Google IT Support Course 2
The Bits and Bytes of Computer Networking

Week 1

Introduction to Computer Networking

This defined set of standards that computers must
follow in order to communicate properly is called a protocol.
Computer networking is the name we've given to
the full scope of how computers communicate with each other.

The TCP/IP Five Layer Network Model

Let's start at the bottom of our stack,
where we have what's known as the physical layer.
The physical layer is a lot like what it sounds.
It represents the physical devices that interconnect computers.
This includes the specifications for
the networking cables and the connectors that join devices
together along with specifications
describing how signals are sent over these connections.
The second layer in our model is known as the data link layer.
Some sources will call this layer the network interface or the network access layer.
At this layer, we introduce our first protocols.
While the physical layer is all about cabling,
connectors and sending signals,
the data link layer is responsible for
defining a common way of interpreting these signals,
so network devices can communicate.
Lots of protocols exist at the data link layer,
but the most common is known as Ethernet,
although wireless technologies are becoming more and more popular.
Beyond specifying physical layer attributes,
the Ethernet standards also define a protocol responsible
for getting data to nodes on the same network or link.
The third layer, the network layer is also sometimes called the Internet layer.
It's this layer that allows different networks to
communicate with each other through devices known as routers.
A collection of networks connected together through routers is an internetwork,
the most famous of these being the Internet.
Hopefully you've heard of it.
While the data link layer is responsible for getting data across a single link,
the network layer is responsible for getting data
delivered across a collection of networks.
Think of when a device on your home network connects with a server on the Internet.
It's the network layer that helps gets the data between these two locations.

The most common protocol used at this layer is known as IP or Internet Protocol.
IP is the heart of the Internet and most small networks around the world.
Network software is usually divided into client and server categories,
with the client application initiating a request for
data and the server software answering the request across the network.
A single node may be running multiple client or server applications.
So, you might run an email program and a web browser, both client applications,
on your PC at the same time,
and your email and web server might both run on the same server.
Even so, emails end up in
your email application and web pages end up in your web browser.
That's because our next layer, the transport layer.
While the network layer delivers data between two individual nodes,
the transport layer sorts out
which client and server programs are supposed to get that data.
When you heard about our network layer protocol IP,
you may have thought of TCP IP,
which is a pretty common phrase.
That's because the protocol most commonly used in the fourth layer,
the transport layer, is known as TCP or Transmission Control Protocol.
While often said together as the phrase TCP IP,
to fully understand and troubleshoot networking issues,
it's important to know that they're
entirely different protocols serving different purposes.
Other transfer protocols also use IP to get around,
including a protocol known as UDP or User Datagram Protocol.
The big difference between the two is that TCP provides
mechanisms to ensure that data is reliably delivered while UDP does not.
Last but not least,
the fifth layer is known as the application layer.
There are lots of different protocols at this layer,
and as you might have guessed from the name, they are application-specific.
Protocols used to allow you to browse the web or send receive email are some common ones.
The protocols at play in the application layer will be most familiar to you,
since they are ones you probably interacted with
directly before even if you didn't realize it.
You can think of layers like different aspects of a package being delivered. (the content of the internet)

#	Layer Name	Protocol	Protocol Data Unit	Addressing
5	Application	HTTP, SMTP, etc..	Messages	n/a
4	Transport	TCP/UDP	Segment	Port #'s
3	Network	IP	Datagram	IP address
2	Data Link	Ethernet, Wi-Fi	Frames	MAC Address
1	Physical	10 Base T, 802.11	Bits	n/a

The Basics of Networking Devices

Cables are what connect different devices to each
other allowing data to be transmitted over them.
Most network cables used today can be split into two categories, copper and fiber.
Copper cables are the most common form of networking cable.
They're made up of multiple pairs of copper wires inside plastic insulator.
You may already know that computers communicate in
binary which people represent with ones and zeros.
The sending device communicates binary data across
these copper wires by changing the voltage between two ranges.
The system at the receiving end is able to interpret these voltage changes as
binary ones and zeros which can then be translated into different forms of data.
The most common forms of copper twisted pair cables used in networking are Cat5,
Cat5e, and Cat6 cables.
These are all shorthand ways of saying category five or category six cables.
These categories have different physical characteristics like the number of twists in
the pair of copper wires that result in different usable lengths and transfer rates.
Cat5 is older and has been mostly replaced by Cat5e and Cat6 cables.
From the outside, they all look about the same,
and even internally, they're very similar to the naked eye.
The important thing to know is that differences in how
the twisted pairs are arranged inside these cables can
drastically alter how quickly data can be sent across
them and how resistant these signals are to outside interference.
Cat5e cables have mostly replaced
those older Cat5 cables because their internals reduce crosstalk.
Crosstalk is when an electrical pulse on
one wire is accidentally detected on another wire.

So, the receiving end isn't able to understand the data causing a network error.
Higher level protocols have methods for detecting
missing data and asking for the data a second time.
But of course, this takes up more time.
The higher quality specifications of
a Cat5e cable make it less likely that data needs to be re-transmitted.
That means, on average,
you can expect more data to be transferred in the same amount of time.
Cat6 cables follow an even more strict specification to avoid crosstalk,
making those cables more expensive.
Cat6 cables can transfer data faster and more reliably than Cat5e cables can,
but because of their internal arrangement,
they have a shorter maximum distance when used at higher speeds.
The second primary form of networking cable is known as Fiber,
short for fiber optic cables.
Fiber cables contain individual optical fibers which
are tiny tubes made out of glass about the width of a human hair.
These tubes of glass can transport beams of light.
Unlike copper, which uses electrical voltages,
fiber cables use pulses of light to represent the ones and zeros of the underlying data.
Fiber is even sometimes used specifically in environments where there's a lot of
electromagnetic interference from outside sources
because this can impact data being sent across copper wires.
Fiber cables can generally transport data quicker than copper cables can,
but they're much more expensive and fragile.
Fiber can also transport data
over much longer distances than copper can without suffering potential data loss.

Luckily, there are network devices that allow for
many computers to communicate with each other.
The most simple of these devices is a hub.
A hub is a physical layer device that allows for
connections from many computers at once.
All the devices connected to a hub will end up talking to
all other devices at the same time.
It's up to each system connected to the hub to determine
if the incoming data was meant for them, or to ignore it if it isn't.
This causes a lot of noise on the network and
creates what's called a collision domain.
A collision domain is a network segment
where only one device can communicate at a time.
If multiple systems try sending data at the same time,
the electrical pulses sent across the cable can interfere with each other.
This causes these systems to have to wait for
a quiet period before they try sending their data again.
It really slows down network communications, and
is the primary reason hubs are fairly rare.
They're mostly a historical artifact today.
A much more common way of connecting many computers is with a more sophisticated
device, known as a network switch, originally known as a switching hub.
A switch is very similar to a hub, since you can connect many devices to it so
they can communicate.
The difference is that while a hub is a layer 1 or

physical layer device, a switch is a level 2 or data link device.
This means that a switch can actually inspect the contents of the Ethernet protocol data being sent around the network, determine which system the data is intended for and then only send that data to that one system.
This reduces or even completely eliminates the size of collision domains on a network.

Hubs and switches are the primary devices used to connect computers on a single network, usually referred to as a LAN, or local area network.
But we often want to send or receive data to computers on other networks.
This is where routers come into play.
A router is a device that knows how to forward data between independent networks.
While a hub is a layer one device and a switch is a layer two device.
A router operates at layer three, a network layer.
Just like a switch can inspect Ethernet data to determine where to send things, a router can inspect IP data to determine where to send things.
Routers store internal tables containing information about how to route traffic between lots of different networks all over the world.
The most common type of router you'll see is one for a home network, or a small office.
These devices generally don't have very detailed routing tables.
The purpose of these routers is mainly just to take traffic originating from inside the home, or office LAN, and to forward it along to the ISP, or Internet Service Provider.
Once traffic is at the ISP, a way more sophisticated type of router takes over.
These core routers form the backbone of the Internet and are directly responsible for how we send and receive data all over the Internet every single day.
Core ISP routers don't just handle a lot more traffic than a home or a small office router.
They also have to deal with much more complexity in making decision about where to send traffic.
A core router usually has many different connections to many other routers.
Routers share data with each other via a protocol known as BGP, or Border Gateway Protocol.
That lets them learn about the most optimal paths to forward traffic.
When you open a web browser and load a webpage, the traffic between computers and the web servers could have traveled over dozens of different routers.
The Internet is incredibly large and complicated.
And routers are global guides for getting traffic to the right places.

All of the network devices you've just learned about, exist so that computers can communicate with each other, whether they're in the same room or thousands of miles apart.
We've been calling these devices nodes, and we'll keep doing that, but it's also important to understand the concepts of servers and clients.
The simplest way to think of a server is as something that provides data to something requesting that data.
The thing receiving the data is referred to as a client.
While we often talk about nodes being servers or clients, the reason our definition uses a word as vague as something, is because it's not just nodes that can be servers or clients.

Individual computer programs running on
the same node can be servers and clients to each other too.
It's also important to call out that most devices aren't purely a server or a client.
Almost all nodes are both at some point in time.
Quite the multitasking overachievers.
That all being said, in most network topographies,
each node is primarily either a server or a client.
Sometimes we refer to an email server as an email server,
even though it's itself a client of a DNS server.
Why? Because its primary reason for existing is to serve data to clients.
Likewise, if a desktop machine occasionally acts as
a server in the sense that it provides data to another computer,
its primary reason for existing is to fetch data from servers,
so that the user at the computer can do their work.
To sum up, a server is anything that can provide data to a client,
but we also use the words to refer to
the primary purpose of various nodes on our network.

The Physical Layer

The physical layer consists of devices and
means of transmitting bits across computer networks.
A bit is the smallest representation of data that a computer can understand.
It's a one or a zero.
These ones and
zeros sends across networks at the lowest level are what make up the frames and
packets of data that we'll learn about when we cover the other layers.
The takeaway is that it doesn't matter whether you're streaming your
favorite song, emailing your boss, or using an ATM,
what you're really doing is sending ones and zeros across the physical layer
of the many different networks between you and the server you're interacting with.
A standard copper network cable,
once connected to devices on both ends, will carry a constant electrical charge.
Ones and zeros are sent across those network cables
through a process called modulation.
Modulation is a way of varying the voltage of this charge moving across the cable.
When used for computer networks,
this kind of modulation is more specifically known as line coding.
It allows devices on either end of a link to understand that an electrical charge
in a certain state is a zero, and in another state is a one.
Through this seemingly simple techniques, modern networks are capable of moving
10 billion ones and zeros across a single network cable every second.

The most common type of cabling used for connecting
computing devices is known as twisted pair.
It's called a twisted pair cable because it features
pairs of copper wires that are twisted together.
These pairs act as a single conduit for information,
and their twisted nature helps protect against
electromagnetic interference and crosstalk from neighboring pairs.
A standard cat six cable has eight wires

consisting of four twisted pairs inside a single jacket.
Exactly how many pairs are actually in
use depends on the transmission technology being used.
But in all modern forms of networking,
it's important to know that these cables allow for duplex communication.
Duplex communication is the concept that
information can flow in both directions across the cable.
On the flip side, a process called simplex communication is unidirectional.
Think about a baby monitor,
where the transmission of data only goes in
one direction making it a simplex communication.
A phone call on the other hand is duplex since both parties can listen and speak.
The way networking cables ensure that duplex communication is possible is
by reserving one or two pairs for communicating in one direction.
They then use the other one or two pairs for communicating in the other direction.
So, devices on either side of
a networking link can both communicate with each other at the exact same time.
This is known as full duplex.
If there's something wrong with the connection,
you might see a network link degrade and report itself as operating as half-duplex.
Half-duplex means that, while communication is possible in each direction,
only one device can be communicating at a time.

The final steps of how the physical layer works take
place at the end points our network links.
Twisted Pair network cables are terminated with
a plug that takes the individual internal wires and exposes them.
The most common plug is known as an RJ-45 or Registered Jack 45.
It's one of many cable plugs specifications but
by far the most common in computer networking.
A network cable with an RJ-45 plug can connect to an RJ-45 network port.
Network ports are generally directly
attached to the devices that make up a computer network.
Switches would have many network ports because their purpose is to connect many devices.
But servers and desktops usually only have one or two.
Your laptop, tablet or phone probably don't have any.
But we'll get to wireless networking in the later module.
Most network ports have two small LEDs.
One is the link light and the other is the activity light.
The link light will be lit when a cable is properly
connected to two devices that are both powered on.
The activity light will flash when data is actively transmitted across the cable.
A long time ago,
the flashing in the activity light corresponded
directly to the one's and zero's being sent.
Today, computer networks are so fast that the activity light doesn't
really communicate much other than if there's any traffic or not.
On switches, sometimes the same LED is used for both link and activity status.
It might even indicate other things like links speed.
You'll have to read up on a particular piece of hardware you're working with but there
will almost always be some troubleshooting data available to you through port lights.
Sometimes a network port isn't connected directly to a device.
Instead, there might be network ports mounted in a wall or underneath your desk.

These ports are generally connected to the network via cables
ran through the walls that eventually end at a patch panel.
A. Patch panel is a device containing many net ports but it does no other work.
It's just a container for the endpoints of many runs of cable.
Additional cables are then generally ran from a patch panel to switches or
routers to provide network access to the computers at the other end of those links.

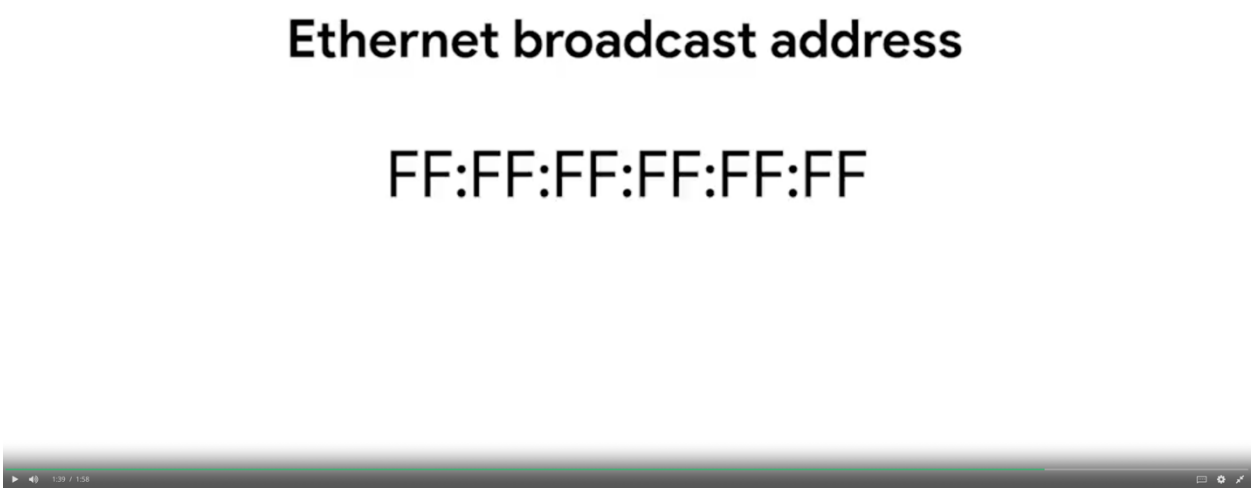
The Data Link Layer

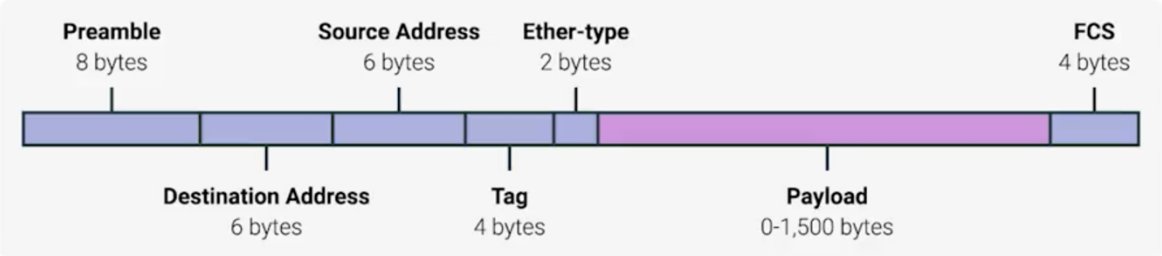
The protocol most widely used to send data across individual links is known as Ethernet.
Ethernet and the data link layer provide a means for
software at higher levels of the stack to send and receive data.
One of the primary purposes of this layer is to essentially abstract away
the need for any other layers to care
about the physical layer and what hardware is in use.
By dumping this responsibility on the data link layer, the Internet,
transport and application layers can all operate the
same no matter how the device they're running on is connected.
So, for example, your web browser doesn't need to know if it's
running on a device connected via a twisted pair or a wireless connection.
It just needs the underlying layers to send and receive data for it.
Ethernet, as a protocol,
solved the problem of collision detection of hubs, before switches came.
We generally abbreviate this to CSMA/CD.
CSMA/CD is used to determine when
the communications channels are clear and when the device is free to transmit data.
The way CSMA/CD works is actually pretty simple.
If there's no data currently being transmitted on the network segment,
a node will feel free to send data.
If it turns out that two or more computers end up trying to send data at the same time,
the computers detect this collision and stop sending data.
Each device involved with the collision then waits
a random interval of time before trying to send data again.
This random interval helps to prevent all the computers
involved in the collision from colliding
again the next time they try to transmit anything.
When a network segment is a collision domain,
it means that all devices on that segment
receive all communication across the entire segment.
This means we need a way to identify which node the transmission was actually meant for.
This is where something known as
a media access control address or MAC address comes into play.
A MAC address is
a globally unique identifier attached to an individual network interface.
It's a 48-bit number normally represented by six groupings of two hexadecimal numbers.
Just like how binary is a way to represent numbers with only two digits,
hexadecimal is a way to represent numbers using 16 digits.
Since we don't have numerals to represent any individual digit larger than nine,
hexadecimal numbers employed the letters
A, B, C, D, E, and F

to represent the numbers 10,
11, 12, 13, 14, and 15.
Another way to reference each group of numbers in a MAC address is an octet.
An octet, in computer networking,
is any number that can be represented by 8 bits.
In this case, two hexadecimal digits can represent the same numbers that 8 bits can.
Now, you may have noticed that we mentioned that MAC addresses are globally unique,
which might have left you wondering how that could possibly be.
The short answer is that a 48-bit number is much larger than you might expect.
The total number of a possible MAC addresses that could exist is 2 to the power 48
or 281,474,976,710,656
unique possibilities.
That's a whole lot of possibilities.
A MAC address is split into two sections.
The first three octets of a MAC address are known as
the organizationally unique identifier or OUI.
These are assigned to individual hardware manufacturers by
the IEEE or the Institute of Electrical and Electronics Engineers.
This is a useful bit of information to keeping
your back pocket because it means that you can
always identify the manufacturer of a network interface purely by its MAC address.
The last three octets of MAC address can be
assigned in any way that the manufacturer would like with
the condition that they only assign
each possible address once to keep all MAC addresses globally unique.
Ethernet uses MAC addresses to ensure that the data it
sends has both an address for the machine that sent the transmission,
as well as the one that the transmission was intended for.
In this way, even on a network segment,
acting as a single collision domain,
each node on that network knows when traffic is intended for it.

So far, we've discussed ways for
one device to transmit data to one other device.
This is what's known as unicast.
A unicast transmission is always meant for just one receiving address.
At the Ethernet level,
this is done by looking at a special bit in the destination MAC address.
If the least significant bit in the first octet of a destination address is set
to zero, it means that Ethernet frame is intended for only the destination address.
This means it would be sent to all devices on the collision domain, but
only actually received and processed by the intended destination.
If the least significant bit in the first octet of a destination address
is set to one, it means you're dealing with a multicast frame.
A multicast frame is similarly set to all devices on the local network signal.
What's different is that it will be accepted or discarded by each device
depending on criteria aside from their own hardware MAC address.
Network interfaces can be configured to accept lists of configured multicast
addresses for these sort of communication.
The third type of Ethernet transmission is known as broadcast.
An Ethernet broadcast is sent to every single device on a LAN.
This is accomplished by using a special destination known as a broadcast address.
The Ethernet broadcast address is all Fs.

Ethernet broadcasts are used so that devices can learn more about each other.





(Picture represents a frame if a tag such as VLAN is present, if no ta exists, there wont be any tag layer in the frame)

A data packet is an all-encompassing term
that represents any single set of binary data being sent across a network link.
Data packets at the Ethernet level are known as Ethernet frames.
An Ethernet frame is a highly structured collection of information
presented in a specific order.
This way network interfaces at the physical layer can convert a string of
bits, travelling across a link into meaningful data or vice versa.
Almost all sections of an Ethernet frame are mandatory and
most of them have a fixed size.
The first part of an Ethernet frame is known as the preamble.
A preamble is 8 bytes or 64 bits long and can itself be split into two sections.
The first seven bytes are a series of alternating ones and zeros.
These act partially as a buffer between frames and
can also be used by the network interfaces to synchronize internal clocks they use,
to regulate the speed at which they send data.
This last byte in the preamble is known as the SFD or start frame delimiter.
This signals to a receiving device that the preamble is over and
that the actual frame contents will now follow.
Immediately following the start frame delimiter,
comes the destination MAC address.
This is the hardware address of the intended recipient.
Which is then followed by the source MAC address, or
where the frame originated from.
Don't forget that each MAC address is 48 bits or 6 bytes long.
The next part of an Ethernet frame is called the EtherType field.
It's 16 bits long and used to describe the protocol of the contents of the frame.
It's worth calling out that instead of the EtherType field,
you could also find what's known as a VLAN header. (tag)
It indicates that the frame itself is what's called a VLAN frame.
If a VLAN header is present, the EtherType field follows it.
VLAN stands for virtual LAN.
It's a technique that lets you have multiple logical LANs
operating on the same physical equipment.
Any frame with a VLAN tag will only be delivered
out of a switch interface configured to relay that specific tag.
This way you can have a single physical network that operates
like it's multiple LANs.
VLANs are usually used to segregate different forms of traffic.
So you might see a company's IP phones operating on one VLAN,
while all desktops operate on another.
After this, you'll find a data payload of an Ethernet frame.
A payload in networking terms is the actual data being transported,
which is everything that isn't a header.
The data payload of a traditional Ethernet frame
can be anywhere from 46 to 1500 bytes long.
This contains all of the data from higher layers such as the IP, transport and
application layers that's actually being transmitted.
Following that data we have what's known as a frame check sequence.
This is a 4-byte or 32-bit number that represents a checksum value for
the entire frame.
This checksum value is calculated by performing
what's known as a cyclical redundancy check against the frame.
A cyclical redundancy check or CRC, is an important concept for

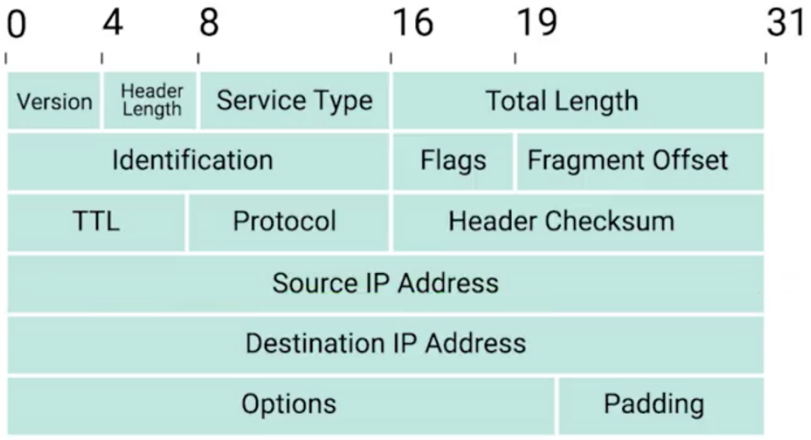
data integrity and is used all over computing, not just network transmissions.
A CRC is basically a mathematical transformation that uses polynomial
division to create a number that represents a larger set of data.
Anytime you perform a CRC against a set of data,
you should end up with the same checksum number.
The reason it's included in the Ethernet frame is so
that the receiving network interface can infer if it received uncorrupted data.
When a device gets ready to send an Internet frame,
it collects all the information we just covered, like the destination and
originating MAC addresses, the data payload and so on.
Then it performs a CRC against that data and attaches the resulting checksum number
as the frame check sequence at the end of the frame.
This data is then sent across a link and received at the other end.
Here, all the various fields of the Ethernet frame are collected and
now the receiving side performs a CRC against that data.
If the checksum computed by the receiving end doesn't match the checksum
in the frame check sequence field, the data is thrown out.
This is because some amount of data must have been lost or
corrupted during transmission.
It's then up to a protocol at a higher layer to decide if that data
should be retransmitted.
Ethernet itself only reports on data integrity.
It doesn't perform data recovery.

Week 2

The Network Layer: IP

IP addresses are 32-bit long numbers made up of 4 octets, and
each octet is normally described in decimal numbers.
8 bits of data, or a single octet,
can represent all decimal numbers from 0 to 255.
For example, 12.34.56.78 is a valid IP address.
But 123.456.789.100 would not be,
because it has numbers larger than could be represented by 8 bits.
This format is known as dotted decimal notation.
The important thing to know for now is that IP addresses are distributed in large
sections to various organizations and companies,
instead of being determined by hardware vendors.
This means that IP addresses are more hierarchical, and
easier to store data about than physical addresses are.
Think of IBM, which owns every single IP that has the number 9 as the first octet.
At a very high level, this means that if an Internet router needs to figure out
where to send a data packet intended for the IP address 9.0.0.1,
that router only has to know to get it to one of IBM's routers.
That router can handle the rest of the delivery process from there.
It's important to call out that IP addresses belong to the networks,
not the devices attached to those networks.
So your laptop will always have the same MAC address, no matter where you use it.
But it'll have a different IP address assigned to it at an Internet cafe than

it would when you're at home.
The LAN at the Internet cafe or the LAN at your house would each be individually responsible for handing out an IP address to your laptop if you power it on there.
For now, remember that on many modern networks, you can connect a new device.
And an IP address will be assigned to it automatically
through a technology known as Dynamic Host Configuration Protocol.
An IP address assigned this way is known as a dynamic IP address.
The opposite of this is known as a static IP address,
which must be configured on a node, manually.
In most cases, static IP addresses are reserved for servers and
network devices, while dynamic IP addresses are reserved for clients.



(IP address is 32 bits, 4 octets, each octet is 8 bits)

Just like all the data packets at the Ethernet layer have a specific name,
Ethernet frames, so do packets at the network layer.
Under the IP protocol,
a packet is usually referred to as an IP datagram.
Just like any Ethernet frame,
an IP datagram is a highly structured series of fields that are strictly defined.
The two primary sections of an IP datagram are the header and the payload.
You'll notice that an IP datagram header
contains a lot more data than an Ethernet frame header does.
The very first field is four bits,
and indicates what version of Internet protocol is being used.

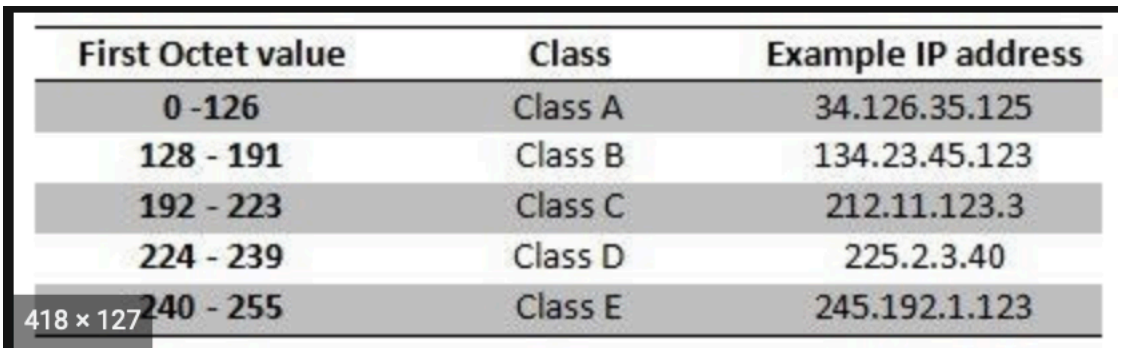
The most common version of IP is version four or IPv4.
Version six or IPv6,
is rapidly seeing more widespread adoption,
but we'll cover that in a later module.
After the version field,
we have the Header Length field.
This is also a four bit field that declares how long the entire header is.
This is almost always 20 bytes in length when dealing with IPv4.
In fact, 20 bytes is the minimum length of an IP header.
You couldn't fit all the data you need for
a properly formatted IP header in any less space.
Next, we have the Service Type field.
These eight bits can be used to specify details
about quality of service or QoS technologies.
The important takeaway about QoS is that there are services that allow routers to
make decisions about which IP datagram may be more important than others.
The next field is a 16 bit field,
known as the Total Length field.
It's used for exactly what it sounds like;
to indicate the total length of the IP datagram it's attached to.
The identification field, is a 16-bit number that's used to group messages together.
IP datagrams have a maximum size
and you might already be able to figure out what that is.
Since the Total Length field is 16 bits,
and this field indicates the size of an individual datagram,
the maximum size of a single datagram is the largest number you
can represent with 16 bits: 65,535.
If the total amount of data that needs to be sent
is larger than what can fit in a single datagram,
the IP layer needs to split this data up into many individual packets.
When this happens, the identification field is used so that the receiving end
understands that every packet with
the same value in that field is part of the same transmission.
Next up, we have two closely related fields.
The flag field and the Fragmentation Offset field.
The flag field is used to indicate if a datagram is allowed to be fragmented,
or to indicate that the datagram has already been fragmented.
Fragmentation is the process of taking
a single IP datagram and splitting it up into several smaller datagrams.
While most networks operate with similar settings in
terms of what size an IP datagram is allowed to be,
sometimes, this could be configured differently.
If a datagram has to cross from a network allowing
a larger datagram size to one with a smaller datagram size,
the datagram would have to be fragmented into smaller ones.
The fragmentation offset field contains values used by the receiving end to
take all the parts of
a fragmented packet and put them back together in the correct order.
Let's move along to The Time to Live or TTL field.
This field is an 8-bit field that indicates how
many router hops a datagram can traverse before it's thrown away.
Every time a datagram reaches a new router,
that router decrements the TTL field by one.

Once this value reaches zero,
a router knows it doesn't have to forward the datagram any further.
The main purpose of this field is to make sure that when there's
a misconfiguration in routing that causes an endless loop,
datagrams don't spend all eternity trying to reach their destination.
An endless loop could be when router A thinks router B is the next hop,
and router B thinks router A is the next hop, spoiler alert.
In an upcoming module,
you'll learn that the TTL field has valuable troubleshooting qualities,
but secrets like these are only released to those who keep going.
After the TTL field,
you'll find the Protocol field.
This is another 8-bit field that contains
data about what transport layer protocol is being used.
The most common transport layer protocols are TCP and UDP,
and we'll cover both of those in detail in the next few lessons.
So next, we find the header checksum field.
This field is a checksum of the contents of the entire IP datagram header.
It functions very much like the Ethernet checksum field we discussed in the last module.
Since the TTL field has to be recomputed at every router that a datagram touches,
the checksum field necessarily changes, too.
After all of that, we finally get to two very important fields,
the source and destination IP address fields.
Remember that an IP address is a 32 bit number so,
it should come as no surprise that these fields are each 32 bits long.
Up next, we have the IP options field.
This is an optional field and is used to set
special characteristics for datagrams primarily used for testing purposes.
The IP options field is usually followed by a padding field.
Since the IP options field is both optional and variable in length,
the padding field is just a series of zeros used
to ensure the header is the correct total size.
Now that you know about all of the parts of an IP datagram,
you might wonder how this relates to what we've learned so far.
You might remember that in our breakdown of an Ethernet frame,
we mentioned a section we described as the data payload section.
This is exactly what the IP datagram is,
and this process is known as encapsulation.
The entire contents of an IP datagram are
encapsulated as the payload of an Ethernet frame.
You might have picked up on the fact that our IP datagram also has a payload section.
The contents of this payload are the entirety of
a TCP or UDP packet which we'll cover later.

IP addresses can be split into two sections,
the network ID, and the host ID.
Earlier, we mentioned that IBM owns all IP addresses
that having a nine as the value of the first octet in an IP address.
If we take an example IP address of 9.100.100.100,
the network ID would be the first octet,
and the host ID, would be the second,
third and fourth octets.
The address class system is a way of

defining how the global IP address space is split up.
There are three primary types of address classes.
Class A, class B,
and class C**. Class A addresses are**
those where the first octet is used for the network ID,
and the last three are used for the host ID.
Class B addresses are where the first two octets are used for the network ID,
and the second two,
are used for the host ID.
Class C addresses, as you might have guessed,
are those where the first three octets are used for the network ID,
and only the final octet is used for the host ID.
Each address class represents a network of vastly different size.
For example, since a class A network has a total of 24 bits of host ID space,
this comes out to two to the twenty-fourth,
or 16,777,216 individual addresses.
Compare this with a class C network,
which only has eight bits of host ID space.
For a class C network,
this comes out to two to the eighth, or 256 addresses.
You can also tell exactly what address class
an IP address belongs to just by looking at it.
If the very first bit of an IP address is a zero,
it belongs to a class A network,
if the first bits are one, zero,
it belongs to a class B network.
Finally, if the first bits are 110,
it belongs to a class C network.
Since humans aren't great at thinking in binary,
it's good to know that this also translates nicely to
how these addresses are represented in dotted decimal notation.
You might remember that each octet in an IP address is eight bits,
which means each octet can take a value between zero and 255.
If the first bit has to be a zero,
as it is with the class A address,
the possible values for the first octet are zero through 127.
This means that any IP address with
a first octet with one of those values is a class A address.
Similarly, class B addresses are restricted to those that begin
with the first octet value of 128 through 191,
and class C addresses begin with the first octet value of 192 through to 223.
You might notice that this doesn't cover every possible IP address.
That's because there are two other IP address classes,
but they're not quite as important to understand.
Class D addresses always begin with the bits 1110,
and are used for multicasting,
which is how a single IP datagram can be sent to an entire network at once.
These addresses begin with decimal values between 224 and 239.
Lastly, class C addresses make up all of the remaining IP addresses,
but they're unassigned and only used for testing purposes.
In practical terms, this class system has mostly been replaced
by a system known as CIDR or Classless inter-domain routing.
But the address class system is still in place in many ways,

and is important to understand for anyone
looking for a well rounded networking education



First Octet value	Class	Example IP address
0 -126	Class A	34.126.35.125
128 - 191	Class B	134.23.45.123
192 - 223	Class C	212.11.123.3
224 - 239	Class D	225.2.3.40
240 - 255	Class E	245.192.1.123

418 × 127

Congrats! You now understand how both MAC addresses are used
at the Data Link Layer and how IP addresses are used at the network layer.
Now we need to discuss how these two separate addresses types relate to each other.
This is where Address Resolution Protocol or ARP comes into play.
ARP is a protocol used to discover
the hardware address of a node with a certain IP address.
Once an IP datagram has been fully formed,
it needs to be encapsulated inside an Ethernet frame.
This means, that the transmitting device needs
a destination MAC address to complete the Ethernet frame header.
Almost all network connected devices while retaining local ARP table.
In ARP table is just a list of IP addresses and the MAC addresses associated with them.
Let's say we want to send some data to the IP address 10.20.30.40.
It might be the case that this destination doesn't have an entry in the ARP table.
When this happens, the node that wants to send data sends
a broadcast ARP message to the Mac broadcast address which is all Fs. FF:FF:FF:FF
These kinds of broadcast ARP messages are
delivered to all computers on the local network.
When the network interface that's been assigned an IP of
10.20.30.40 receives this ARP broadcast,
it sends back what's known as an ARP response.
This response message will contain the MAC address for the network interface in question.
Now, the transmitting computer knows what MAC address to put in
the destination hardware address field and the Ethernet frame is ready for delivery.
It will also likely store this IP address in its local ARP table
so that it won't have to send
an ARP broadcast the next time he needs to communicate with this IP.
Handy. ARP table entries generally expire
after a short amount of time to ensure changes in the network are accounted for.

The Network Layer: Subnetting

Subnetting is the process of taking a large network and splitting
it up into many individual smaller subnetworks or subnets.

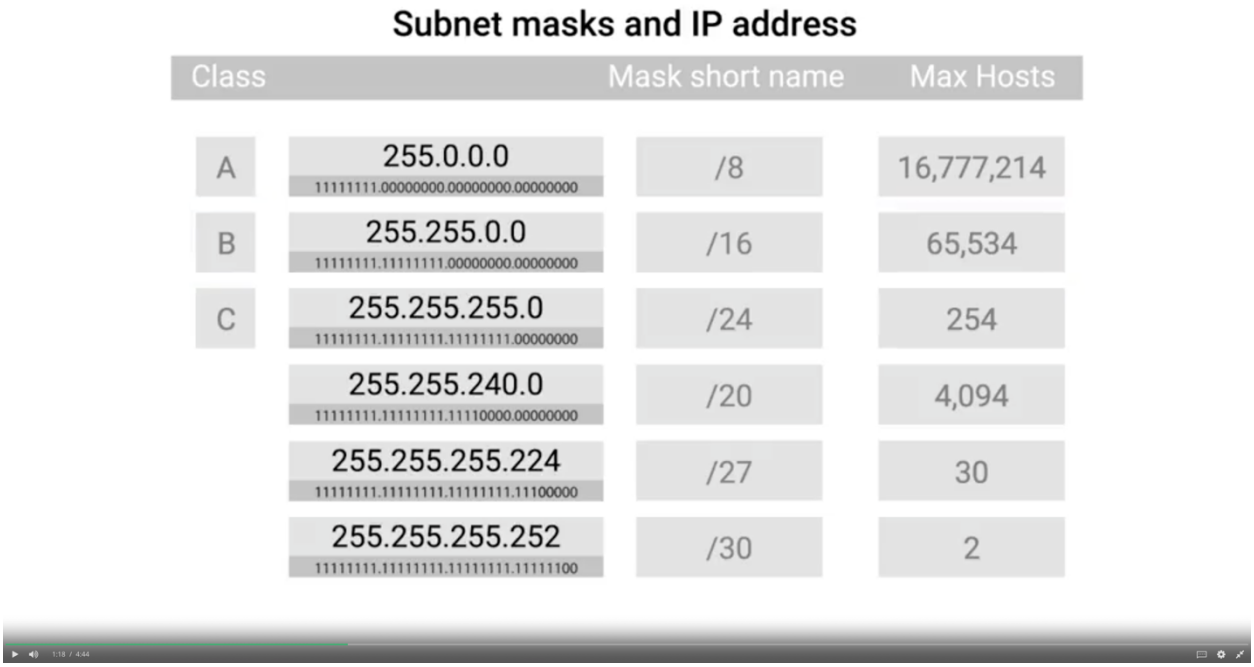
As you might remember, from the last lesson,
address classes give us a way to break the total global IP space into discrete networks.
If you want to communicate with the IP address 9.100.100.100,
core routers on the Internet know that this IP belongs to the 9.0.0.0 class A network.
They then route the message to the gateway router
responsible for the network by looking at the network ID.
A gateway router specifically serves as the entry and exit path to a certain network.
You can contrast this with core Internet routers,
which might only speak to other core routers.
Once your packet gets to the gateway router for the 9.0.0.0 class A network,
that router is now responsible for getting
that data to the proper system by looking at the host ID.
This all makes sense until you remember that a single class A network
contains 16,777,216 individual IPs.
That's just way too many devices to connect to the same router.
This is where subnetting comes in.
With subnets, you can split your large network up into many smaller ones.
These individual subnets will all have
their own gateway routers serving as the ingress and egress point for each subnet.

So far, we've learned about network IDs,
which are used to identify networks,
and host IDs, which are used to identify individual hosts.
If we want to split things up even further,
and we do, we'll need to introduce a third concept, the subnet ID.
You might remember that an IP address is just a 32-bit number.
In a world without subnets,
a certain number of these bits are used for the network ID,
and a certain number of the bits are used for the host ID.
In a world with subnetting,
some bits that would normally comprise the host ID are actually used for the subnet ID.
With all three of these IDs representable by a single IP address,
we now have a single 32-bit number that can be
accurately delivered across many different networks.
At the internet level, core routers only care about the network ID
and use this to send the datagram
along to the appropriate gateway router to that network.
That gateway router then has some additional information that it can use to send
that datagram along to
the destination machine or the next router in the path to get there.
Finally, the host ID is used by
that last router to deliver the datagram to the intended recipient machine.
Subnet IDs are calculated via what's known as a subnet mask.
Just like an IP address,
subnet masks are 32-bit numbers that are normally written now as four octets in decimal.
The easiest way to understand how subnet masks work is to compare one to an IP address.

Let's work with the IP address 9.100.100.100 again.
You might remember that each part of an IP address is an octet,
which means that it consists of eight bits.
The number 9 in binary is just 1001.
But since each octet needs eight bits,
we need to pad it with some zeros in front.

As far as an IP address is concerned,
having a number 9 as the first octet is actually represented as 0000 1001.
Similarly, the numeral 100 as an eight-bit number is 0110 0100.
So, the entire binary representation of the IP address
9.100.100.100 is a lot of ones and zeros.
A subnet mask is a binary number that has two sections.
The beginning part, which is the mask itself is
a string of ones just zeros come after this,
the subnet mask, which is the part of the number with all the ones,
tells us what we can ignore when computing a host ID.
The part with all the zeros tells us what to keep.
Let's use the common subnet mask of 255.255.255.0.
This would translate to 24 ones followed by eight zeros.
The purpose of the mask or the part that's all ones is
to tell a router what part of an IP address is the subnet ID.
You might remember that we already know how to get the network ID for an IP address.
For 9.100.100.100, a Class A network,
we know that this is just the first octet.
This leaves us with the last three octets.
Let's take those remaining octets and imagine
them next to the subnet mask in binary form.
The numbers in the remaining octets that have a
corresponding one in the subnet mask are the subnet ID.
The numbers in the remaining octets that have a corresponding zero are the host ID.
The size of a subnet is entirely defined by its subnet mask.
So for example, with the subnet mask of 255.255.255.0,
we know that only the last octet is available for host IDs,
regardless of what size the network and subnet IDs are.
A single eight-bit number can represent 256 different numbers,
or more specifically, the numbers 0-255.
This is a good time to point out that, in general,
a subnet can usually only contain two less than the total number of host IDs available.
Again, using a subnet mask of 255.255.255.0,
we know that the octet available for host IDs can contain the numbers 0-255,
but zero is generally not used and
255 is normally reserved as a broadcast address for the subnet.
This means that, really,
only the numbers 1-254 are available for assignment to a host.
While this total number less than two approach is almost always true,
generally speaking, you'll refer to the number of
host available in a subnet as the entire number.
So, even if it's understood that two addresses aren't available for assignment,
you'd still say that eight bits of host IDs space have 256 addresses available, not 254.
This is because those other IPs are still IP addresses,
even if they aren't assigned directly to a node on that subnet.
Now, let's look at a subnet mask that doesn't draw
its boundaries at an entire octet or eight bits of address.
The subnet mask 255.255.255.224 would translate to 27 ones followed by five zeros.
This means that we have five bits of host ID space or a total of 32 addresses.
This brings up a shorthand way of writing subnet masks.
Let's say we're dealing with our old friend
9.100.100.100 with a subnet mask of 255.255.255.224.
Since that subnet mask represents 27 ones followed by five zeros,

a quicker way of referencing this is with the notation /27.
The entire IP and subnet mask can be written now as 9.100.100.100/27.
Neither notation is necessarily more common than the other,
so it's important to understand both.



CIDR (Class less inter domain routing) is an even more flexible approach to describing blocks of IP addresses.
It expands on the concept of subnetting by using subnet masks to demarcate networks.
To demarcate something means to set something off.
When discussing computer networking,
you'll often hear the term demarcation point to
describe where one network or system ends and another one begins.
In our previous model, we relied on a network ID,
subnet ID, and host ID to deliver an IP datagram to the correct location.
With CIDR, the network ID and subnet ID are combined into one.
CIDR is where we get
this shorthand slash notation that we discussed in the earlier video on subnetting.
This slash notation is also known as CIDR notation.

CIDR	Subnet Mask	Total IPs	Usable IPs
/32	255.255.255.255	1	1
/31	255.255.255.254	2	0
/30	255.255.255.252	4	2
/29	255.255.255.248	8	6
/28	255.255.255.240	16	14
/27	255.255.255.224	32	30
/26	255.255.255.192	64	62
/25	255.255.255.128	128	126
/24	255.255.255.0	256	254
/23	255.255.254.0	512	510
/22	255.255.252.0	1024	1022
/21	255.255.248.0	2048	2046
/20	255.255.240.0	4096	4094
/19	255.255.224.0	8192	8190
/18	255.255.192.0	16,384	16,382
/17	255.255.128.0	32,768	32,766
/16	255.255.0.0	65,536	65,534
/15	255.254.0.0	131,072	131,070
/14	255.252.0.0	262,144	262,142
/13	255.248.0.0	524,288	524,286
/12	255.240.0.0	1,048,576	1,048,574
/11	255.224.0.0	2,097,152	2,097,150
/10	255.192.0.0	4,194,304	4,194,302
/9	255.128.0.0	8,388,608	8,388,606
/8	255.0.0.0	16,777,216	16,777,214
/7	254.0.0.0	33,554,432	33,554,430
/6	252.0.0.0	67,108,864	67,108,862
/5	248.0.0.0	134,217,728	134,217,726
/4	240.0.0.0	268,435,456	268,435,454
/3	224.0.0.0	536,870,912	536,870,910
/2	192.0.0.0	1,073,741,824	1,073,741,822
/1	128.0.0.0	2,147,483,648	2,147,483,646
/0	0.0.0.0	4,294,967,296	4,294,967,294

The Network Layer: Routing

From a very basic standpoint, a router is a network device
that forwards traffic depending on the destination address of that traffic.
A router is a device that has at least two network interfaces,
since it has to be connected to two networks to do its job.
Basic routing has just a few steps.
One, a router receives a packet of data on one of its interfaces.
Two, the router examines the destination IP of this packet.
Three, the router then looks up the destination network of this IP
in its routing table.
Four, the router forwards that out though the interface that's closest to
the remote network as determined by additional info within the routing table.
These steps are repeated as often as needed until the traffic
reaches its destination.
Let's imagine a router connected to two networks.
We'll call the first network, Network A and give it
an address space of 192.168.1.0/24.
We'll call the second network, Network B and
give it an address space of 10.0.0.0/24.
The router has an interface on each network.
On Network A, it has an IP of 192.168.1.1 and
on Network B, it has an IP of 10.0.254.
Remember, IP addresses belong to networks, not individual nodes on a network.
A computer on Network A with an IP address of 192.168.1.100 sends a packet
to the address 10.0.0.10.
This computer knows that 10.0.0.10 isn't on its local subnet.
So it sends this packet to the MAC address of its gateway, the router.
The router's interface on Network A receives the packet
because it sees that destination MAC address belongs to it.
The router then trips away the Data Link Layer encapsulation,
leaving the network layer content, the IP datagram.
Now, the router can directly inspect the IP datagram header for
the destination IP field.
It finds the destination IP of 10.0.0.10.
The router looks at it's routing table and sees that Network B, or
the 10.0.0.0/24 network, is the correct network for the destination IP.
It also sees that, this network is only one hop away.
In fact, since it's directly connected, the router even has the MAC address for
this IP in its ARP table.
Next, the router needs to form a new packet to forward along to Network B.
It takes all of the data from the first IP datagram and duplicates it.
But decrements the TTL field by one and calculates a new checksum.
Then it encapsulates this new IP datagram inside of a new Ethernet frame.
This time, it sets its own MAC address of the interface on network B
as the source MAC address.
Since it has the MAC address of 10.0.0.10 in its ARP table,
it sets that as the destination MAC address.
Lastly, the packet is sent out of its interface on Network B and
the data finally gets delivered to the node living at 10.0.0.10.
That's a pretty basic example of how routing works, but
let's make it a little more complicated and introduce a third network.

Everything else is still the same.
We have network A whose address space is 192.168.1.0/24.
We have network B whose address space is 10.0.0/24.
The router that bridges these two networks still has the IPs of
192.168.1.1 on Network A and 10.0.0.254 on Network B.
But let's introduce a third network, Network C.
It has an address space of 172.16.1.0/23.
There is a second router connecting network B and network C.
It's interface on network B has an IP of 10.0.0.1 and
its interface on Network C has an IP of 172.16.1.1.
This time around our computer at 192.168.1.100
wants to send some data to the computer that has an IP of 172.16.1.100.
We'll skip the Data Link Layer stuff, but
remember that it's still happening, of course.
The computer at 192.168.1.100 knows that 172.16.1.100 is not on its local network,
so it sends a packet to its gateway, the router between Network A and Network B.
Again, the router inspects the content of this packet.
It sees a destination address of 172.16.1.100 and
through a lookup of its routing table, it knows that the quickest
way to get to the 172.16.1.0/23 network is via another router.
With an IP of 10.0.0.1.
The router decrements the TTL field and
sends it along to the router of 10.0.0.1.
This router then goes through the motions,
knows that the destination IP of 172.16.1.100 is directly connected and
forwards the packet to its final destination.
That's the basics of routing.
The only difference between our examples and
how things work on the Internet is scale.
Routers are usually connected to many more than just two networks.
Very often, your traffic may have to cross a dozen routers before it reaches
its final destination.
And finally, in order to protect against breakages,
core Internet routers are typically connected in a mesh,
meaning that there might be many different paths for a packet to take.
Still, the concepts are all the same.
Routers inspect the destination IP, look at the routing table to determine which
path is the quickest and forward the packet along the path.
This happens over and over.
Every single pocket making up every single bit of traffic all over the Internet
at all times

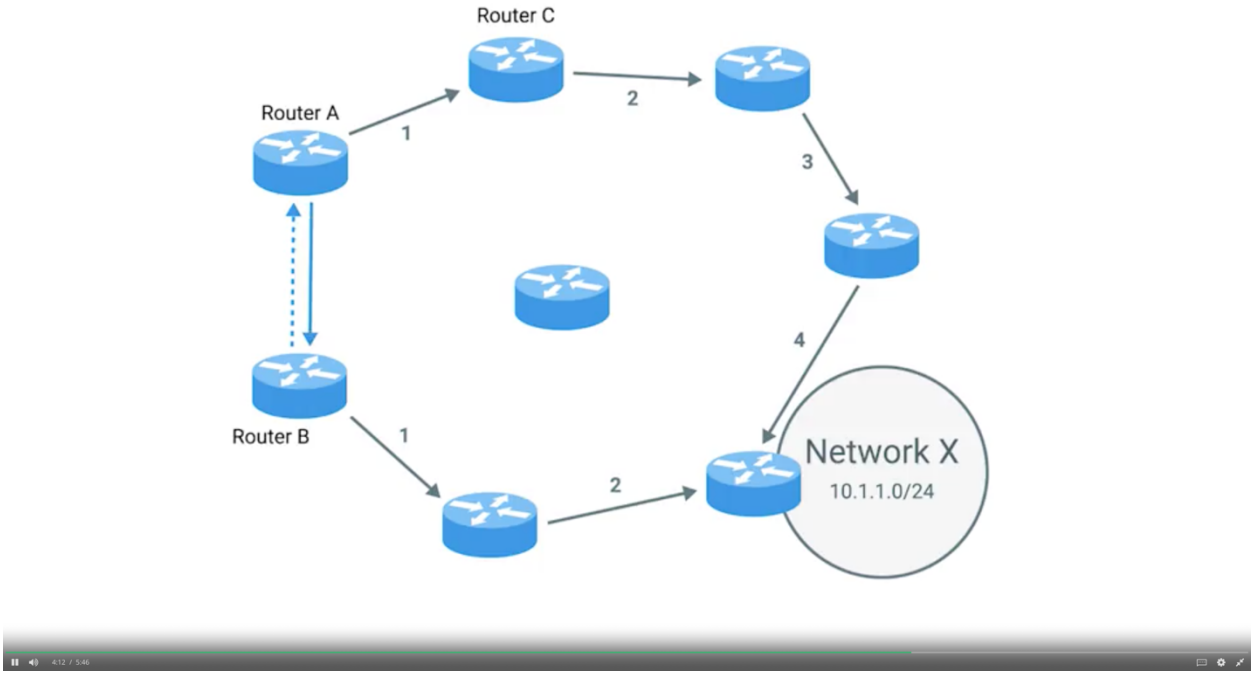
Routing tables can vary a ton depending on the make and class of the router,
but they all share a few things in common.
The most basic routing table will have four columns.
Destination network, this column would contain
a row for each network that the router knows about,
this is just the definition of the remote network,
a network ID, and the net mask.
These could be stored in one column inside a notation,
or the network ID and net mask might be in a separate column.
Either way, it's the same concept,
the router has a definition for a network and therefore

knows what IP addresses might live on that network.
When the router receives an incoming packet,
it examines the destination IP address and determines which network it belongs to.
A routing table will generally have a catchall entry,
that matches any IP address that it doesn't have an explicit network listing for.
Next hop, this is the IP address of the next router that should receive data intended for
the destination networking question or this could just state
the network is directly connected and that there aren't any additional hops needed.
Total hops, this is the crucial part to understand routing and how routing tables work,
on any complex network like the Internet,
there will be lots of different paths to get from point A to point
B. Routers try to pick the shortest possible path at
all times to ensure timely delivery of data but
the shortest possible path to
a destination network is something that could change over time,
sometimes rapidly, intermediary routers could go down,
links could become disconnected,
new routers could be introduced,
traffic congestion could cause certain routes to become too slow to use.
We'll get to know how routers know the shortest path in an upcoming video.
For now, it's just important to know that for each next hop and each destination network,
the router will have to keep track of how far away that destination currently is.
That way, when it receives updated information from neighboring routers,
it will know if it currently knows about
the best path or if a new better path is available.
Interface, the router also has to know which of
its interfaces it should for traffic matching the destination network out of.
In most cases, routing tables are pretty simple.
The really impressive part is that,
many core Internet routers have millions of rows in the routing tables.
These must be consulted for every single packet that
flows through a router on its way to its final destination.

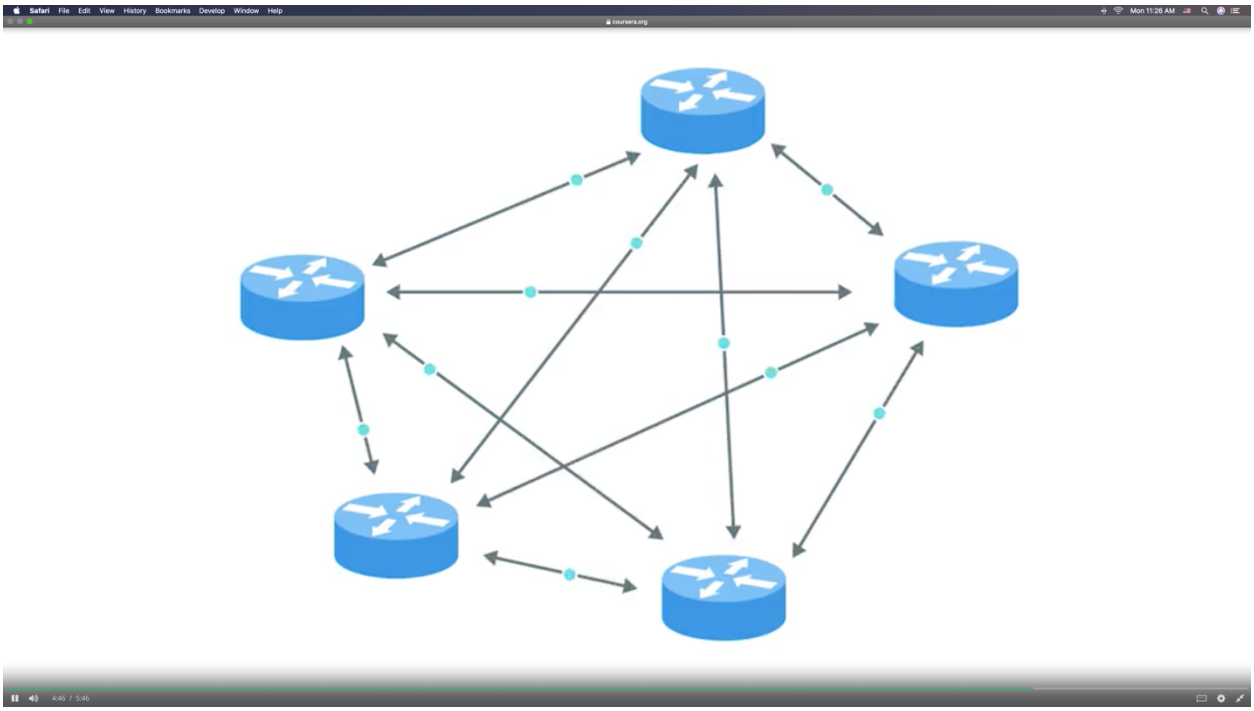
In order to learn about the world around them,
routers use what are known as routing protocols.
These are special protocols the routers use to speak to each other
in order to share what information they might have.
This is how a router on one side of the planet can eventually learn about
the best path to a network on the other side of the planet.
Routing protocols fall into two main categories: interior gateway protocols and
exterior gateway protocols.
Interior gateway protocols are further split into two categories: Link
state routing protocols and distance-vector protocols.
In this video we'll cover the basics of interior gateway protocols.
Interior gateway protocols are used by routers to share information
within a single autonomous system.
In networking terms, an autonomous system is a collection of networks
that all fall under the control of a single network operator.
The best example of this would be a large corporation
that needs to route data between their many offices, and
each of which might have their own local area network.
Another example is the many routers employed by an Internet service provider

whose reaches are usually national in scale.
You can contrast this with exterior gateway protocols which are used for
the exchange of information between independent autonomous systems.

The two main types of interior gateway protocols
are link state routing protocols and distance-vector protocols.
Their goals are super similar but the routers that employ them share different
kinds of data to get the job done.
Distance vector protocols are an older standard.
A router using a distance vector protocol basically just takes its routing table
which is a list of every network known to it and
how far away these networks are in terms of hops.
Then the router sends this list to every neighboring router,
which is basically every router directly connected to it.
In computer science, a list is known as a vector.
This is why a protocol that just sends a list of distances to networks,
is known as a distance vector protocol.
With a distance vector protocol,
routers don't really know that much about the total state of an autonomous system.
They just have some information about their immediate neighbors.
For a basic glimpse into how distance vector protocols work,
let's look at how two routers might influence each other's routing tables.
Router A has a routing table with a bunch of entries.
One of these entries is for 10.1.1.0/24 network which we'll refer to as Network X.
Router A believes that the quickest path to network X
is through its own interface two, which is where router C is connected.
Router A knows that sending data intended for Network X through interface
two to Router C means it'll take four hops to get to the destination.
Meanwhile, Router B is only two hops removed from Network X and
this is reflected in its routing table.
Router B, using a distance vector protocol,
sends the basic contents of it's routing table to Router A.
Router A sees that Network X is only two hops away from Router B.
Even with the extra hop to get from Router A to Router B,
this means that network X is only three hops away
from router A if it forwards data to router B instead of router C.
Armed with this new information, router A updates its routing table to reflect this.
In order to reach network X in the fastest way,
it should forward traffic through its own interface 1 to router B.
Now, distance vector protocols are pretty simple.
But they don't allow for a router to have much information about the state of the world outside of their own direct neighbors.
Because of this,
a router might be slow to react to a change in the network far away from it.



This is why link-state protocols were eventually invented.
Routers using a link-state protocol take a more sophisticated approach to determining the best path to a network.
Linked state protocols got their name because each router advertises the state of the link of each of its interfaces.
These interfaces can be connected to other routers or they could be direct connections to networks.
The information about each router is propagated to every other router on the autonomous system.
This means that every router on the system knows every detail about every other router in the system.
Each router then uses this much larger set of information and runs complicated algorithms against it to determine what the best path to any destination network might be.
Link state protocols require both more memory in order to hold all of this data and also much more processing power.
This is because it has to run algorithms against this data in order to determine the quickest path to update the routing tables.
As computer hardware has become more powerful and cheaper over the years, link state protocols have mostly made distance vector protocols outdated.



Exterior gateway protocols are used to communicate data between routers
representing the edges of an autonomous system.
Since routers sharing data using interior gateway protocols
are all under control of the same organization.
Routers use exterior gateway protocols
when they need to share information across different organizations.
Exterior gateway protocols are really key to the Internet operating how
it does today.
So, thanks exterior gateway protocols.
The Internet is an enormous mesh of autonomous systems.
At the highest levels, core Internet routers
need to know about autonomous systems in order to properly forward traffic.
Since autonomous systems are known and
defined collections of networks, getting data to the edge router
of an autonomous system is the number one goal of core Internet routers.
The IANA or the Internet Assigned Numbers Authority,
is a non-profit organization that helps manage things like IP address allocation.
The Internet couldn't function without a single authority for
these sorts of issues.
Otherwise, anyone could try and
use any IP space they wanted, which would cause total chaos online.
Along with managing IP address allocation, the IANA is also responsible for
ASN, or Autonomous System Number allocation.
ASNs are numbers assigned to individual autonomous systems.
Just like IP addresses, ASNs are 32-bit numbers.
But, unlike IP addresses, they're normally referred to
as just a single decimal number, instead of being split out into readable bits.
There are two reasons for this.
First, IP addresses need to be able to represent a network ID portion and
a host ID portion for each number.

This is more easily accomplished by splitting the number in four sections of
eight bits, especially back in the day when address classes ruled the world.
An ASN, never needs to change in order for it to represent more networks or hosts.
Its just the core Internet routing tables that need to be updated to know what
the ASN represents.
Second, ASNs are looked at by humans, far less often, than IP addresses are.
So because it can be useful to be able to
look at the IP 9.100.100.100 and know that 9.0.0.0/8 address
space is owned by IBM, ASNs represent entire autonomous systems.
Just being able to look up the fact that AS19604 belongs to IBM is enough.
Unless you one day end up working at an Internet service provider,
understanding more details about how exterior gateway protocols work,
is out of scope for most people in IT.
But grasping the basics of autonomous systems, ASNs, and
how core Internet routers route traffic between them,
is important to understand some of the basic building blocks of the Internet.

Even as far back as 1996,
it was obvious that the internet was growing at a rate that couldn't be sustained.
When IP was first defined,
it defined an IP address as a single 32-bit number.
A single 32-bit number can represent 4,294,967,295
unique numbers which definitely sounds like a lot,
but as of 2017,
there are an estimated 7.5 billion humans on earth.
This means that the IPv4 standard doesn't even
have enough IP addresses available for every person on the planet.
It also can account for entire data centers filled with
thousands and thousands of computers
required for large scale technology companies to operate.
So, in 1996, RFC 1918 was published.
RFC stands for Request for Comments,
and has a long standing way for those responsible for keeping the internet
running to agree upon the standard requirements to do so.
RFC 1918, outlined a number of
networks that would be defined as non-routable address space.
Non-routable address space is basically exactly what it sounds like.
They are ranges of IPs set aside for use by anyone that cannot be routed to.
Not every computer connected to the internet needs to be
able to communicate with every other computer connected to the internet.
Non-routable address space allows for nodes on such a network to communicate
with each other but no gateway router will
attempt to forward traffic to this type of network.
This might sound super limiting,
and in some ways, it is.
In a future module,
we'll cover a technology known as NAT or Network Address Translation.
It allows for computers on
non-routable address space to communicate with other devices on the internet.
But for now, let's just discuss non-routable address space in a vacuum.
RFC 1918 defined three ranges of
IP addresses that will never be routed anywhere by co-routers.
That means that they belong to no one and that anyone can use them.

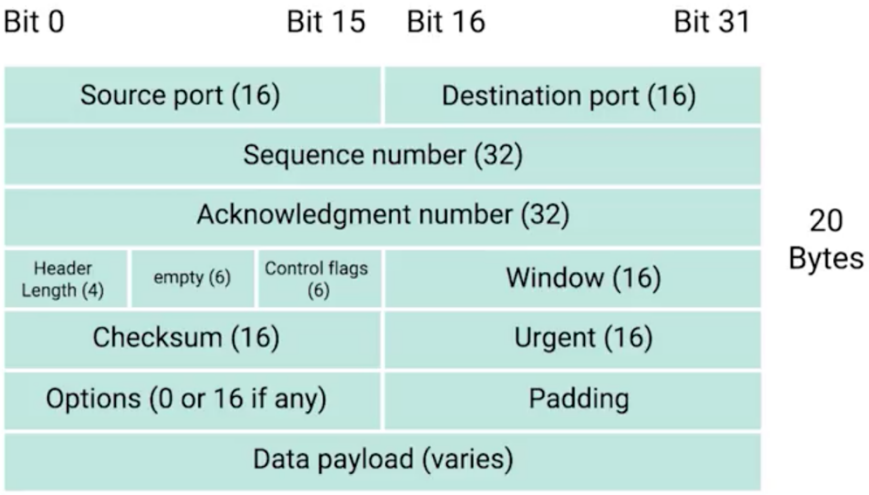
In fact, since they are separated from the way traffic moves across the internet,
there's no limiting to how many people might use
these addresses for their internal networks.
The primary three ranges of non-routable address space are 10.0.0.0/8,
172.16.0.0/12, and 192.168.0.0/16.
These ranges are free for anyone to use for their internal networks.
It should be called out that interior gateway protocols will route these address spaces.
So, they are appropriate for use within
an autonomous system but exterior gateway protocols will not.
We've covered a lot in this module and congratulations to you for sticking with it.
Next up, we'll cover the transport and application layers.
But first up, another quiz.
You can do it.
And remember, you can always go back and review the material as much as you need to.

Week 3

The Transport Layer

The transport layer is responsible for
lots of important functions of reliable computer networking.
These include multiplexing and
demultiplexing traffic, establishing long running connections and
ensuring data integrity through error checking and data verification.
The transport layer has the ability to multiplex and demultiplex,
which sets this layer apart from all others.
Multiplexing in the transport layer means that nodes on the network have the ability
to direct traffic toward many different receiving services.
Demultiplexing is the same concept, just at the receiving end,
it's taking traffic that's all aimed at the same node and
delivering it to the proper receiving service.
The transport layer handles multiplexing and demultiplexing through ports.
A port is a 16-bit number
that's used to direct traffic to specific services running on a networked computer.
Different network services run while listening on specific ports for
incoming requests.
For example, the traditional port for HTTP or unencrypted web traffic is port 80.
If we want tor request the webpage from a web server running on a computer listening
on IP 10.1.1.100, the traffic would be directed to port 80 on that computer.
Ports are normally denoted with a colon after the IP address.
So the full IP and port in this scenario could be described as 10.1.1.100:80.
When written this way, it's known as a socket address or socket number.
The same device might also be running an FTP or file transfer protocol server.
FTP traditionally listens on port 21, so if you wanted to establish a connection to
an FTP server running on the same IP that our example web server was running on,
you direct traffic to 10.1.1.100 port 21.
You might find yourself working in IT support at a small business.
In these environments,
a single server could host almost all of the applications needed to run a business.
The same computer might host an internal website, the mail server for

the company, file server for sharing files, a print server for
sharing network printers, pretty much anything.
This is all possible because of multiplexing and demultiplexing, and
the addition of ports to our addressing scheme.



Just like how an Ethernet frame encapsulates an IP datagram,
an IP datagram encapsulates a TCP segment.
Remember that an Ethernet frame has a payload section
which is really just the entire contents of an IP datagram.
Remember also that an IP datagram has
a payload section and this is made up of what's known as a TCP segment.
A TCP segment is made up of a TCP header and a data section.
This data section, as you might guess,
is just another payload area for where the application layer places its data.
A TCP header itself is split into lots of fields containing lots of information.
First, we have the source port and the destination port fields.
The destination port is the port of the service the traffic is intended for,
which we talked about in the last video.
A source port is a high numbered port chosen from
a special section of ports known as ephemeral ports.
We'll cover ephemeral ports in more detail in a little bit.
For now, it's enough to know that a source port is
required to keep lots of outgoing connections separate.
You know how a destination port, say port 80,
is needed to make sure traffic reaches a web server running on a certain IP?
Similarly, a source port is needed so that when the web server replies,
the computer making the original request can send
this data to the program that was actually requesting it.

It is in this way that when it web server responds to your requests to view
a webpage that this response gets
received by your web browser and not your word processor.
Next up is a field known as the sequence number.
This is a 32-bit number that's used to keep track of where in
a sequence of TCP segments this one is expected to be.
You might remember that lower on our protocol stack,
there are limits to the total size of what we send across the wire.
In Ethernet frame, it's usually limited in size to 1,518 bytes,
but we usually need to send way more data than that.
At the transport layer,
TCP splits all of this data up into many segments.
The sequence number in a header is used to keep track of
which segment out of many this particular segment might be.
The next field, the acknowledgment number,
is a lot like the sequence number.
The acknowledgment number is the number of the next expected segment.
In very simple language,
a sequence number of one and an acknowledgement number of
two could be read as this is segment one,
expect segment two next.
The data offset field comes next.
This field is a four-bit number that communicates
how long the TCP header for this segment is.
This is so that the receiving network device
understands where the actual data payload begins.
Then, we have six bits that are reserved for the six TCP control flags.
We'll cover the control flags and what they each mean in the next video,
so flag this for later.
The next field is a 16-bit number known as the TCP window.
A TCP window specifies the range of
sequence numbers that might be sent before an acknowledgement is required.
As we'll cover in more details soon,
TCP is a protocol that's super reliant on acknowledgements.
This is done in order to make sure that all expected data is actually being
received and that the sending device doesn't
waste time sending data that isn't being received.
The next field is a 16-bit checksum.
It operates just like the checksum fields at the IP and Ethernet level.
Once all of this segment has been ingested by a recipient,
the checksum is calculated across the entire segment and is compared with the checksum
in the header to make sure that there was no data lost or corrupted along the way.
The Urgent pointer field is used in conjunction with one of
the TCP control flags to point out
particular segments that might be more important than others.
This is a feature of TCP that hasn't really ever seen
adoption and you'll probably never find it in modern networking.
Even so, it's important to know what all sections of the TCP header are.
Next up, we have the options field.
Like the urgent pointer field,
this is rarely used in the real world,
but it's sometimes used for more complicated flow control protocols.
Finally, we have some padding which is just a sequence of zeros

to ensure that the data payload section begins at the expected location.

As a protocol, TCP establishes connections used to send long chains of segments of data.
You can contrast this with the protocols that are lower in the networking model.
These include IP and Ethernet,
which just send individual packets of data.
As an IT support specialist,
you need to understand exactly how that works,
so you can troubleshoot issues,
where network traffic may not be behaving in the expected manner.
The way TCP establishes a connection,
is through the use of different TCP control flags,
used in a very specific order.
Before we cover how connections are established and closed,
let's first define the six TCP control flags.
We'll look at them in the order that they appear in a TCP header. Heads up.
This isn't necessarily in
the same order of how frequently they're set, or how important they are.
The first flag is known as URG,
this is short for Urgent.
A value of one here indicates that the segment is
considered urgent and that the urgent pointer field has more data about this.
Like we mentioned in the last video,
this feature of TCP has never really had wide spreaded option and isn't normally seen.
The second flag is ACK, short for acknowledge.
A value of one in this field means that
the acknowledgment number field should be examined.
The third flag is PSH,
which is short for Push.
This means, that the transmitting device wants the receiving device to push
currently- buffered data to the application on the receiving end as soon as possible.
A buffer is a computing technique,
where a certain amount of data is held somewhere,
before being sent somewhere else.
This has lots of practical applications.
In terms of TCP,
it's used to send large chunks of data more efficiently.
By keeping some amount of data in a buffer,
TCP can deliver more meaningful chunks of data to the program waiting for it.
But in some cases,
you might be sending a very small amount of information,
that you need the listening program to respond to immediately.
This is what the push flag does.
The Fourth flag is RST, short for Reset.
This means, that one of the sides in a TCP connection hasn't been
able to properly recover from a series of missing or malformed segments.
It's a way for one of the partners in a TCP connection to basically say,
"Wait, I can't put together what you mean,
let's start over from scratch."
The fifth flag is SYN,
which stands for Synchronize.
It's used when first establishing a TCP connection and make sure
the receiving end knows to examine the sequence number field.

And finally, our six flag is FIN,
which is short for Finish.
When this flag is set to one,
it means the transmitting computer doesn't have
any more data to send and the connection can be closed.
For a good example of how TCP control flags are used,
let's check out how a TCP connection is established.
Computer A will be
our transmitting computer and computer B will be our receiving computer.
To start the process off, computer A,
sends a TCP segment to computer B with this SYN flag set.
This is computer A's way of saying,
"Let's establish a connection and look at my sequence number field,
so we know where this conversation starts."
Computer B then responds with a TCP segment,
where both the SYN and ACK flags are set.
This is computer B's way of saying, "Sure,
let's establish a connection and I acknowledge your sequence number."
Then computer A responds again with just the ACK flag set,
which is just saying, "I acknowledge your acknowledgement.
Let's start sending data." I love how polite they are to each other.
This exchange involving segments that have SYN,
SYN/ACK and ACK sets,
happens every single time a TCP connection is established anywhere.
And is so famous that it has a nickname.
The three way handshake.
A handshake is a way for two devices to ensure
that they're speaking the same protocol and will be able to understand each other.
Once the three way handshake is complete,
the TCP connection is established.
Now, computer A is free to send whatever data it wants to computer B and vice versa.
Since both sides have now sent SYN/ACK pairs to each other,
a TCP connection in this state is operating in full duplex.
Each segment sent in either direction should be responded
to by TCP segment with the ACK field set.
This way, the other side always knows what has been received.
Once one of the devices involved
with the TCP connection is ready to close the connection,
something known as a four way handshake happens.
The computer ready to close the connection,
sends a FIN flag,
which the other computer acknowledges with an ACK flag.
Then, if this computer is also ready to close the connection,
which will almost always be the case.
It will send a FIN flag.
This is again responded to by an ACK flag.
Hypothetically, a TCP connection can stay open in
simplex mode with only one side closing the connection.
But this isn't something you'll run into very often.

A socket is the instantiation of an endpoint in a potential TCP connection.
An instantiation is the actual implementation of something defined elsewhere.
TCP sockets require actual programs to instantiate them.

You can contrast this with a port which is more of a virtual descriptive thing.
In other words, you can send traffic to any port you want,
but you're only going to get a response if a program has opened a socket on that port.
TCP sockets can exist in lots of states.
And being able to understand what those mean will help you
troubleshoot network connectivity issues as an IT support specialist.
We'll cover the most common ones here.
LISTEN. Listen means that a TCP socket is ready and listening for incoming connections.
You'd see this on the server side only.
SYN_SENT. This means that a synchronization request has been sent,
but the connection hasn't been established yet.
You'd see this on the client side only.
SYN_RECEIVED.
This means that a socket previously in a listener state,
has received a synchronization request and sent a SYN_ACK back.
But it hasn't received the final ACK from the client yet.
You'd see this on the server side only.
ESTABLISHED. This means that the TCP connection is in working order,
and both sides are free to send each other data.
You'd see this state on both the client and server sides of the connection.
This will be true of all the following socket states,
too. So keep that in mind.
FIN_WAIT. This means that a FIN has been sent,
but the corresponding ACK from the other end hasn't been received yet.
CLOSE_WAIT. This means that the connection has been closed at the TCP layer,
but that the application that opened
the socket hasn't released its hold on the socket yet.
CLOSED. This means that the connection has been fully terminated,
and that no further communication is possible.
There are other TCP socket states that exist.
Additionally, socket states and their names,
can vary from operating system to operating system.
That's because they exist outside of the scope of the definition of TCP itself.
TCP, as a protocol,
is universal in how it's used since every device speaking to
TCP protocol has to do this in the exact same way for communications to be successful.
Choosing how to describe the states of
a socket at the operating system level isn't quite as universal.
When troubleshooting issues at the TCP layer,
make sure you check out
the exact socket state definitions for the systems you're working with.

So far, we have mostly focused on TCP which is a connection-oriented protocol.
A connection-oriented protocol is one that establishes a connection,
and uses this to ensure that all data has been properly transmitted.
A connection at the transport layer implies that
every segment of data sent is acknowledged.
This way, both ends of the connection always know which bits of
data have definitely been delivered to the other side and which haven't.
Connection-oriented protocols are important
because the Internet is a vast and busy place,
and lots of things could go wrong while trying to get data from point A to point B.
You might remember from our lesson about

the physical air that even some minor crosstalk from
a neighboring twisted pair in
the same cable can be enough to make a cyclical redundancy check fail.
This could cause the entire frame to be discarded.
Yikes. If even a single bit doesn't get transmitted properly,
the resulting data is often incomprehensible by the receiving end.
And remember that at the lowest level,
a bit is just an electrical signal within a certain voltage range.
But there are plenty of other reasons why traffic might not reach
its destination beyond liners. It could be anything.
Pure congestion might cause a router to drop
your traffic in favor of forwarding more important traffic,
or a construction company could cut a fiber cable connecting two YSPs.
Anything's possible. Connection-oriented protocols like TCP protect
against this by forming connections and through the constant stream of acknowledgements.
Our protocols at lower levels of our network model like IP and
Ethernet do use checksums to ensure that all the data they received was correct.
But did you notice that we never discussed
any attempts to resending data that doesn't pass this check?
That's because that's entirely up to the transport layer protocol.
At the IP or Ethernet level,
if a checksum doesn't compute,
all of that data is just discarded.
It's up to TCP to determine when to resend
this data since TCP expects an ACK for every bit of data it sends,
it's in the best position to know what data successfully got
delivered and it can make the decision to resend a segment if needed.
This is another reason why sequence numbers are so important.
While TCP will generally send all segments in sequential order,
they may not always arrive in that order.
If some of the segments had to be resend due to errors at lower layers,
it doesn't matter if they arrive slightly out of order.
This is because sequence numbers allow for all of the data to
be put back together in the right order. It's pretty handy.
Now, as you might have picked up on,
there's a lot of overhead with connection-oriented protocols like TCP.
You have to establish the connection,
you have to send a stream of constant streams of acknowledgements,
you have to tear the connection down at the end.
That all accounts for a lot of extra traffic.
While this is important traffic,
it's really only useful if you absolutely
positively have to be sure your data reaches its destination.
You can contrast this with connectionless protocols.
The most common of these is known as UDP,
or User Datagram Protocol.
Unlike TCP, UDP doesn't rely on
connections and it doesn't even support the concept of an acknowledgement.
With UDP, you just set a destination port and send the packet.
This is useful for messages that aren't super important.
A great example of UDP is streaming video.
Let's imagine that each UDP datagram is a single frame of a video.
For the best viewing experience,

you might hope that every single frame makes it to the viewer
but it doesn't really matter if a few get lost along the way.
A video will still be pretty watchable unless it's missing a lot of its frames.
By getting rid of all the overhead of TCP,
you might actually be able to send higher quality video with UDP.
That's because you'll be saving more of the available bandwidth for actual data
transfer instead of the overhead of establishing
connections and acknowledging delivered data segments.

A firewall is just a device that blocks traffic that meets certain criteria.
Firewalls are a critical concept to keeping a network secure since they
are the primary way you can stop traffic you don't want from entering a network.
Firewalls can actually operate at lots of different layers of the network.
There are firewalls that can perform inspection of application layer traffic,
and firewalls that primarily deal with blocking ranges of IP addresses.
The reason we cover firewalls here is that they're most commonly used
at the transportation layer.
Firewalls that operate at the transportation layer
will generally have a configuration that enables them to block traffic
to certain ports while allowing traffic to other ports.
Let's imagine a simple small business network.
The small business might have one server which hosts multiple network services.
This server might have a web server that hosts the company's website,
while also serving as the file server for a confidential internal document.
A firewall placed at the perimeter of the network could be configured to allow
anyone to send traffic to port 80 in order to view the web page.
At the same time, it could block all access for external IPs to any other port.
So that no one outside of the local area network could access the file server.
Firewalls are sometimes independent network devices, but
it's really better to think of them as a program that can run anywhere.
For many companies and almost all home users, the functionality of a router and
a firewall is performed by the same device.
And firewalls can run on individual hosts instead of being a network device.
All major modern operating systems have firewall functionality built-in.
That way, blocking or allowing traffic to various ports and
therefore to specific services can be performed at the host level as well.

The Application Layer

Now, we can finally talk about how those actual applications send and
receive data using the application layer.
Just like with every other layer,
TCP segments have a generic data section to them.
As you might have guessed, this payload section is actually the entire contents of
whatever data applications want to send to each other.
It could be contents of a web page, if a web browser is connecting to a web server.
This could be the streaming video content of your Netflix app on your PlayStation
connecting with the Netflix servers.
It could be the contents of a document your word processor is sending
to a printer.
And many more things.

There are a lot of protocols used at the application layer,
and they are numerous and diverse.
At the data link layer, the most common protocol is ethernet.
I should call out that wireless technologies do use other protocols at
this layer, which we'll cover in a future module.
At the network layer, use of IP is everywhere you look.
At the transport layer, TCP and UDP cover most of the use cases.
But at the application layer, there are just so
many different protocols in use, it wouldn't make sense for us to cover them.
Even so, one concept you can take away about application layer protocols
is that they're still standardized across application types.
Let's dive a little deeper into web servers and web browsers for an example.
There are lots of different web browsers.
You could be using Chrome, Internet Explorer, Safari, you name it.
They'll need to speak the protocol.
The same thing is true for web servers.
In this case, the web browser would be the client, and
the web server would be the server.
The most popular web servers are Microsoft IIS, Apache, and nginx.
But they also need to speak the same protocol.
This way, you ensure that no matter which browser you're using,
you'd still be able to be speak to any server.
For web traffic, the application layer protocol is known as HTTP.
All of these different web browsers and web servers have to communicate using
the same HTTP protocol specifications in order to ensure interoperability.
The same is true for most other classes of application.
You might have dozens of choices for an FTP client, but
they all need to speak the FTP protocol in the same way.

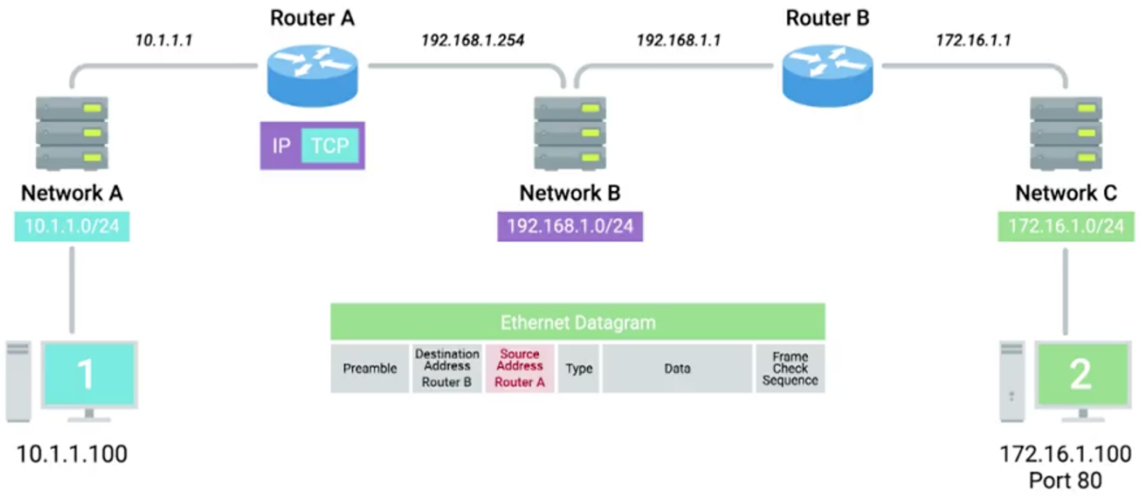
All the Layers: United

Now that you know the basics of how every layer of out network model works, let's go
through an exercise to look at how everything works at every step of the way.
Spoiler alert, things are about to get a little geeky, in a good way.
Imagine three networks, network A will contain address space 10.1.1.0/24.
Network B Will contain address space 192.168.1.0/24,
and network C will be 172.16.1.0/24.
Router A sits between network A and network B.
With an interface configured with an IP of 10.1.1.1 on network A,
and an interface at 192.168.1.254 on network B.
There's a second router, router B, which connects networks B and C.
It has an interface on network B with an IP address of 192.168.1.1,
and an interface on network C with an IP address of 172.16.1.1.
Now let's put a computer on one of the networks.
Imagine it's a desktop, sitting on someone's desk at the workplace.
It'll be our client in this scenario, and we'll refer to it as computer 1.
It's part of Network A and has been assigned an IP address of 10.1.1.100.
Now, let's put another computer on one of our other networks.
This one is a server in a data center, it'll act as our server in this scenario,
and we'll refer to it as computer 2.
It's part of network C, and has been assigned an IP address of
172.16.1.100, and has a web server listening on port 80.

An end user sitting at computer 1 opens up a web browser and
enters 172.16.1.100 into the address bar, let's see what happens.
The web browser running on computer 1 knows it's been ordered to
retrieve a web page from 172.16.1.100.
The web browser communicates with the local networking stack, which is the part
of the operating system responsible for handling networking functions.
The web browser explains that it's going to want to establish
a TCP connection to 172.16.1.100, port 80.
The networking stack will now examine its own subnet.
It sees that it lives on the network 10.1.1.0/24,
which means that the destination 172.16.1.100 is on another network.
At this point, computer 1 knows that it'll have to send any data to its gateway for
routing to a remote network.
And it's been configured with a gateway of 10.1.1.1.
Next, computer 1 looks at its ARP table to determine what MAC address
of 10.1.1.1 is, but it doesn't find any corresponding entry.
Uh-oh, it's okay, computer A crafts an ARP request for
an IP address of 10.1.1.1,
which it sends to the hardware broadcast address of all Fs.
This ARP discovery request is sent to every node on the local network.
When router A receives this ARP message, it sees that it's the computer
currently assigned the IP address of 10.1.1.1.
So it responds to computer 1 to let it know about its
own MAC address of 00:11:22:33:44:55.
Computer 1 receives this response and now knows the hardware address of its gateway.
This means that it's ready to start constructing the outbound packet.
Computer 1 knows that it's being asked by the web browser to form
an outbound TCP connection, which means it'll need an outbound TCP port.
The operating system identifies the ephemeral port of 50000 as being
available, and opens a socket connecting the web browser to this port.
Since this is a TCP connection, the networking stack knows that before
it can actually transmit any of the data the web browser wants it to,
it'll need to establish a connection.
The networking stack starts to build a TCP segment.
It fills in all the appropriate fields in the header,
including a source port of 50000 and a destination port of 80.
A sequence number is chosen and is used to fill in the sequence number field.
Finally, the SYN flag is set, and a checksum for the segment is calculated and
written to the checksum field.
Our newly constructed TCP segment is now passed along to the IP layer
of the networking stack.
This layer constructs an IP header.
This header is filled in with the source IP, the destination IP, and
a TTL of 64, which is a pretty standard value for this field.
Next, the TCP segment is inserted as the data payload for the IP datagram.
And a checksum is calculated for the whole thing.
Now that the IP datagram has been constructed,
computer 1 needs to get this to its gateway, which it now knows
has a MAC address of 00:11:22:33:44:55,
so an Ethernet Datagram is constructed.
All the relevant fields are filled in with the appropriate data, most notably,
the source and destination MAC addresses.
Finally, the IP datagram is inserted as the data payload of the Ethernet frame,

and another checksum is calculated. Now we have an entire Ethernet frame ready to be sent across the physical layer.
The network interface connected to computer 1 sends this binary data
as modulations of the voltage of an electrical current running across
a CAT6 cable that's connected between it and a network switch.
This switch receives the frame and inspects the destination MAC address.
The switch knows which of its interfaces this MAC address is attached to,
and forwards the frame across only the cable connected to this interface.
At the other end of this link is router A, which receives the frame and
recognizes its own hardware address as the destination.
Router A knows that this frame is intended for itself.
So it now takes the entirety of the frame and calculates a checksum against it.
Router A compares this checksum with the one in the Ethernet frame header and
sees that they match.
Meaning that all of the data has made it in one piece.
Next, Router A strips away the Ethernet frame,
leaving it with just the IP datagram.
Again, it performs a checksum calculation against the entire datagram.
And again, it finds that it matches, meaning all the data is correct.
It inspects the destination IP address and
performs a lookup of this destination in its routing table.
Router A sees that in order to get data to
the 172.16.1.0/24 network, the quickest path is one hop away via
Router B, which has an IP of 192.168.1.1.
Router A looks at all the data in the IP datagram,
decrements the TTL by 1, calculates a new checksum reflecting
that new TTL value, and makes a new IP datagram with this data.
Router B knows that it needs to get this datagram to router B,
which has an IP address of 192.168.1.1.
It looks at its ARP table, and sees that it has an entry for
192.168.1.1.
Now router A can begin to construct an Ethernet frame
with the MAC address of its interface on network B as the source.
And the MAC address on router B's interface on network B as the destination.
Once the values for all fields in this frame have been filled out,
router A places the newly constructed IP datagram into the data payload field.
Calculates a checksum, and places this checksum into place, and
sends the frame out to network B.
Just like before, this frame makes it across network B, and
is received by router B.
Router B performs all the same checks, removes the the Ethernet
frame encapsulation, and performs a checksum against the IP datagram.
It then examines the destination IP address.
Looking at its routing table,
router B sees that the destination address of computer 2, or
172.16.1.100, is on a locally connected network.
So it decrements the TTL by 1 again, calculates a new checksum, and
creates a new IP datagram.
This new IP datagram is again encapsulated by a new Ethernet frame.
This one with the source and destination MAC address of router B and computer 2.
And the whole process is repeated one last time.
The frame is sent out onto network C,
a switch ensures it gets sent out of the interface that computer 2 is connected to.

Computer 2 receives the frame, identifies its own MAC address as the destination,
and knows that it's intended for itself.
Computer 2 then strips away the Ethernet frame, leaving it with the IP datagram.
It performs a CRC and recognizes that the data has been delivered intact.
It then examines the destination IP address and recognizes that as its own.
Next, computer 2 strips away the IP datagram,
leaving it with just the TCP segment.
Again, the checksum for this layer is examined, and everything checks out.
Next, computer 2 examines the destination port, which is 80.
The networking stack on computer 2 checks to ensure that there's an open
socket on port 80, which there is.
It's in the listen state, and held open by a running Apache web server.
Computer 2 then sees that this packet has the SYN flag set.
So it examines the sequence number and stores that, since it'll need to put that
sequence number in the acknowledgement field once it crafts the response.
After all of that, all we've done is get a single
TCP segment containing a SYN flag from one computer to a second one.
Everything would have to happen all over again for
computer 2 to send a SYN-ACK response to computer 1.
Then everything would have to happen all over again for
computer 1 to send an ACK back to computer 2, and so on and so on.



Week 4

Name Resolution

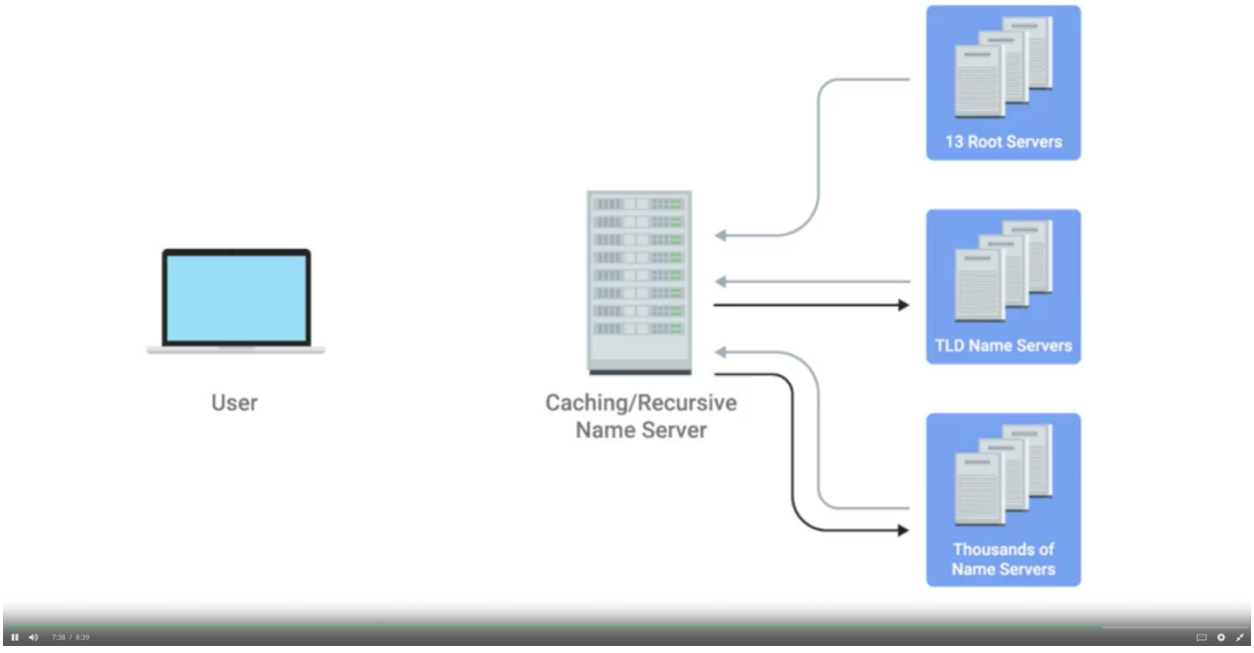
Humans are much better at remembering words.
That's where DNS, or domain name system, comes into play.
DNS is a global and highly distributed
network service that resolves strings of letters into IP address for you.
Let's say you wanted to check a weather website to see what the temperature is
going to be like.
It's much easier to type www.weather.com into a web
browser than it is to remember that one of the IP adresses for
this site is 184.29.131.121.
The IP address for a domain name can also change all the time for
a lot of different reasons.
A domain name is just the term we use for something that can be resolved by DNS.
In the example we just used, www.weather.com would be the domain name,
and the IP it resolves to could change, depending on a variety of factors.
Let's say that weather.com was moving their web server to a new data center.
Maybe they signed a new contract, or the old data center was shutting down.
By using DNS, an organization can just change what IP a domain name resolves to,
and the end user would never even know.
So, not only does DNS make it easier for humans to remember how to get to
a website, It also lets administrative changes happen behind the scenes
without an end user having to change their behavior.
Try to imagine a world where you'd have to remember every IP for
every website you visit, while also having to memorize new ones if something changed.
We'd spend our whole day memorizing numbers.
The importance of DNS for how the Internet operates, today, can't be overstated.
IP addresses might resolve to different things depending on where in
the world you are.
While most Internet communications travel at the speed of light,
the further you have to route data, the slower things will become.
In almost all situations, it's going to be quicker to transmit a certain amount of
data between places that are geographically close to each other.
If you're a global web company, you'd want people from all over the world to have
a great experience accessing your website.
So instead of keeping all of your web servers in one place,
you could distribute them across data centers across the globe.
This way, someone in New York, visiting a website,
might get served by a web server close to New York, while someone in New Delhi
might get served by a web server closer to New Delhi.
Again, DNS helps provide this functionality.
Because of its global structure, DNS lets organizations decide,
if you're in the region, resolve the domain name to this IP.
If you're in this other region, resolve this domain to this other IP.

At its most basic, DNS is a system that converts domain names into IP addresses.
It's the way humans are likely to remember and categorize
things resolved into the way computers prefer to think of things.

This process of using DNS to turn
a domain name into an IP address is known as name resolution.
Let's take a closer look at exactly how this works.
The first thing that's important to know is that DNS servers,
are one of the things that need to be specifically configured at a node on a network.
For a computer to operate on a modern network,
they need to have certain number of things configured.
Remember, that MAC addresses are hard coded and tied to specific pieces of hardware.
But we've also covered that the IP address, subnet mask,
and gateway for a host must be specifically configured,
a DNS server, is the fourth and final part of the standard modern network configuration.
These are almost always the four things that must be
configured for a host to operate on a network in an expected way.
I should call out, that a computer can operate just fine without
DNS or without a DNS server being configured,
but as we covered in the last video,
this makes things difficult for any human that might be using that computer.
There are five primary types of DNS servers; caching name servers,
recursive name servers, root name servers,
TLD name servers, and authoritative name servers.
As we dive deeper into these,
it's important to note that any given DNS server can fulfill many of these roles at once.
Caching and recursive name servers are
generally provided by an ISP or your local network.
Their purpose is to store domain name lookups for a certain amount of time.
As you'll see in a moment,
there are lots of steps in order to perform
a fully qualified resolution of a domain name.
In order to prevent this from happening every
single time a new TCP connection is established,
your ISP or local network will generally have a caching name server available.
Most caching name servers are also recursive name servers.
Recursive name servers are ones that perform full DNS resolution requests.
In most cases, your local name server will perform the duties of both,
but it's definitely possible for a name server to be
either just caching or just recursive.
Let's introduce an example to better explain how this works.
You and your friend are both connected to the same network and
you both want to check out Facebook.com,
your friend enters www.facebook.com into a web browser,
which means that their computer now needs to know the IP of
www.facebook.com in order to establish a connection.
Both of your computers are on the same network which usually means,
that they both been configured with the same name server.
So your friends computer ask the name server for the IP
of www.facebook.com which it doesn't know,
this name server now performs a fully recursive resolution to
discover the correct IP for www.facebook.com.
This involves a bunch of steps we'll cover in just a moment.
This IP is then both delivered to your friend's computer and stored locally in a cache.
A few minutes later you enter www.facebook.com into a web browser.
Again, your computer needs to know the IP for this domain,
so your computer asks the local name server it's been configured with,

which is the same one your friend's computer was just talking to.
Since the domain name www.Facebook.com had just been looked up,
the local name server still has the IP that it resolved to stored and
is able to deliver that back to your computer without having to perform a full lookup.
This is how the same servers act as a caching server.
All domain names in the global DNS system have a TTL or time to live.
This is a value in seconds,
that can be configured by the owner of a domain name for how long a name server is
allowed to cache in entry before it should
discard it and perform a full resolution again.
Several years ago, it was normal for these TTL's to be really long,
sometimes a full day or more.
This is because the general bandwidth available on the Internet was just much less,
so network administrators didn't want to waste what bandwidth
was available to them by constantly performing full DNS lookups.
As the Internet has grown and gone faster,
these TTL's for most domains have dropped to anywhere from a few minutes to a few hours.
But it's important to know that sometimes you still
run into a domain names with very lengthy TTL's,
it means that it can take up to the length of a total TTL for
a change in DNS record to be known to the entire Internet.
Now, let's look at what happens when
your local recursive server needs to perform a full recursive resolution.
The first step is always to contact a root named server,
there are 13 total root name servers and they're
responsible for directing queries toward the appropriate TLD name server.
In the past, these 13 root servers were distributed to very specific geographic regions,
but today, they're mostly distributed across the globe via anycast.
Anycast is a technique that's used to route traffic
to different destinations depending on factors like location,
congestion, or link health.
Using anycast, a computer can send a data gram to a specific IP but could
see it routed to one of many different actual destinations depending on a few factors.
This should also make it clear that there aren't really
only 13 physical route name servers anymore.
It's better to think of them as 13 authorities that
provide route name lookups as a service.
The root servers will respond to
a DNS lookup with the TLD name server that should be queried.
TLD stands for top level domain and
represents the top of the hierarchical DNS name resolution system.
A TLD is the last part of any domain name,
using www.facebook.com as an example again,
the dot com portion should be thought of as the TLD.
We'll go into more details about
the different components of a domain name in an upcoming lesson.
For each TLD in existence,
there is a TLD name server,
but just like with root servers,
this doesn't mean there's only physically one server in question,
it's most likely a global distribution of
any cast accessible servers responsible for each TLD.
The TLD name servers will respond again with a redirect,

this time informing the computer performing
the name lookup with what authoritative name server to contact.
Authoritative name servers are responsible for the last two parts of
any domain name which is the resolution at which
a single organization may be responsible for DNS lookups.
Using www.weather.com as an example,
the TLD name server would point a lookup at the authoritative server for Weather.com,
which would likely be controlled by the Weather Channel,
the organization itself that runs the site.
Finally, the DNS lookup could be redirected at the authoritative server for
weather.com which would finally provide the actual IP of the server in question.
This strict hierarchy is very important to the stability of the internet,
making sure that all full DNS resolutions go through
a strictly regulated and controlled series of lookups to get the correct responses,
is the best way to protect against malicious parties redirecting traffic.
Your computer will blindly send traffic to whatever IP it's told to.
So by using a hierarchical system controlled by trusted entities in the way DNS does,
we can better ensure that the responses to DNS lookups are accurate.
Now that you see how many steps are involved,
it should make sense why we trust our local name servers to cache DNS lookups,
its so that full lookup path doesn't have to happen for every single TCP connection.
In fact, your local computer from your phone to
a desktop will generally have its own temporary DNS cache as well,
that way, it doesn't have to bother
its local name server for every TCP connection either.



DNS is at application layer and works through UDP, not TCP at transport layer. This makes a connectionless, setup without flags,

acknowledgements or handshakes, making the whole lookup process fast.

Name Resolution in Practice

DNS in practice operates with a set of defined resource record types.
These allow for different kinds of DNS resolutions to take place.
There are dozens of different resource record types defined,
but a lot of them only serve very specialized purposes.
We'll cover the most basic ones here.
The most common resource record is known as an A record.
An A record is used to point a certain domain name at a certain IPv4 IP address.
In our earlier discussions of DNS,
we made the assumption that
the DNS resolver was asking for the A record for a domain name.
In its most basic use,
a single A record is configured for a single domain name.
But, a single domain name can have multiple A records, too.
This allows for a technique known as
DNS round robin to be used to balance traffic across multiple IPs.
Round robin is a concept that involves iterating
over a list of items one by one in an orderly fashion.
The hope is that this ensures
a fairly equal balance of each entry on the list that's selected.
Let's say we're in charge of a domain name www.microsoft.com.
Microsoft is a large company and their website likely sees a lot of traffic.
To help balance this traffic across multiple servers.
We configure four A records for www.microsoft.com
at the authoritative name server for the microsoft.com domain.
We'll use the IPs 10.1.1.1,
10.1.1.2, 10.1.1.3, and 10.1.1.4.
When the DNS Resolver performs a look-up of www.microsoft.com,
all four IPs would be returned in the order first configured: 10.1 1.1,
followed by 10.1.1.2, followed by 10.1.1.3, and finally, 10.1.1.4.
The DNS resolving computer would know that it should try to use the first entry,
10.1.1.1, but it knows about all four just in case a connection to 10.1.1.1 fails.
The next computer to perform a look up for
www.microsoft.com would also receive all four IPs in the response,
but the ordering will have changed.
The first entry would be 10.1.1.2, followed by 10.1.1.3,
followed by 10.1.1,4, and finally,
10.1.1.1 would be last on that list.
This pattern will continue for every DNS resolution attempt,
cycling through all of the A records configured
and balancing the traffic across these IPs.
That's the basics of how DNS round robin logic works.
Another resource record type that's becoming more and more popular is the Quad A record.
A Quad A record is very similar to an A record except that it
returns in IPv6 address instead of an IPv4 address.
We'll cover the details of IPv6 in a future module.

The CNAME record is also super common.
A CNAME record is used to redirect traffic from one domain to another.
Let's say that Microsoft runs their web servers at www.microsoft.com.
They also want to make sure that anyone that enters just
microsoft.com into their web browser will get properly redirected.
By configuring a CNAME record for microsoft.com that resolves to www.microsoft.com,
the resolving client would then know to perform
another resolution attempt, this time for www.microsoft.com,
and then use the IP returned by that second attempt.
CNAMEs are really useful because they ensure you only
have to change the canonical IP address of a server in one place.
In fact, CNAME is just shorthand for canonical name.
If we look again at our original example of making sure that visitors to both
microsoft.com and www.microsoft.com get to the same place.
We could do this in two ways.
We could set up identical A records for both microsoft.com
and www.microsoft.com domain names,
and this would work just fine.
But if the underlying IP address ever changes,
we need to change it in two places.
The A records for both nicrosoft.com and www.microsoft.com.
By setting up a CNAME
that points microsoft.com at www.microsoft.com,
you'd only have to change the A record for www.microsoft.com.
And you'd know that clients pointing at either domain would get the new IP address.
This might not seem like a huge deal with just two records to worry about,
but large companies with complex presences on
the web might have dozens of these kinds of redirections.
It's always easier to only have one source of truth.
Another important resource record type is the MX record.
MX stands for mail exchange and
this resource record is used in order to deliver e-mail to the correct server.
Many companies run their web and mail servers on different machines with different IPs,
so the MX record makes it easy to ensure
that email gets delivered to a company's mail server,
while other traffic like web traffic would get delivered to their web server.
A record type very similar to the MX record is the SRV record.
SRV stands for service record,
and it's used to define the location of various specific services.
It serves the exact same purpose as the MX resource record type except for one thing,
while MX is only for mail services,
an SRV record can be defined to return the specifics of many different service types.
For example, SRV records are often used to return the records of services like CalDAV,
which has a calendar and scheduling service.
The text record type is an interesting one.
TXT stands for text and was originally intended to be used only
for associating some descriptive text with a domain name for human consumption.
The idea was that,
you could leave notes or messages that humans could discover
and read to learn more about arbitrary specifics of your network.
But over the years as the Internet and
services that run on it have become more and more complex,
the text record has been increasingly used to convey

additional data intended for other computers to process.
Since the text record has a field that's entirely free form,
clever engineers have figured out ways to use it to communicate data
not originally intended to be communicated by a system like DNS.
It's pretty clever, right? This text record is often used to communicate
configuration preferences about network services that
you've entrusted other organizations to handle for your domain.
For example, it's common for the text record to be used to convey
additional info to an email as a service provider,
which is a company that handles your email delivery for you.
There are lots of other DNS resource record types in common use like
the NS or SOA records which are used to define authority information about DNS zones.
We'll cover DNS zones in more detail in a future video.

Any given domain name has three primary parts, and
they all serve specific purposes.
Let's take the domain name www.google.com, the three part here should be
pretty easy to spot since they're each set off from each other by a period.
They're www, google, and com,
the last part of a domain name is known as the TLD or Top Level Domain.
In this case it's the .com portion of the domain name.
There are only a certain restricted number of defined TLDs available,
although that number has been growing a lot in recent years.
The most common TLDs are ones you're probably already familiar with .com, .net,
.edu and so on.
You've probably also seen some country specific TLDs such as .de for
Germany or .cn for China.
Due to the growth of the Internet,
many of the TLDs originally defined have become very crowded.
So today, a number of vanity TLDs are available,
everything from .museum to .pizza.
Administration and definition of TLDs is handled by a non-profit
organization known as ICANN, or the Internet Corporation for
Assigned Names and Numbers and I can tell you what they do.
ICANN is a sister organization to the IANA, and together they help define and
control both the global IP spaces, along with the global DNS system.
A domain is the name commonly used to refer to the second part of a domain name,
which would be, google in our example.
Domains are used to demarcate where control moves from a TLD name server,
to an authoritative name server.
This is typically under the control of an independent organization, or
someone outside of ICANN.
Domains can be registered and chosen by any individual or
company, but they must all end in one of the predefined TLDs.
While it costs money to officially register a domain with a registrar,
subdomains can be freely chosen and
assigned by anyone who controls such a registered domain.
A registrar is just a company
that has an agreement with ICANN to sell unregistered domain names.
We'll talk more about dealing with registrars in a future module.
Technically you can have lots of subdomain names, for
example host.sub.sub.subdomain.domain.com can be completely valid.
Although you rarely see fully qualified domain names with that many levels.

DNS can technically support up to 127 levels
of domain in total for a single fully qualified domain name.
There are some other restrictions in place for how your domain name can be specified.
Each individual section can only be 63 characters long and
a complete FQDN is limited to a total of 255 characters

An authoritative name server is actually responsible for a specific DNS zone.
DNS zones are a hierarchical concept.
The root name servers we covered earlier are responsible for the root zone.
Each TLD name server is responsible for the zone covering its specific TLD,
and what we refer to as authoritative name servers are
responsible for some even finer-grained zones underneath that.
The root and TLD name servers are actually just authoritative name servers, too.
It's just that the zones that they're authoritative for are special cases.
I should call out that zones don't overlap.
For example, the administrative authority of the TLD name server for
the.com TLD doesn't encompass the google.com domain.
Instead, it ends at the authoritative server responsible for google.com.
The purpose of DNS zones is to allow for easier control over multiple levels of a domain.
As the number of resource records in a single domain increases,
it becomes more of a headache to manage them all.
Network administrators can ease this pain by
splitting up their configurations into multiple zones.
Let's imagine a large company that owns the domain largecompany.com.
This company has offices in Los Angeles,
Paris, and Shanghai, very cosmopolitan.
Let's say each office has around
200 people with their own uniquely named desktop computer.
This would be 600 A records to keep track of if it was all configured as a single zone.
What the company could do, instead,
is split up each office into their own zone.
So now, we could have la.largecompany.com,
pa.largecompany.com, and sh.largecompany.com as subdomains,
each with their own DNS zones.
A total of four authoritative name servers would now be required for the setup,
one for largecompany.com and one for each of the subdomains.
Zones are configured through what are known as zone files,
simple configuration files that declare all resource records for a particular zone.
So a zone file has to contain an SOA,
or a Start of Authority resource record declaration.
This SOA record declares the zone
and the name of the name server that is authoritative for it.
Along with the SOA record,
you'll usually find NS records which indicate
other name servers that may also be responsible for this zone.
For simplicity's sake, we've been referring to server in the
singular when discussing what's responsible for its zone, whether at the root,
TLD, or domain level,
but there are often going to be
multiple physical servers with their own FQDNs and IP addresses involved.
Having multiple servers in place for something as important as DNS is pretty common.
Why? Well, if one server were to have a problem or suffer a hardware failure,
you could always rely on one of the other ones to serve DNS traffic.

Besides SOA and NS records,
you'll also find some or all of
the other resource record types we've already covered, like A,
quad A, and CNAME records along with configurations such
as default TTL values for the records served by this zone.
Just like how subdomains can go many, many layers deep,
zones can be configured to do this too but,
just like with subdomains,
it's rare to see zones deeper than just a few levels.
Sometimes, you'll also see what are known as reverse lookup zone files.
These let DNS resolvers ask for an IP,
and get the FQDN associated with it returned.
These files are the same as zone files except,
instead of A and quad A records,
which resolve names to IPs,
you'll find mostly pointer resource record declarations.
As you might have guessed, a PTR,
or Pointer Record, resolves an IP to a name.

DHCP

Managing hosts on a network can be a daunting and time consuming task.
Every single computer on a modern TCP/IP based network
needs to have at least four things to specifically configured.
An IP Address, the subnet mask for
the local network, a primary gateway, and a name server.
On their own, these four things don't seem like much, but when you have to configure
them on hundreds of machines, it becomes super tedious.
Out of these four things, three are likely the same on just about every node on
the network, the subnet mask, the primary gateway, and DNS server.
But the last item, an IP address,
needs to be different on every single node on the network.
That could require a lot of tricky configuration work, and
this is where DHCP, or Dynamic Host Configuration Protocol, comes into play.
DHCP is an application layer protocol
that automates the configuration process of hosts on a network.
With DHCP, a machine can query a DHCP server when the computer
connects to the network and receive all the network configuration in one go.
Not only does DHCP reduce the administrative overhead of having to
configure lots of network devices on a single network, it also helps address
the problem of having to choose what IP to assign to what machine.
Every computer on a network requires an IP for communications, but
very few of them require an IP that would be commonly known.
For servers or network equipment on your network,
like your gateway router is static and known IP address, it's pretty important.
For example, the devices on a network need to know the IP of their gateway at
all times.
If the local DNS server was malfunctioning, network administrators
would still need a way to connect to some of these devices through their IP.
Without a static IP configured for a DNS server, it
would be hard to connect to it to diagnose any problems if it was malfunctioning.

But for a bunch of client devices like desktops, or laptops, or even mobile phones, it's really only important that they have an IP on the right network. It's much less important exactly which IP that is.
Using DHCP, you can configure a range of IP addresses that's set aside for these client devices.
This ensures that any of these devices can obtain an IP address when they need one, but solves the problem of having to maintain a list of every node on the network and its corresponding IP.
There are a few standard ways that DHCP can operate.
DHCP dynamic allocation is the most common, and it works how we described it just now.
A range of IP addresses is set aside for client devices and one of these IPs is issued to these devices when they request one.
Under a dynamic allocation, the IP of a computer could be different, almost every time it connects to the network.
Automatic allocation is very similar to dynamic allocation, in that a range of IP addresses is set aside for assignment purposes.
The main difference here is that the DHCP server is asked to keep track of which IPs it's assigned to certain devices in the past.
Using this information, the DHCP server will assign the same IP to the same machine each time, if possible.
Finally, there's whats known as fixed allocation.
Fixed allocation requires a manually specified list of MAC address and the corresponding IPs.
When a computer requests an IP, the DHCP server looks for its MAC address in a table, and assigns the IP that corresponds to that MAC address.
If the MAC address isn't found, the DHCP server might fall back to automatic or dynamic allocation, or it might refuse to assign an IP altogether.
This can be used as a security measure to ensure that only devices that have had their MAC address specifically configured at the DHCP server will ever be able to obtain an IP and communicate on the network.

DHCP is an application layer protocol, which means it relies on the transport, network, data link and physical layers to operate.
But you might have noticed that the entire point of DHCP is to help configure the network layer itself.
The process by which a client configured to use DHCP attempts to get network configuration information is known as DHCP discovery.
The DHCP discovery process has four steps.
First, we have the server discovery step.
The DHCP clients sends what's known as a DHCP discover message out onto the network.
Since the machine doesn't have an IP and it doesn't know the IP of the DHCP server, a specially crafted broadcast message is formed instead.
DHCP listens on UDP port 67 and DHCP discovery messages are always sent from UDP port 68.
So the DHCPDISCOVER message is encapsulated in a UDP datagram with a destination port of 67 and a source port of 68.
This is then encapsulated inside of an IP datagram with a destination IP of 255.255.255.255, and a source IP of 0.0.0.0.
This broadcast message would get delivered to every node on the local area network.
And if a DHCP server is present, it would receive this message.

Next, the DHCP server would examine its own configuration and
would make a decision on what, if any, IP address to offer to the client.
This would depend on if it's configured to run with dynamic, automatic or
fixed address allocation.
The response would be sent as a DHCPOFFER message with a destination port of 68,
a source port of 67, a destination broadcast IP of
255.255.255.255, and its actual IP as the source.
Since the DHCP offer is also a broadcast,
it would reach every machine on the network.
The original client would recognize that this message was intended for itself.
This is because the DHCPOFFER has the field that specifies the MAC address
of the client that sent the DHCPDISCOVER message.
The client machine would now process this DHCPOFFER
to see what IP is being offered to it.
Technically, a DHCP client could reject this offer.
It's totally possible for
multiple DHCP servers to be running on the same network, and for a DHCP client
to be configured to only respond to an offer of an IP within a certain range.
But this is rare.
More often, the DHCP client would respond
to the DHCPOFFER message with a DHCPREQUEST message.
This message essentially says, yes,
I would like to have an IP that you offer to me.
Since the IP hasn't been assigned yet,
this is again sent from an IP of 0.0.0.0, and
to the broadcast IP of 255.255.255.255.
Finally, the DHCP server receives the DHCPREQUEST message and
responds with a DHCPACK or DHCP acknowledgement message.
This message is again sent to a broadcast IP of 255.255.255.255,
and with a source IP corresponding to the actual IP of the DHCP server.
Again, the DHCP client would recognize that this message was intended for
itself by inclusion of its MAC address in one of the message fields.
The networking stack on the client computer can now
use the configuration information presented to it by the DHCP server
to set up its own network layer configuration.
At this stage,
the computer that's acting as the DHCP client should have all the information
it needs to operate in a full fledged manner on the network it's connected to.
All of this configuration is known as DHCP lease as it includes an expiration time.
A DHCP lease might last for days or only for a short amount of time.
Once a lease has expired, the DHCP client would need to negotiate a new lease
by performing the entire DHCP discovery process all over again.
A client can also release its lease to the DHCP server,
which it would do when it disconnects from the network.
This would allow the DHCP server to return the IP address
that was assigned to its pool of available IPs.

NAT

At its most basic level NAT,
is a technology that allows a gateway usually a router or a

firewall to rewrite the source IP of an outgoing IP datagram,
while retaining the original IP in order to rewrite it into the response,
to explain this better,
let's look at a simple NAT example.
Let's say we have two networks,
network A consists of the 10.1 1.0/24 address space,
and network B consists of the 192.
1 6 8.1.0/24 address space.
Sitting between these networks is a router that
has an interface on network A with an IP of
10.1.1.1 and an interface on Network B of 192.168. 1.1
Now let's put two computers on these networks.
Computer one is on network A,
and has an IP of 10.1.1.100 and computer two
is on network B and has an IP of 192.168.1.100.
Computer one wants to communicate with a web server on computer two.
So it crafts the appropriate packet at all layers and sends this to its primary gateway.
The router sitting between the two networks,
so far this is a lot like many of our earlier examples,
but in this instance the router is configured to perform NAT for any outbound packets.
Normally a router will inspect the contents of an IP datagram,
decrement the TTL by one,
recalculate the checksum and forward
the rest of the data at the network layer without touching it.
But with NAT the router will also rewrite the source IP address,
which in this instance becomes the routers IP on Network B or 192.
168. 1.1. When the datagram gets to computer two,
it will look like it originated from the router not from computer one.
Now computer two crafts its response and sends it back to the router.
The router knowing that this traffic is actually intended for computer one,
rewrites the destination IP field before forwarding it along.
What NAT is doing in this example is hiding the IP of computer one from computer two.
This is known as IP masquerading.
IP masquerading is an important security concept.
The most basic concept at play here,
is that no one can establish a connection to
your computer if they don't know what IP address it has.
By using NAT in the way we've just described,
we could actually have hundreds of computers on network A,
all of their IPs being translated by the router to its own.
To the outside world,
the entire address space of network A is protected and invisible.
This is known as one to many NAT,
and you'll see it in use on lots of LANs today.

We now have potentially hundreds of responses all directed at
the same IP and the router at this IP
needs to figure out which responses go to which computer.
The simplest way to do this,
is through port preservation.
Port preservation is a technique where the source port chosen by a client,
is the same port used by the router.
Remember that outbound connections choose a source port at random,

from the ephemeral ports or the ports in the range 49,152 through 65, 535.
In the simplest setup,
a router setup to NAT outbound traffic,
will just keep track of what this source port is,
and use that to direct traffic back to the right computer.
Let's imagine a device with an IP of 10.1.1.100.
It wants to establish an outbound connection and the networking stack of
the operating system chooses port 51,300 for this connection.
Once this outbound connection gets to the router,
it performs network address translation and places
its own IP in the source address field of the IP datagram,
but it leaves the source port in the TCP datagram
the same and stores this data internally in a table.
Now, when traffic returns to the router and port 51,300,
it knows that this traffic needs to be forwarded back to the IP 10.1.1.100.
Even with how large the set of ephemeral ports is,
it's still possible for two different computers on
a network to both choose the same source port around the same time.
When this happens, the router normally selects an unused port at random to use instead.
Another important concept about NAT and the transport layer, is port forwarding.
Port forwarding is a technique where
a specific destination ports can be configured to always be delivered to specific nodes.
This technique allows for complete IP masquerading,
while still having services that can respond to incoming traffic.
Let's use our network 10.1.1.0 \24 again to demonstrate this.
Let's say there's a web server configured with an IP of 10.1.1.5.
With port forwarding, no one would even have to know this IP.
Prospective web clients would only have to know about the external IP of the router.
Let's say it's 192.168.1.1.
Any traffic directed at port 80 on 192.168.1.1,
would get automatically forwarded to 10.1.1.5.
Response traffic would have the source IP
rewritten to look like the external IP of the router.
This technique not only allows for IP masquerading,
it also simplifies how external users might interact
with lots of services all run by the same organization.
Let's imagine a company with both a web server and mail server,
both need to be accessible to the outside world
but they run on different servers with different IPs.
Again, let's say the web server has an IP of 10.1.1.5,
and the mail server has an IP of 10.1.1.6.
With port forwarding, traffic for either of these services could be aimed
at the same external IP and therefore the same DNS name,
but it would get delivered to
entirely different internal servers due to their different destination ports.

(diff. btw them is that the first one deals with known specific ports through router. But the other one can choose whatever port to establish connection between 2 computers)

The IANA has been in charge of distributing IP addresses since 1988.
Since that time, the internet has expanded at an incredible rate.

The 4.2 billion possible IPv4 addresses have been predicted to run out for
a long time and they almost have.
For some time now, the IANA has primarily been responsible with assigning address
blocks to the five regional internet registries or RIRs.
The five RIRs are AFRINIC, which serves the continent of Africa,
ARIN which serves the United States, Canada and parts of the Caribbean.
APNIC, which is responsible for
most of Asia, Australia and New Zealand and Pacific Island nations.
LACNIC which covers Central and South America and
any parts of the Caribbean not covered by ARIN.
And finally RIPE, which serves Europe, Russia and
the Middle East and portions of Central Asia.
These five RIRs have been responsible for assigning IP address blocks
to organizations within their geographic areas and most have already run out.
The IANA assigned the last unallocated slash eight network blocks
to various RIRs on February 3rd, 2011.
Then in April 2011, APNIC ran out of addresses.
RIPE was next, in September of 2012.
LACNIC ran out of addresses to assign in June 2014.
And ARIN did the same in September 2015.
Only AFNIC has some IPs left, but those are predicted to be depleted by 2018.
But implementing IPv6 worldwide is going to take some time.
For now, we wanted to continue to grow and we want more people and devices
to connect to it but without IP addresses to assign, a workaround is needed.
Spoiler alert, you already know about the major components of this workaround.
NAT and non-routable address space.
Remember that non-routable address space was defined in RFC1918 and
consists of several different IP ranges that anyone can use.
And unlimited number of networks can use non-routable address space internally
because internet routers won't forward traffic to it.
This means there's never any global collision of IP addresses
when people use those address spaces.
Non-routable address space is largely usable today
because of technologies like NAT.
With NAT,
you can have hundreds even thousands of machines using non-routable address space.
Yet, with just a single public IP,
all those computers can still send traffic to and receive traffic from the internet.
All you need is one single IPv4 address and via NAT,
a router with that IP can represent lots and lots of computers behind it.
It's not a perfect solution, but until IPv6 becomes more globally available,
non-routable address space and NAT will have to do

VPNs and Proxy Services

Virtual Private Networks or VPNs,
are a technology that allows for the extension of a private or local network,
to a host that might not work on that same local network.
VPNs come in many flavors and accomplish lots of different things.
But the most common example of how VPNs are used,
is for employees to access their businesses network when they're not in the office.

VPNs are a tunneling protocol.
Which means, they provision access to something not locally available.
When establishing a VPN connection,
you might also say that a VPN tunnel has been established.
Let's go back to the example of an employee who needs to
access company resources while not in the office.
The employee could use a VPN client to establish a VPN tunnel to their company network.
This would provision their computer with what's known as a virtual interface,
with an IP that matches
the address space of the network that established a VPN connection to.
By sending data out of this virtual interface,
the computer could access internal resources
just like if it was physically connected to the private network.
Most VPNs work by using the payload section of the transport layer to
carry an encrypted payload that actually contains an entire second set of packets.
The network, the transport,
and the application layers of a packet intended to traverse the remote network.
Basically, this payload is carried to the VPNs endpoint,
where all the other layers are stripped away and discarded.
Then, the payload is unencrypted,
leaving the VPN server with the top three layers of a new packet.
This gets encapsulated with the proper data link layer information,
and sent out across the network.
This process is completed in the inverse,
in the opposite direction.
VPNs, usually requires strict authentication procedures in order to ensure that
they can only be connected to by computers and users authorized to do so.
In fact, VPNs were one of
the first technologies where two-factor authentication became common.
Two-factor authentication is a technique where
more than just a username and password are required to authenticate.
Usually, a short-lived numerical token is
generated by the user through a specialized piece of hardware or software.
VPNs can also be used to establish site-to-site connectivity.
Conceptually, there isn't much difference between
how this works compared to a remote employee situation.
It's just that the router,
or sometimes a specialized VPN device on one network,
establishes the VPN tunnel to the router or VPN device on another network.
This way, two physically separated offices might be able to
act as one network and access network resources across the tunnel.
It's important to call out that just like NAT,
VPNs a general technology concept,
not a strictly defined protocol.
There are lots of unique implementations of VPNs.
And the details of how they all work can differ a ton.
The most important takeaway is that VPNs are a technology that use
encrypted tunnels to allow for a remote computer or network,
to act as if it's connected to a network that it's not actually physically connected to.

A proxy service is a server that acts on behalf of a client in order to access another service.
Proxies sit between clients and other servers, providing some additional benefit, anonymity, security, content filtering, increased performance, a couple other things.
Most often, you'll hear the term, proxy, used to refer to web proxies.
As you might guess, these are proxies specifically built for web traffic.
A web proxy can serve lots of purposes.
Many years ago, when most Internet connections were much slower than they are today, lots of organizations used web proxies for increased performance.
Using a web proxy, an organization would direct all web traffic through it, allowing the proxy server itself to actually retrieve the webpage data from the Internet.
It would then cache this data.
This way, if someone else requested the same webpage, it could just return the cached data instead of having to retrieve the fresh copy every time.
This kind of proxy is pretty old and you wont often find them in use today, why?
Well, for one thing, most organizations now have connections fast enough that caching individual webpages doesn't provide much benefit.
Also, the Web has become much more dynamic.
Going to www.twitter.com is going to look different to every person with their own Twitter account, so caching this data wouldn't do much good.
A more common use of a web proxy today might be to prevent someone from accessing sites, like Twitter, entirely.
A company might decide that accessing Twitter during work hours reduces productivity.
By using a web proxy, they can direct all web traffic to it, allow the proxy to inspect what data is being requested, and then allow or deny this request, depending on what site is being accessed.
Another example of a proxy is a reverse proxy.
A reverse proxy is a service that might appear to be a single server to external clients, but actually represents many servers living behind it.
A good example of this is how lots of popular websites are architected today.
Very popular websites, like Twitter, receive so much traffic that there's no way a single web server could possibly handle all of it.
A website that popular might need many, many web servers in order to keep up with processing all incoming requests.
A reverse proxy, in this situation, could act as a single front-end for many web servers living behind it.
From the clients' perspective, it looks like they're all connected to the same server.
But behind the scenes, this reverse proxy server is actually distributing these incoming requests to lots of different physical servers.
Much like the concept of DNS Round Robin, this is a form of load balancing.
Another way that reverse proxies are commonly used by popular websites is to deal with decryption.
More than half of all traffic on the Web is now encrypted, and encrypting and decrypting data is a process that can take a lot of processing power.

Week 5

POTS and Dial-up

A dial-up connection uses POTS for data transfer, and gets its name
because the connection is established by actually dialing a phone number.
If you used dial up, back in the day, this noise might sound familiar to you.
[NOISE] For some of us it was like nails on a chalkboard as
we waited to get connected to the Internet.
Transferring data across a dial-up connection is done through devices called
modems.
Modem stands for modulator demodulator, and
they take data that computers can understand and
turn them into audible wavelengths that can be transmitted over POTS.
After all, the telephone system was developed to transmit voice messages or
sounds from one place to another.
This is conceptually similar to how line coding is used to turn ones and
zeroes into modulating electrical charges across Ethernet cables.
Early modems had very low baud rates.
A baud rate is a measurement of how many
bits could be passed across a phone line in a second.
By the late 1950s, computers could generally only send each other data
across a phone line at about a 110 bits per second.
By the time USENET was being developed,
this rate had increased to around 300 bits per second.
And by the time dial-up access to the Internet became a household commodity in
the early 1990s, this rate had increased to 14.4 kilobits per second.
Improvements continue to be made, but
widespread adoption of broadband technologies,
which we'll discuss in the next lesson, replaced a lot of these improvements.
Dial-up Internet connectivity is pretty rare today but
it hasn't completely gone away.
In some rural areas, it might be the only option still available.
You might never run into a dial-up Internet connection during your IT career.
But it's still important to know that for
several decades this technology represented the main
way computers communicated with each other over long distances.
I'm just glad we don't have to choose between using the phone or
using the Internet anymore.
Next, let's take a deep dive into the world of broadband.

Broadband Connections

In terms of internet connectivity,
it's used to refer to any connectivity technology that isn't dial-up Internet.
Broadband Internet is almost always much faster than
even the fastest dial-up connections and refers to connections that are always on.
We'll deep dive into four of

the most common broadband solutions available today: T-carrier technologies,
digital subscriber lines or DSL,
cable broadband, and fiber connections.

T-Carrier Technologies were first invented by AT&T in order to
provision a system that allowed lots of phone calls to travel across a single cable.
Every individual phone call was made over
individual pairs of copper wire before Transmission System 1,
the first T-Carrier specification called T1 for short.
With the T1 specification,
AT&T invented a way to carry
up to 24 simultaneous phone calls across a single piece of twisted pair copper.
Years later, the same technology was re-purposed for data transfers.
Each of the 24 phone channels was capable of transmitting data at 64 kilobits per second,
making a single T1 line capable of transmitting data at 1.544 megabits per second.
Over the years, the phrase T1 has come to mean
any twisted-pair copper connection capable of speeds of 1.544 megabits per second even,
if it doesn't strictly follow the original Transmission System 1 specification.
Originally, T1 technology was only used to connect
different telecom company sites to each other
and to connect these companies to other telecom companies.
But with the rise of the internet as a useful business tool in the 1990s,
more and more businesses started to pay to have T1 lines
installed at their offices to have faster internet connectivity.
More improvements to the T1 line were made by
developing a way of multiple T1s to act as a single link.
So, T3 line is 28 T1s all multiplexed
achieving a total throughput speed of 44.736 megabits per second.
You'll still find T-Carrier Technologies in use today,
but they've usually been surpassed by other broadband technologies.
For small business offices,
cable broadband or fiber connections are now way more common,
since they're much cheaper to operate.
For inner ISP communications,
different fiber technologies have all replaced older copper-based ones.

The public telephone network was a great option for getting
people connected to the Internet since it already had infrastructure everywhere.
For a long time, dial-up connections were
the main way that people connected to the Internet from home.
But there were certain limitations with trying to transmit
data as what were essentially just audio waves.
As people wanted faster and faster Internet access,
telephone companies began to wonder if they could
use the same infrastructure but in a different way.
The research showed that twisted pair copper used by
modern telephone lines was capable of transmitting way
more data than what was needed for voice-to-voice calls.
By operating at a frequency range that didn't interfere with normal phone calls,
a technology known as digital subscriber line or
DSL was able to send
much more data across the wire than traditional dial-up technologies.

To top it all off, this allowed for normal voice phone calls and
data transfer to occur at the same time on the same line.
Like how dial-up uses modems,
DSL technologies also use their own modems.
But, more accurately, they're known as DSLAMs
or Digital Subscriber Line Access Multiplexers.
Just like dial-up modems,
these devices establish data connections across phone lines,
but unlike dial-up connections, they're usually long-running.
This means that the connection is generally established when the DSLAM is
powered on and isn't torn down until the DSLAM is powered off.
There are lots of different kinds of DSL available,
but they all vary in a pretty minor way.
For a long time,
the two most common types of DSL were ADSL and SDSL.
ADSL stands for Asymmetric Digital Subscriber Line.
ADSL connections feature different speeds for outbound and incoming data.
Generally, this means faster download speeds and slower upload speeds.
Home users rarely need to upload as much data
as they download since home users are mostly just clients.
For example, when you open a web page in a web browser,
the upload or outbound data is pretty small.
You're just asking for a certain web page from the web server.
The download or inbound data tends to be much larger since
it'll contain the entire web page including all images and other media.
For this reason, asymmetric lines often
provide a similar user experience for a typical home user,
but at a lower cost.
SDSL, as you might be able to guess,
stands for Symmetric Digital Subscriber Line.
SDSL technology is basically the same as
ADSL except the upload and download speeds are the same.
At one point, SDSL was mainly used by
businesses that hosted servers that needed to send data to clients.
As the general bandwidth available on the Internet has expanded
and as the cost of operation have come down over the years,
SDSL is now more common for both businesses and home users.
Most SDSL technologies and have an upper cap of
1.544 megabits a second or the same as a T1 line.
Further developments in SDSL technology have yielded things like
HDSL or High Bit-rate Digital Subscriber Lines.
These are DSL technologies that provision speeds above 1.544 megabits per second.
There are lots of other minor variations in DSL technology out in
the wild offering different bandwidth options and operating distances.
These variations can be so numerous and minor,
it's not really practical to try to cover them here.
If you ever need to know more about a specific DSL line,
you should contact the ISP that provides it for more details.

Much like how DSL was developed, cable companies quickly realized that
the coaxial cables generally used by cable television delivery into a person's
home were capable of transmitting much more data than what was required for
TV viewing.

By using frequencies that don't interfere with television broadcast,
cable-based Internet access technologies
were able to deliver high speed Internet access across these same cables.
This is the technology that we refer to when we say cable broadband.
One of the main differences in how cable Internet access works
when compared to other broadband solutions is that cable
is generally what's known as a shared bandwidth technology.
With technologies like DSL or even dial up, the connection from your home or
business goes directly to what's known as a Central Office or CO.
A long time ago, the COs were actually offices staffed with telephone operators
who used a switchboard to manually connect the caller with the callee.
As technology improved, the COs became smaller pieces of automated hardware that
handled these functions for the telephone companies, but the name stayed the same.
Technologies that connect directly to a CO can guarantee a certain amount of
bandwidth available over that connection since it's point to point.
On the flip side of this,
are cable Internet technologies, which employ a shared bandwidth model.
With this model in place, many users share a certain amount of bandwidth until
the transmissions reach the ISP's core network.
This could be anywhere from a single city block
to entire subdivisions in the suburbs.
It just depends on how that area was originally wired for cables.
Today, most cable operators have tried to upgrade their networks to the point that
end users might not always notice the shared bandwidth.
But it's also still common to see cable Internet connections slow down during
periods of heavy use.
Like when lots of people in the same region are using their Internet connection
at the same time.
Cable Internet connections are usually managed by what's known as a cable modem.
This is a device that sits at the edge of a consumer's network and
connects it to the cable modem termination system, or CMTS.
The CMTS is what connects lots of different cable connections
to an ISP's core network.

The core of the internet has long used fiber for its connections,
both due to higher speeds and because fiber allows for
transmission to travel much further without degradation of the signal.
Remember that fiber connections use light for
data transmission instead of electrical currents.
The absolute maximum distance an electrical signal can travel across
a copper cable before it degrades too much and requires a repeater is thousands of feet.
But, certain implementations of fiber connections can travel many,
many miles before a signal degrades.
Producing and laying fiber is a lot more expensive than using copper cables.
So, for a long time,
it was a technology you only saw in use by ISPs
within their core networks or maybe for use within data centers.
But in recent years,
it's become popular to use fiber to deliver data closer and closer to the end user.
Exactly how close to the end user can vary a ton across implementations,
which is why the phrase FTTX was developed.
FTTX stands for fiber to the X,
where the X can be one of many things.

We'll cover a few of these possibilities.
The first term you might hear is FTTN,
which means fiber to the neighborhood.
This means that fiber technologies are used to deliver data to
a single physical cabinet that serves a certain amount of the population.
From this cabinet, twisted pair copper
or coax might be used for the last length of distance.
The next version you might come across is FTTB.
This stands for fiber to the building,
fiber to the business or even a fiber to the basement,
since this is generally where cables to buildings physically enter.
FTTB is a setup where fiber technologies
are used for data delivery to an individual building.
After that, twisted pair copper is typically
used to actually connect those inside of the building.
A third version you might hear is FTTH,
which stands for fiber to the home.
This is used in instances where fiber is actually run to
each individual residents in a neighborhood or apartment building.
FTTH and FTTB may both also be referred to as FTTP,
fiber to the premises.
Instead of a modem,
the demarcation point for fiber technologies is known
as Optical Network Terminator, or ONT.
An ONT converts data from protocols the fiber network can
understand to those that are
more traditional twisted pair copper networks can understand.

WANs

Let's say that you're in charge of the network as
the sole IT support specialist at a small company.
At first, the business only has a few employees with a few computers in a single office.
You decide to use non-routable address space for
the internal IPs because IP addresses are scarce and expensive.
You set up a router and configure it to perform NAT.
You configure a local DNS server and a DHCP server to make network configuration easier.
And of course, for all of this to really work,
you sign a contract with an ISP to deliver a link to
the Internet to this office so your users can access the web.
Now imagine the company grows.
You're using non-routable address space for your internal IPs,
so you have plenty of space to grow there.
Maybe some salespeople need to connect to resources
on the LAN you've set up while they're on the road,
so you configure a VPN server and make
sure the VPN server is accessible via port forwarding.
Now, you can have employees from all over the world connect to the office LAN.
Business is good and the company keeps growing.
The CEO decides that it's time to open a new office in another city across the country.
Suddenly, instead of a handful of sales people

requiring remote access to the resources on your network,
you have an entire second office that needs it.
This is where wide area networks or WAN technologies come into play.
Unlike a LAN or a local area network,
WAN stands for wide area network.
A wide area network acts like a single network but
spans across multiple physical locations.
WAN technologies usually require that you
contract a link across the Internet with your ISP.
This ISP handles sending your data from one side to the other.
So, it could be like all of your computers are in the same physical location.
A typical WAN set up has a few sections.
Imagine one network of computers on one side of
the country and another network of computers on the other.
Each of those networks ends at a demarcation point,
which is where the ISPs network takes over.
The area between each demarcation point and
the ISP's actual core network is called a local loop.
This local loop would be something like a T-carrier line or
a high-speed optical connection to the provider's local regional office.
From there, it would connect out to the ISP's core network and the Internet a large.
WANs work by using a number of different protocols at
the data link layer to transport your data from one site to another.
In fact, these same protocols are what are sometimes at work
at the core of the Internet itself instead of our more familiar Ethernet.
Covering all the details of these protocols is out of the scope of this course,
but in an upcoming lesson,
we'll give you some links to the most popular WAN protocols.

Point to point vpns can also be used to eliminate the need of WANs or WAN protocols.

Wireless Networking

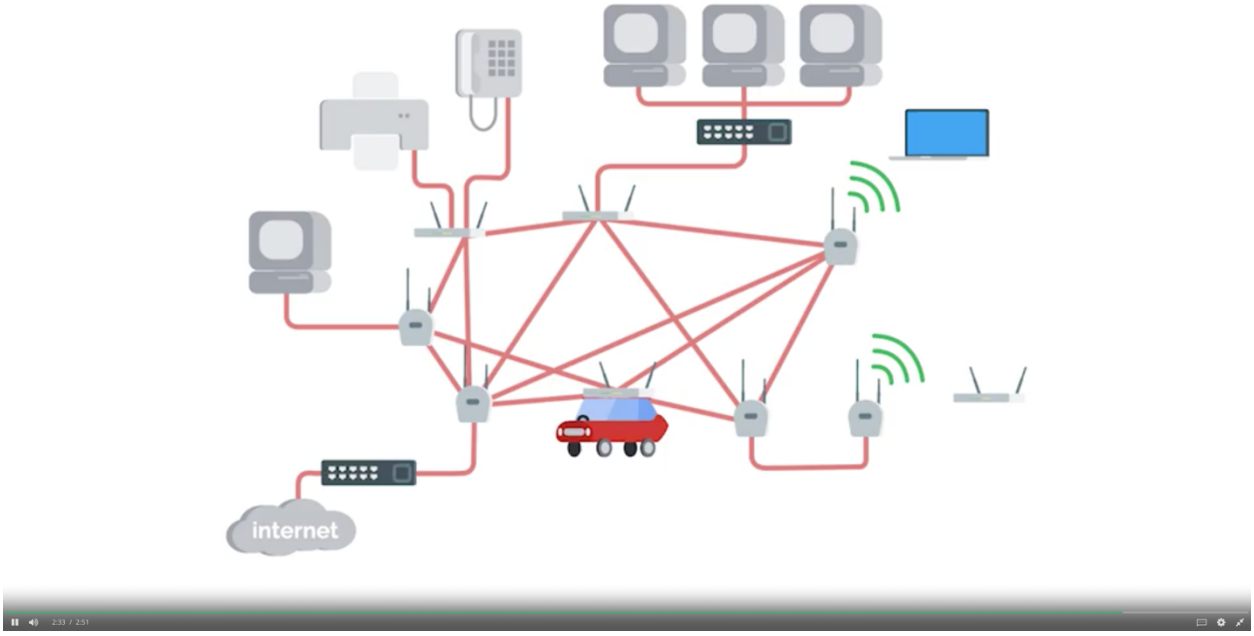
Wireless networking, is exactly what it sounds like.
A way to network without wires.
The most common specifications for how wireless networking devices should communicate,
are defined by the IEEE 802.11 standards.
This set of specifications,
also called the 802.11 family,
make up the set of technologies we call WiFi.
Wireless networking devices communicate with each other through radiowaves.
Different 802.11 standards generally use the same basic protocol,
but might operate at different frequency bands.
A frequency band is a certain section of
the radio spectrum that's been agreed upon to be used for certain communications.
In North America, FM radio transmissions operate between 88 and 108 megahertz.
This specific frequency band is called the FM broadcast band.
WiFi networks operate on a few different frequency bands.

Most commonly, the 2.4 gigahertz and 5 gigahertz bands.
There are lots of 802.11 specifications
including some that exist just experimentally or for testing.
The most common specifications you might run into are 802.11b,
802.11a, 802.11g, 802.11n, and 802.11ac.
We won't go into detail about each one here.
For now, just know that we've listed these in the order they were adopted.
Each newer version of the 802.11 specifications has generally seen some improvement,
whether it's higher access speeds,
or the ability for more devices to use the network simultaneously.
In terms of our networking model,
you should think of 802.11 protocols as defining
how we operate at both the physical and the data link layers.
An 802.11 frame has a number of fields.
The first is called the frame control field.
This field is 16 bits long,
and contains a number of sub-fields that are used to
describe how the frame itself should be processed.
This includes things like what version of the 802.11 was used.
The next field is called a duration field.
It specifies how long the total frame is.
So, the receiver knows how long it should expect to have to listen to the transmission.
After this, are four address fields.
Let's take a moment to talk about why there are four instead of the normal two.
We'll discuss different types of
wireless network architectures in more detail later in this lesson,
but the most common setup includes devices called access points.
A wireless access point is a device that
bridges the wireless and wired portions of a network.
A single wireless network might have
lots of different access points to cover a large area.
Devices on a wireless network will associate with a certain access point.
This is usually the one they're physically closest to.
But, it can also be determined by all sorts of other things
like general signal strength, and wireless interference.
Associations isn't just important for
the wireless device to talk to a specific access point,
it also allows for incoming transmissions to
the wireless device to be sent by the right access point.
There are four address fields,
because there needs to be room to indicate
which wireless access point should be processing the frame.
So, we'd have our normal source address field,
which would represent the MAC address of the sending device.
But, we'd also have the intended destination on the network,
along with a receiving address and a transmitter address.
The receiver address would be
the MAC address of the access point that should receive the frame,
and the transmitter address would be
the MAC address of whatever has just transmitted the frame.
In lots of situations,
the destination and receiver address might be the same.
Usually, the source and transmitter addresses are also the same.

But, depending on exactly how a specific wireless network has been architected,
this won't always be the case.
Sometimes, wireless access points will relay these frames from one another.
Since all addresses in an 802.11 frame are Mac addresses,
each of those four fields is 6 bytes long.
In between the third and fourth address fields,
you'll find the sequence control field.
The sequence control field is 16 bits long and mainly
contains a sequence number used to keep track of ordering the frames.
After this is the data payload section
which has all of the data of the protocols further up the stack.
Finally, we have a frame check sequence field
which contains a checksum used for a cyclical redundancy check.
Just like how ethernet does it.

There are a few main ways that a wireless network can be configured.
There are Ad-hoc networks where nodes all speak directly to each other.
There are wireless LANS or WLANS where
one or more access points act as a bridge between a wireless and a wired network.
And there are Mesh networks which are kind of a hybrid of the two.
Ad-hoc networks are the simplest of the three.
In an ad-hoc network,
there isn't really any supporting network infrastructure.
Every device involved with the network communicates with every other device within range,
and all nodes help pass along messages.
Even though they are most simple,
ad-hoc networks aren't the most common type of wireless network,
but they do have some practical applications.
Some smartphones can establish ad-hoc networks with
other smartphones in the area so that people can exchange photos,
video, or contact information.

You'll also sometimes see ad-hoc networks used in industrial or warehouse settings,
where individual pieces of equipment might need to
communicate with each other but not with anything else.
Finally, ad-hoc networks can be powerful tools during disaster situations.
If a natural disaster like an earthquake or
hurricane knocks out all of the existing infrastructure in an area,
disaster relief professionals can use an ad-hoc network
to communicate with each other while they perform search and rescue efforts.
The most common type of wireless network you'll run into in
the business world is a wireless LAN or WLAN.
A wireless LAN consist of one or more access points
which act as bridges between the wireless and wired networks.
The wired network operates as a normal LAN,
like the types we've already discussed.
The wired LAN contains the outbound internet link.
In order to access resources outside of the WLAN,
wireless devices would communicate with access points.
They then forward traffic along to the Gateway router,
where everything proceeds like normal.
Finally, we have what's known as Mesh networks.
Mesh networks are kind of like ad-hoc networks,
since lots of the devices communicate with each other wirelessly forming
a mesh if you were to draw lines for all the links between all the nodes.
Most Mesh networks you'll run into are made up of only wireless access points,
and will still be connected to a wired network.
This kind of network lets you deploy
more access points to the Mesh without having to run a cable to each of them.
With this kind of setup,
you can really increase the performance and range of a wireless network.



//mesh network

The concept of channels is one of
the most important things to understand about wireless networking.
Channels are individual, smaller sections of
the overall frequency band used by a wireless network.
Channels are super important because they help address
a very old networking concern, collision domains.
You might remember that a collision domain is
any one network segment where one computer can interrupt another.
Communications that overlap each other can't be properly understood by the receiving end.
So when two or more transmissions occur at the same time,
also called a collision,
all devices in question have to stop their transmissions.
They wait a random amount of time and try again when things quiet down.
This really slows things down.
The problem caused by collision domains has been mostly
reduced on wired networks through devices called switches.
Switches remember which computers live on which physical interfaces.
So traffic is only sent to the node It's intended for.
Wireless networking doesn't have cables,
so there aren't physical interfaces for a wireless device to connect to.
That means, we can have something that works like a wireless switch.
Wireless devices are doomed to talk over each other.
Channels help fix this problem to a certain extent.
When we were talking about the concept the frequency bands,
we mentioned that FM radio in North America operates
between 80 megahertz and 108 megahertz.
But when we discuss the frequency bands we use by Wi-Fi,
we just mentioned 2.4 Gigahertz and five Gigahertz.
This is because that's really just shorthand
for where these frequency bands actually begin.
For wireless networks that operate on a 2.4 Gigahertz band,
what we really mean is that they operate on
roughly the band from 2.4 Gigahertz to 2.5 Gigahertz.
Between these two frequencies are a number of channels,
each with a width of a certain megahertz.
Since different countries and regions have
different regulatory committees for what radio frequencies might be used for what,
exactly how many channels are available for use depends on where in the world you are.
For example, dealing with an 802.11b network,
channel one operates at 2412 megahertz,
but since the channel width is 22 megahertz,
the signal really lives on the frequencies between 2401 megahertz and 2423 megahertz.
This is because radio waves are imprecise things.
So, you need some buffer around
what exact frequencies a transmission might actually arrive on.
Some channels overlap but some are
far enough apart so they won't interfere with each other at all.
Let's look again at 802.11b network running on a 2.4 Gigahertz band,
because it's really the simplest and the concepts
translate to all other 802.11 specifications.
With a channel width of 22 megahertz,
channel one with its midpoint at 2412 megahertz,
is always completely isolated from channel six with its midpoint at 2437 megahertz.

For an 802.11b network,
this means that channels one and six and 11 are the only ones that never overlap at all.
That's not all that matters, though.
Today, most wireless networking equipment is built
to auto sense what channels are most congested.
Some access points will only perform this analysis when they start up,
others will dynamically change their channel as needed.
Between those two scenarios and manually specified channels,
you can still run into situations where you experience heavy channel congestion.
This is especially true in
dense urban areas with lots of wireless networks in close proximity.
So, why is this important in the world of I.T.
support? Well, understanding how these channels overlap for all of the 802.11
specifications is a way you can help troubleshoot
bad wireless connectivity problems or slowdowns in the network.
You want to avoid collision domains wherever you can.
I should call out that it's not important to
memorize all of the individual numbers we've talked about.
The point is to understand how collision domains
are a necessary problem with all wireless networks,
and how you can use your knowledge in
this space to optimize wireless network deployments.
You want to make sure that both your own access points and
those of neighboring businesses overlap channels as little as possible.

When you're sending data over a wired link,
your communication has a certain amount of inherent privacy.
The only devices that really know what data is being
transmitted are the two nodes on either end of the link.
Someone or some device that happens to be in close proximity can't just read the data.
With wireless networking, this isn't really the case,
since there aren't cables,
just radio transmissions being broadcast through the air,
anyone within range could hypothetically intercept any transmissions,
whether they were intended for them or not.
To solve this problem, WEP was invented.
WEP stands for Wired Equivalent Privacy,
and it's an encryption technology that provides a very low level of privacy.
Actually, it's really right there in the name, wired equivalent privacy.
Using WEP protects your data a little but it should really only
be seen as being as safe as sending unencrypted data over a wired connection.
The WEP standard is a really weak encryption algorithm.
It doesn't take very long for a bad actor to be
able to break through this encryption and read your data.
You'll learn more about key lengths and encryption in a future course.
But for now, it's important to know that the number of
bits in an encryption key corresponds to how secure it is,
the more bits in a key the longer it takes for someone to crack the encryption.
WEP only uses 40 bits for its encryption keys and with the speed of modern computers,
this can usually be cracked in just a few minutes.
WEP was quickly replaced in most places with WPA or Wi-Fi Protected Access.
WPA, by default, uses a 128-bit key,
making it a whole lot more difficult to crack than WEP.

Today, the most commonly used encryption algorithm for wireless networks is WPA2,
an update to the original WPA.
WPA2 uses a 256-bit key make it even harder to crack.
Another common way to help secure wireless networks is through MAC filtering.
With MAC filtering, you configure your access points to only allow for
connections from a specific set of MAC addresses belonging to devices you trust.
This doesn't do anything more to help encrypt
wireless traffic being sent through the air,
but it does provide an additional barrier preventing
unauthorized devices from connecting to the wireless network itself.

Another super popular form of wireless networking is cellular networking,
also called mobile networking.
Cellular networks are now common all over the world.
In some places, using a cellular network for
Internet access is the most common way of connecting.
At a high level, cellular networks have a lot in common with the 802.11
networks we've already talked about.
Just like there are many different 802.11 specifications,
there are lots of different cellular specifications.
Just like Wi-Fi, cellular networking operates over radio waves, and there
are specific frequency bands specifically reserved for cellular transmissions.
One of the biggest differences is that these frequencies can travel over longer
distances more easily, usually over many kilometers or miles.
Cellular networks are built around the concept of cells.
Each cell is assigned a specific frequency band for use.
Neighboring cells are set up to use bands that don't overlap, just like how we
discussed the optimal setup for a W Lan with multiple access points.
In fact, the cell towers that broadcast and receive cellular transmissions can be
thought of like access points, just with a much larger range.
Lots of devices today use cellular networks for communication.
And not just phones, also tablets and some laptops also have cellular antennas.
It's become more and
more common for high-end automobiles to have built-in cellular access, too.

Week 6

Many of the protocols and devices we've covered have built-in
functionalities to help protect against some of these issues.
These functionalities are known as error detection and error recovery.
Error-detection, is the ability for
a protocol or program to determine that something went wrong.
Error-recovery is the ability for a protocol or program to attempt to fix it.
For example, you might remember that cyclical redundancy checks are used by
multiple layers to make sure that the correct data was received by the receiving end.
If a CRC value doesn't match the data payload,
the data is discarded.
At that point, the transport layer will decide if the data needs to be reset.
But, even with all of these safeguards in place,
errors still pop up.
Misconfigurations occur, hardware breaks down and system incompatibilities come to light

Verifying Connectivity

When network problems come up,
the most common issue you'll run into is
the inability to establish a connection to something.
It could be a server you can't reach at all or a website that isn't loading.
Maybe you can only reach your resource on
your LAN and can't connect to anything on the internet.
Whatever the problem is,
being able to diagnose
connectivity issues is an important part of network troubleshooting.
By the end of this lesson,
you'll be able to use a number of
important troubleshooting tools to help resolve these issues.
When a network error occurs,
the device that detects it needs some way to
communicate this to the source of the problematic traffic.
It could be that a router doesn't know how to route to
a destination or that a certain port isn't reachable.
It could even be that the TTL of an IP datagram
expired and no further router hops will be attempted.
For all of these situations and more,
ICMP or internet control message protocol is used to communicate these issues.
ICMP is mainly used by router or remote hosts to
communicate while transmission has failed back to the origin of the transmission.
The makeup of an ICMP packet is pretty simple.
It has a header with a few fields and a data section that's used
by host to figure out which of their transmissions generated the error.
The first field is the type field,
eight bits long which specifies what type of message is being delivered.
Some examples are destination unreachable or time exceeded.
Immediately after this is the code field which
indicates a more specific reason for the message than just the type.
For example, of the destination unreachable type,
there are individual codes for things like destination network
unreachable and destination port unreachable.
After this is a 16 bit checksum
that works like every other checksum field we've covered so far.
Next up is a 32 bit field with an uninspired name, Rest of header.
You think they could come up with something a bit more
interesting but I can't really think of anything good.
So, who am I to judge?
Anyway, this field is optionally used
by some of the specific types and codes to send more data.
After this is the data payload for an ICMP packet.
The payload for an ICMP packet exists entirely so that
the recipient of the message knows which of
their transmissions caused the error being reported.
It contains the entire IP header and
the first eight bytes of the data payload section of the offending packet.

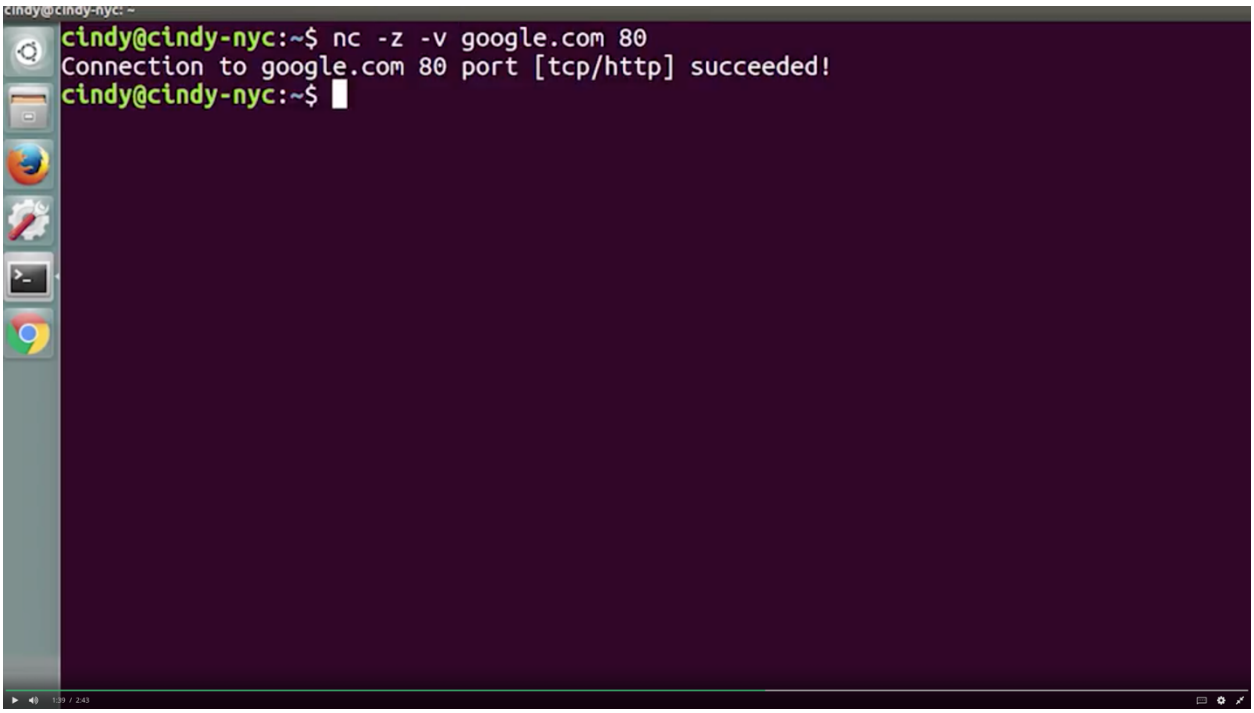
ICMP wasn't really developed for humans to interact with.
The point is so that these sorts of
error messages can be delivered between networked computers automatically.
But, there's also a specific tool and
two message types that are very useful to human operators.
This tool is called ping.
Some version of it exist on just about every operating system,
and has for a very long time.
Ping is a super simple program and
the basics are the same no matter which operating system you're using.
Ping lets you send a special type of ICMP message called an Echo Request.
An ICMP echo request essentially just ask the destination,
"Hey, are you there?"
If the destination is up and running and able to communicate on the network,
it will send back an ICMP echo reply message type.
You can invoke the ping command from the command line of any modern operating system.
In its most basic use,
you just type ping and a destination IP or a fully qualified domain name.
If you don't know how to use a command line in an operating system,
don't worry, you will soon.
We'll cover that in another course.
Output of the ping command is very
similar across each of the different operating systems.
Every line of output will generally display the address sending the ICMP echo reply,
and how long it took for the round trip communications.
It will also have the TTL remaining and how large the ICMP message is in bytes.
Once the command ends,
there will also be some statistics displayed
like percentage of packets transmitted and received,
the average round trip time,
and a couple of other things like that.
On Linux and Mac OS,
the ping command will run until it's
interrupted by an end user sending an interrupt event.
They do this by pressing the control key and the C key at the same time.
On Windows, ping defaults to only sending four echo requests.
In all environments, ping supports a number of command line flags
that let you change its behavior like the number of echo requests to send,
how large they should be,
and how quickly they should be sent.
Check out the documentation for your operating system to learn a little bit more.

With ping, you now have a way to
determine if you can reach a certain computer from another one.
You can also understand the general quality of the connection.
But communications across networks,
especially across the Internet usually,
cross lots of intermediary nodes.
Sometimes, you need a way to determine where in
the long chain of router hops the problems actually are.
Traceroute to the rescue.
Traceroute is an awesome utility that lets you discover the paths between two nodes,
and gives you information about each hop along the way.

The way traceroute works,
is through a clever manipulation technique of the TTL field at the IP level.
We learned earlier that the TTL field is decremented by one,
by every router that forwards the packet.
When the TTL field reaches zero,
the packet is discarded and
an ICMP Time Exceeded message is sent back to the originating host.
Traceroute uses the TTL field by first setting it to one for the first packet,
then two for the second,
three for the third and so on.
By doing this clever little action,
traceroute makes sure that
the very first packet sent will be discarded by the first router hop.
This results in an ICMP Time Exceeded message,
the second packet will make it to the second router,
the third will make it to the third, and so on.
This continues until the packet finally makes it all the way to its destination.
For each hop, traceroute will send three identical packets.
Just like with ping, the output of a traceroute command is pretty simple.
On each line, you'll see the number of
the hop and the round trip time for all three packets.
You will also see the IP of the device at each hop,
and a host name if traceroute can resolve one.
On Linux and MacOS,
traceroute sends UDP packets to very high port numbers.
On Windows, the command has a shortened name tracert,
and defaults to using ICMP echo request.
On all platforms, traceroute has
more options than can be specified using command line flags.
Two more tools that are similar to traceroute are mtr on Linux and
MacOS and pathping on Windows.
These two tools act as long running traceroutes.
So you can better see how things change over a period of time.
Mtr works in real time and will continually update
its output with all the current aggregate data about the traceroute.
You can compare this with pathping,
which runs for 50 seconds and then displays the final aggregate data all at once.

We've covered a bunch of ways to test connectivity between machines at
the network layer.
But sometimes, you need to know if things are working at the transport layer.
For this, there are two super powerful tools at your disposal.
Netcat on Linux and Mac OS and Test-NetConnection on Windows.
The Netcat tool can be run through the command nc, and
has two mandatory arguments, a host and a port.
Running nc google.com 80 would
try to establish a connection on port 80 to google.com.
If the connection fails, the command will exit.
If it succeeds, you'll see a blinking cursor, waiting for more input.
This is a way for you to actually send application layered data to the listening
service from your own keyboard.
If you're really only curious about the status of a report,
you can issue the command, with a -Z flag, which stands for Zero Input/Output Mode.

A -V flag, which stands for Verbose, is also useful in this scenario.
This makes the commands output useful to human eyes
as opposed to non-verbose output, which is best for usage in scripts.
Side note, verbose basically means talking too much.
So, while I bet you want to throw up a flag on me and my jabbering,
we still have lots to get through.
Okay, so by issuing the Netcat command with the -Z and
-V flags, the command's output will simply tell you
if a connection to the port in question is possible or not.
On Windows,
Test-NetConnection is a command with some of the similar functionality.
If you run Test-NetConnection with only a host specified,
it will default to using an ICMP echo request, much like the program ping.
But, it will display way more data,
including the data link layer protocol being used.
When you issue Test-NetConnection with the -port flag,
you can ask it to test connectivity to a specific port.
It's important to call out that both Netcat and Test-NetConnection
are way more powerful than the brief port connectivity examples we've covered here.
In fact, they're such complex tools that covering all of their
functionality would be too much for one video.
You should read up about all of the other things these super powerful tools can do.



//To perform commands using netcat
https://en.wikipedia.org/wiki/Netcat
//To perform commands using Test-NetConnection

https://docs.microsoft.com/en-us/powershell/module/nettcpip/test-netconnection?view=win10-ps

Digging into DNS

Name resolution is an important part of how the Internet works.
Most of the time, your operating system handles all look ups for you.
But as an IT support specialist, sometimes it can be useful to run these queries yourself, so you can see exactly what's happening behind the scenes.
Luckily, there are lots of different command line tools out there to help you with this.
The most common tool is known as nslookup.
And it's available on all three of the operating systems we've been discussing, Linux, Mac, and Windows.
A basic use of nslookup is pretty simple.
You execute the nslookup command with the host name following it.
And the output displays what server was used to perform the request and the resolution result.
Let's say you needed to know the IP address for a twitter.com.
You would just enter nslookup twitter.com and the A record would be returned.
Nslookup is way more powerful than just that.
It includes an interactive mode that lets you set additional options and run lots of queries in a row.
To start an interactive nslookup session,
you just enter nslookup, without any hostname following it.
You should see an angle bracket acting as your prompt.
From interactive mode, you can make lots of requests in a row.
You can also perform some extra configuration to help with more in-depth trouble shooting.
While in interactive mode, if you type server,
then an address, all the following name resolution queries will be attempted to be made using that server instead of the default name server.
You can also enter set type= followed by a resource record type.
By default, nslookup will return A records.
But this lets you explicitly ask for AAAA or MX or
even text records associated with the host.
If you really want to see exactly what's going on, you can enter set debug.
This will allow the tool to display the full response packets,
including any intermediary requests and all of their contents.
Warning, this is a lot of data and
can contain details like the TTL left, if it's a cached response,
all the way to the serial number of the zone file the request was made against.

The Cloud

You've probably been hearing people talk about the cloud more and more.
There are public clouds and private clouds and
hybrid clouds and rain clouds but not really relevant here.
There are cloud clients and cloud storage and cloud servers too.

You might hear the cloud mentioned in newspaper headlines and TV advertisements.
The cloud is the future, so we're told,
and IT support specialists really need to keep
up on the latest innovations in tech in order to support them.
But what exactly is the cloud?
The truth is the cloud isn't
a single technology or invention or anything tangible at all.
It's just a concept,
and to throw in another cloud joke,
a pretty nebulous one at that.
The fact that the term 'the cloud' has been
applied to something so difficult to define is pretty fitting.
Basically, cloud computing is a technological approach where computing resources
are provisioned in a shareable way so
that lots of users get what they need when they need it.
It's an approach that leans heavily on the idea that companies
provide services for each other using these shared resources.
At the heart of cloud computing is a technology known as hardware virtualization.
Hardware virtualization is a core concept of how cloud computing technologies work.
It allows the concept of a physical machine and
a logical machine to be abstracted away from each other.
With virtualization, a single physical machine called a
host could run many individual virtual instances called guests.
An operating system expects to be able to
communicate with the underlying hardware in certain ways.
Hardware virtualization platforms employ what's called a hypervisor.
A hypervisor is a piece of software that runs and manages virtual machines while also
offering these guests a virtual operating platform
that's indistinguishable from actual hardware.
With virtualization, a single physical computer
can act as the host for many independent virtual instances.
They each run their own independent operating system and,
in many ways, are indistinguishable from
the same operating systems running on physical hardware.
The cloud takes this concept one step further.
If you build a huge cluster of interconnected machines
that can all function as hosts for lots of virtual guests,
you've got a system that lets you share resources among all of those instances.
Let's try explaining this in a more practical way.
Let's say you have the need for four servers.
First, you need an email server.
You've carefully analyzed things and expect this machine
will need eight gigs of RAM to function properly.
Next, you need a name server.
The name server barely needs any resources
since it doesn't have to perform anything really computational.
But, you can't run it on the same physical machine as
your email server since your email server needs to run on Windows,
and your name server needs to run on Linux.
Now, the smallest server configuration
your hardware vendor sells is a machine with eight gigabytes of RAM.
So you have to buy another one with those specifications.
Finally, you have a financial database.

This database is normally pretty quiet
and doesn't need too many resources during normal operations.
But for your end of month billing processes to complete in a timely manner,
you determine the machine would need 32 gigabytes of RAM.
It has to run on a special version of Linux designed just for
the database so the name server can also run on this machine.
So you order a server with that much RAM and then a
second with the same specifications to act as a backup.
In order to run your business this way,
you have to purchase four machines with a grand total of 80 gigabytes of RAM.
That seems pretty outrageous since it's likely that
only 40 gigabytes of this total RAM will ever be used at one time.
Most of the month you're using much less.
That's a lot of money spent on resources you're either never going to use or rarely use.
So let's forget about that model.
Instead, let's imagine a huge collection of
interconnected servers that can host virtualized servers.
These virtual instances running on this collection of
servers can be given access to the underlying RAM as they need it.
Under this model, the company that runs the collection of servers can charge you to host
virtual instances of your servers instead of you buying the four physical machines.
And it could cost much less than what you'd spend on the four physical servers.
The benefits of the cloud are obvious.
But let's take it a step further.
The cloud computing company that can host
your virtualized instances also offer dozens of other services.
So instead of worrying about setting up your own backup solution,
you can just employ theirs.
It's easy. And if you need a load balancer,
you can just use their solution.
Plus, if any underlying hardware breaks,
they just move your virtual instance to another machine without you even noticing.
To top it all off, since these are all virtual servers and services,
you don't have to wait for the physical hardware you ordered to show up.
You just need to click a few buttons in a web browser.
That's a pretty good deal.
In our analogy, we used an example of what a public cloud is,
a large cluster of machines run by another company.
A private cloud takes the same concepts, but instead,
it's entirely used by a single large corporation
and generally physically hosted on its own premises.
Another term you might run into,
a hybrid cloud, isn't really a separate concept.
It's just a term used to describe situations where companies might run things like
their most sensitive proprietary technologies on
a private cloud while entrusting their less sensitive servers to a public cloud.
Those are the basics of what the cloud is.
It's a new model in computing where large clusters of
machines let us use the total resources available in a better way.
The cloud lets you provision a new server in a matter of moments
and leverage lots of existing services instead of having to build your own.
To sum up, its blue skies ahead for anyone using their cloud.
Sorry, I couldn't resist.

In our last video, we gave you a basic definition of what cloud computing is,
but the term has really come to mean so much more than just hosting virtual machines.
Another term that's been used more and more with
the rise of cloud computing is X as a service.
Here, the X can stand for lots of different things.
The way we've described the cloud so far would probably best be
defined as infrastructure as a service or IaaS.
The idea behind infrastructure as a service is that you
shouldn't have to worry about building your own network or your own servers.
You just pay someone else to provide you with that service.
Recently, we've seen the definition of
the cloud expand well beyond infrastructure as a service.
The most common of these are platform as a service,
or PaaS, and software as a service, or SaaS.
Platform as a service is a subset of
cloud computing where a platform is provided for customers to run their services.
This basically means that an execution engine
is provided for whatever software someone wants to run.
A web developer writing a new application doesn't really
need an entire server complete with a complex file system,
dedicated resources, and all these other things.
It doesn't matter if this server is virtual or not.
They really just need an environment that their web app can run in.
That is what platform as a service provides.
Software as a service takes this one step further.
Infrastructure as a service abstracts away the physical infrastructure you
need and platform as a service abstracts away the server instances you need.
Software as a service is essentially a way of licensing the use of
software to others while keeping that software centrally hosted and managed.
Software as a service has become really popular for certain things.
A great example is e-mail.
Offerings like Gmail for Business from Google or
Office 365 Outlook from Microsoft are really good examples of software as a service.
Using one of those services means you're trusting Google or
Microsoft to handle just about everything about your email service.
Software as a service is a model that's gaining a ton of traction.
Web browsers have become so feature packed that lots of things that required
standalone software in the past can now run well inside of a browser.
And if you can run something in a browser,
it's a prime candidate for SaaS.
Today, you can find everything from word processors to graphic design programs to
human resource management solutions offered under a subscription based SaaS model.
More and more, the point of a business' network is just to provide
an internet connection to access different software or data in the cloud.

Another popular way to use cloud technologies, is cloud storage.
In a cloud storage system,
a customer contracts a cloud storage provider
to keep their data secure, accessible, and available.
This data could be anything from individual documents to large database backups.
There are lots of benefits of cloud storage over a traditional storage mechanism.
Without cloud storage, there's the general headache of managing a storage array.
Hard drives are one of

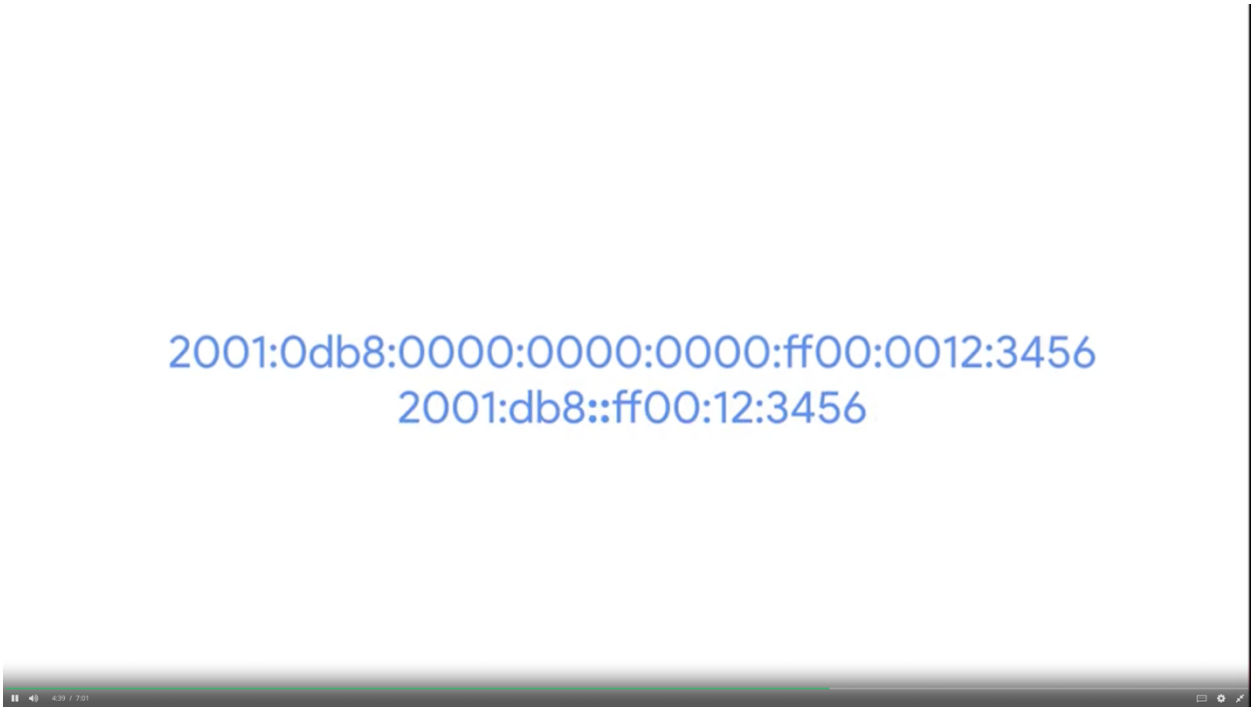
the most frequent components that may experience a malfunction in a computer system.
That means that you'd have to carefully monitor
the devices being used for storage and replace parts when needed.
By using a cloud storage solution,
it's up to the provider to keep the underlying physical hardware running.
Also, cloud storage providers usually operate in lots of different geographic regions.
This let's you easily duplicate your data across multiple sites.
Many of these providers are even global in scale which lets
you make your data more readily available for users all over the world.
This also provides protection against data loss,
since if one region of storage experience has problems,
you can probably still access your data in a different region.
Cloud storage solutions also grow with you.
Typically, you'll pay for exactly how much storage you're using
instead of having a fixed amount like you would with local storage.
While this doesn't always mean that cloud storage is necessarily a cheaper option,
it does mean that you can better manage what your expenses for storage actually are.
Not only is cloud storage useful for replacing large scale local storage arrays,
it's also a good solution for backing up smaller bits of data.
Your smartphone might automatically upload
every picture you take to a cloud storage solution.
If your phone dies, you lose it,
or accidentally delete pictures,
they're still there waiting for you in the cloud.
That way, you'll never lose those precious photos of your pooch, Taco.
Don't worry, Taco, all 2,000 of those photos of you are fully protected.
While I look at some more pictures of Taco,
it's time for a quick

IPv6

A loopback address is a special **IP address**, 127.0. 0.1, reserved by InterNIC for use in testing network cards. This **IP address** corresponds to the software loopback interface of the network card, which does not have hardware associated with it, and does not require a physical connection to a network.

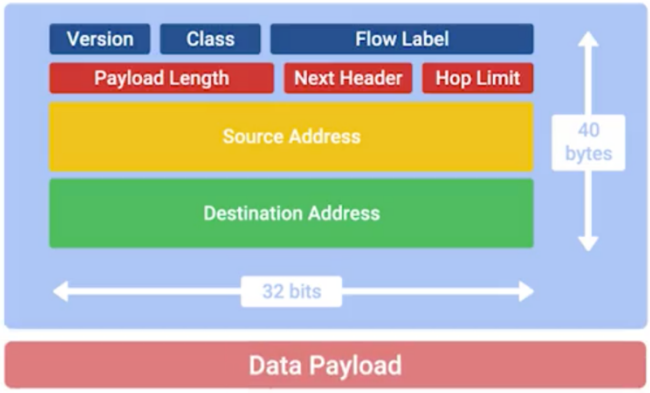
Just like how an IPv4 address is really just a 32-bit binary number,
IPv6 addresses are really just 128-bit binary numbers.
IPv4 addresses are written out in four octets of decimal numbers
just to make them a little more readable for humans.
But trying to do the same for an IPv6 address just wouldn't work.
Instead, IPv6 addresses are usually written out as 8 groups of 16 bits each.
Each one of these groups is further made up of four hexadecimal numbers.
Every single IPv6 address that begins with 2001:0db8 has been reserved for
documentation and education, or for books and courses just like this one.
That's over 18 quintillion addresses,
much larger than the entire IPv4 address space, reserved just for this purpose.
There are two rules when it comes to shortening an IPv6 address.
The first is that you can remove any leading zeros from a group.
The second is that any number of consecutive
groups composed of just zeros can be replaced with two colons.

I should call out that this can only happen once for any specific address.
Otherwise, you couldn't know exactly how many zeros were replaced
by the double colons.
For this IP, we could apply the first rule and
remove all leading zeros from each group.
This would leave us with this.
Once we apply the second rule, which is to replace consecutive sections containing
just zeros with two colons, we'll end up with this.
This still isn't as readable as an IPv4 address, but
it's a good system that helps reduce the length a little bit.
We can see this approach taken to the extreme with IPv6 loopback address.
You might remember that with IPv4, this address is 127.0.0.1.
With IPv6, the loopback address is 31 0s with a 1 at the end,
which can be condensed all the way down to just ::1.
The IPv6 address space has several other reserved address ranges besides just
the one reserved for documentation purposes or the loopback address.
For example, any address that begins with FF00:: is used for
multicast, which is a way of addressing groups of hosts all at once.
It's also good to know that addresses beginning with
FE80:: are used for link-local unicast.
Link-local unicast addresses allow for local network segment communications and
are configured based upon a host's MAC address.
The link-local address are used by an IPv6 host
to receive their network configuration, which is a lot like how DHCP works.
The host's MAC address is run through an algorithm
to turn it from a 48-bit number into a unique 64-bit number.
It's then inserted into the address's host ID.
The IPv6 address space is so huge, there was never any need to think
about splitting it up into address classes like we use to do with IPv4.
From the very beginning,
an IPv6 address had a very simple line between network ID and host ID.
The first 64 bits of any IPv6 address is the network ID, and
the second 64 bits of any IPv6 address is the host ID.
This means that any given IPv6 network has space for over 9 quintillion hosts.
Still, sometimes network engineers might want to split up their network for
administrative purposes.
IPv6 subnetting uses the same CIDR notation that you're
already familiar with.
This is used to define a subnet mask against the network ID portion
of an IPv6 address.



One of the most elegant improvements was made to the IPv6 header,
which is much simpler than the IPv4 one.
The first field in an IPv6 header is the version field.
This is a 4-bit field that defines what version of IP is in use.
You might remember that an IPv4 header begins with this exact same field.
The next field is called the traffic class field.
This is an 8-bit field that defines the type of traffic contained within the IP
datagram and allows for

different classes of traffic to receive different priorities.
The next field is the flow label field.
This is a 20-bit field that's used in conjunction with the traffic class
field for routers to make decisions about the quality of service level for
a specific datagram.
Next you have the payload length field.
This is a 16-bit field that defines how long the data payload section
of the datagram is.
Then you have the next header field.
This is a unique concept to IPv6, and needs a little extra explanation.
IPv6 addresses are four times as long as IPv4 addresses.
That means they have more ones and zeros,
which means that they take longer to transmit across a link.
To help reduce the problems with additional data that IPv6 addresses impose
on the network, the IPv6 header was built to be as short as possible.
One way to do that is to take all of the optional fields and
abstract them away from the IPv6 header itself.
The next header field defines what kind of header is immediately after this
current one.
These additional headers are optional, so they're not required for
a complete IPv6 datagram.
Each of these additional optional headers contain a next header field and allow for
a chain of headers to be formed if there's a lot of optional configuration.
Next we have what's called the hop limit field.
This is an 8-bit field that's identical in purpose to the TTL field in
an IPv4 header.
Finally, we have the source and
destination address fields, which are each a 128 bits.
If the next header field specified another header, it would follow at this time.
If not, a data payload the same length as specified in the payload
length field would follow.



It's just not possible for the entire Internet and
all connected networks to switch to IPv6 all at once.
There would be way too much coordination at play.
Too many old devices that might not even know how to speak IPv6 at all,
still requiring connections.
So the only way IPv6 will ever take hold is to develop a way for
IPv6 and IPv4 traffic to coexist at the same time.
This would let individual organizations make the transition when they can.
One example of how this can work is with what's known as IPv4 mapped address space.
The IPv6 specifications have set aside a number of addresses
that can be directly correlated to an IPv4 address.
Any IPv6 address that begins with 80 zeros, and is then followed
by 16 ones is understood to be part of the IPv4 mapped address space.
The remaining 32 bits of the IPv6 address is
just the same 32 bits of the IPv4 address it's meant to represent.
This gives us a way for IPv4 traffic to travel over an IPv6 network.
But probably more important is for
IPv6 traffic to have a way to travel over IPv4 networks.
It's easier for an individual organization to make the move to IPv6 than it is for
the networks at the core of the Internet to.
So while IPv6 adoption becomes more widespread,
it'll need a way to travel over the old IPv4 remnants of the Internet backbone.
The primary way this is achieve today is through IPv6 tunnels.
IPv6 tunnels are conceptually pretty simple.
They consist of IPv6 tunnels servers on either end of a connection.
These IPv6 tunnel servers take incoming IPv6 traffic and
encapsulate it within traditional IPv4 datagrams.
This is then delivered across the IPv4 Internet space where it's
received by another IPv6 tunnel server.
That server performs the de-encapsulation and
passes the IPv6 traffic further along in the network.
Along with IPv6 tunnel technologies,
the concept of an IPv6 tunnel broker has also emerged.
These are companies that provide IPv6 tunneling endpoints for you, so
you don't have to introduce additional equipment to your network.