the same way.

Some upper layer applications that like UDP are things such as streaming audio, streaming video, and Voice over IP. If you, say, are watching a video stream or maybe listening to an audio stream, most likely you're using the UDP protocol instead of TCP.

If we lose one or two or three packets in the transmission, you might hear or see just a tiny little glitch in the playback on the receiving system, but it's not a big deal. You just keep on going and no one cares.

That's where UDP shines, because it provides a lot more throughput than TCP does, which is really good if you're trying to watch a streaming video or listening to streaming audio.

ICMP 11:41-14:16

In addition to IP, TCP, and UDP, you also need to be familiar with the Internet Control Message Protocol, which we affectionately just call ICMP. ICMP is another core protocol in the IP protocol suite, but it differs in purpose from TCP and UDP. TCP and UDP are transport protocols, whereas the primary role of ICMP is to test and verify network communications between hosts.

For example, let's suppose we have two hosts here. Host A over here has an IP address of 10.0.0.1. Host B over here has an IP address of 10.0.0.2. In order to test network connectivity, we can use a cool little utility called ping at the command prompt of a Linux system to spend special IP packets from host A to host B.

These are called ICMP echo request packets. In this case, we're testing communications between host A, 10.0.0.1, and host B, with an address of 10.0.0.2. Host A sends an IP echo request packet addressed to 10.0.0.2. When 10.0.0.2 gets that ICMP echo request packet, it sends in return an ICMP echo response. It sends that ICMP echo response back to the system that sent the ICMP echo request in the first place.

When this process completes, a lot of key pieces of information become available. First of all, if that echo response packet makes it back to the original sending system, then that sender knows that a viable network connection exists between the sender and the receiver.

It also knows that the receiving system is up and running and responding to network requests. In addition, we can also measure how much time it took between the time we sent the echo request and the time that we received the echo response. By doing that, we can measure the latency involved in the communication session.

There could be latency due to an overloaded destination system. There could be latency due to a bad network connection somewhere. Using the ICMP protocol, we can start to diagnose all of these various issues.

If, on the other hand, we send an ICMP echo request, but we never receive an ICMP echo response from the destination system, then we know something is wrong and we can begin troubleshooting the communication process.

Maybe we have a broken network cable somewhere. Maybe the destination system is down, or maybe the destination system has a firewall running that's ignoring all ICMP echo requests.

Summary 14:17-14:31

That's it for this lesson. In this lesson, we briefly reviewed the IP protocol. We discussed how the IP protocol works in conjunction with the TCP and/or the UDP protocol to deliver data over a network connection. Then we ended this lesson by talking about the role and function of the ICMP protocol.
