



## The Internet: Crash Course Computer Science #29

Crash Course: Computer Science

<https://youtube.com/watch?v=AEaKrq3SpW8>

<https://nerdfighteria.info/v/AEaKrq3SpW8>

Hi, I'm Carrie Anne, and welcome to Crash Course Computer Science!

As we talked about last episode, your computer is connected to a large, distributed network, called The Internet. I know this because you're watching a YouTube video, which is being streamed over that very Internet. It's arranged as an ever-enlarging web of interconnected devices.

For your computer to get this video, the first connection is to your local area network, or LAN, which might be every device in your house that's connected to your wifi router.

This then connects to a Wide Area Network, or WAN, which is likely to be a router run by your Internet Service Provider, or ISP - companies like Comcast, AT&T or Verizon. At first, this will be a regional router, like one for your neighborhood, and then that router connects to an even bigger WAN, maybe one for your whole city or town.

There might be couple more hops, but ultimately you'll connect to the backbone of the internet, made up of gigantic routers with super high-bandwidth connections running between them.

To request this video file from YouTube, a packet had to work its way up to the backbone, travel along that for a bit, and then work its way back down to a YouTube server that had the file.

That might be four hops up, two hops across the backbone, and four hops down, for a total of ten hops. If you're running Windows, MacOS or Linux, you can see the route data takes to different places on the internet by using the traceroute program on your computer. (Instructions are in the Doobly Doo.)

For us here at the Chad & Stacey Emigholz Studio in Indianapolis, the route to the DFTBA server in California goes through 11 stops. We start at 192.168.0.1 - that's the IP address for my computer on our LAN. Then there's the wifi router here at the studio, then a series of regional routers, and then we get onto the backbone, and then we start working back down to the computer hosting "DFTBA.com", which has the IP address 104.24.109.186.

But how does a packet actually get there? What happens if a packet gets lost along the way? If I type "DFTBA.com" into my web browser, how does it know the server's address? These are our topics for today!

[THEME SONG]

As we discussed last episode, the internet is a huge, distributed network that sends data around as little packets. If your data is big enough, like an email attachment, it might get broken up into many packets. For example, this video stream is arriving to your computer right now as a series of packets, and not one gigantic file.

Internet packets have to conform to a standard called the Internet Protocol, or IP. It's a lot like sending physical mail through the postal system - every letter needs a unique and legible address written on it, and there are limits to the size and weight of packages. Violate this, and your letter won't get through.

IP packets are very similar. However, IP is a very low-level protocol - there isn't much more than a destination address in a packet's header, which is the metadata that's stored in front of the data payload. This means that a packet can show up at a computer, but the computer may not know which application to give the data to - Skype or Call of Duty.

For this reason, more advanced protocols were developed that sits on top of IP. One of the simplest and most common is the User Datagram Protocol or UDP. UDP has its own header, which sits inside the data payload. Inside of the UDP header is some useful, extra information. One of them is a port number.

Every program wanting to access the internet will ask its host computer's Operating System to be given a unique port. Like Skype might ask for port number 3478. When a packet arrives to the computer, the Operating System will look inside the UDP header, and read the port number. Then, if it sees, for example, 3478, it will give the packet to Skype.

So to review, IP gets the packet to the right computer, but UDP gets the packet to the right program running on that computer. UDP headers also include something called a checksum, which allows the data to be verified for correctness. As the name suggests, it does this by checking the sum of the data.

Here's a simplified version of how this works:

Let's imagine the raw data in our UDP packet is 89 111 33 32 58 and 41. Before the packet is sent, the transmitting computer calculates the checksum by adding all the data together: 89 plus 111 plus 33 and so on.

In our example, this adds up to a checksum of 364. In UDP, the checksum value is stored in 16 bits. If the sum exceeds the maximum possible value, the upper-most bits overflow, and only the lower bits are used.

Now, when the receiving computer gets this packet, it repeats the process, adding up all the data. 89 plus 11 plus 33 and so on. If that sum is the same as the checksum sent in the header, all is well. But if the numbers don't match, you know that the data got corrupted at some point in transit, maybe because of a power fluctuation or faulty cable.

Unfortunately, UDP doesn't offer any mechanisms to fix the data, or request a new copy - receiving programs are alerted to the corruption, but typically just discard the packet.

Also, UDP provides no mechanisms to know if packets are getting through - a sending computer shoots the UDP packet off, but has no confirmation it ever gets to its destination successfully. Both of these properties sound pretty catastrophic, but some applications are okay with this because UDP is also really simple and fast.

Skype, for example, which uses UDP for video chat, can handle corrupt or missing packets. That's why sometimes if you're on a bad internet connection, Skype gets all glitchy - only some of the UDP packets are making it to your computer.

But this approach doesn't really work for many other types of data transmission. Like, it doesn't really work if you send an email, and it shows up with the middle missing. The whole message really needs to get there correctly!

When it "absolutely, positively needs to get there," programs use the Transmission Control Protocol, or TCP, which like UDP, rides inside the data payload of IP packets. For this reason, people refer to this combination of protocols as TCP/IP.

Like UDP, the TCP header contains a destination port and checksum. But it also contains fancier features, and we'll focus on the key ones.

First off, TCP packets are given sequential numbers. So packet 15 is followed by packet 16, which is followed by 17, and so on, for



## The Internet: Crash Course Computer Science #29

Crash Course: Computer Science

<https://youtube.com/watch?v=AEaKrq3SpW8>

<https://nerdfighteria.info/v/AEaKrq3SpW8>

potentially millions of packets sent during that session.

These sequence numbers allow a receiving computer to put the packets into the correct order, even if they arrive at different times across the network. So if an email comes in all scrambled, the TCP implementation in your computer's operating system will piece it all together correctly.

Second, TCP requires that once a computer has correctly received a packet - and the data passes the checksum - that it sends back an acknowledgment or "ACK" as the cool kids say, to the sending computer. Knowing the packet made it successfully, the sender can now transmit the next packet. But this time, let's say, it waits, and doesn't get an acknowledgment packet back. Something must be wrong.

If enough time elapses, the sender will go ahead and just retransmit the same packet. It's worth noting that the original packet might have actually gotten there, but the acknowledgment is just really delayed. Or perhaps it was the acknowledgment that was lost.

Either way, it doesn't matter, because the receiver has those sequence numbers, and if a duplicate packet arrives, it can be discarded. Also, TCP isn't limited to a back and forth conversation - it can send many packets and have many outstanding ACKs, which increases bandwidth significantly, since you aren't wasting time waiting for acknowledgement packets to return.

Interestingly, the success rate of ACKs, and also the round trip time between sending and acknowledging, can be used to infer network congestion. TCP uses this information to adjust how aggressively it sends packets - a mechanism for congestion control.

So, basically, TCP can handle out-of-order packet delivery, dropped packets - including retransmission - and even throttle its transmission rate according to available bandwidth. Pretty awesome!

You might wonder why anyone would use UDP when TCP has all those nifty features. The single biggest downside are all those acknowledgment packets - it doubles the number of messages on the network, and yet you're not transmitting any more data.

That overhead, including associated delays, is sometimes not worth the improved robustness especially for time-critical applications, like Multi-Player First Person Shooters. And if it's you getting lag-fragged, you'll definitely agree!

When your computer wants to make a connection to a website, you need two things - an IP address and a port. Like port 80, at 172.217.7.238. This example is the IP address and port for the Google web server. In fact, you can enter this into your browser's address bar like so, and you'll end up on the Google homepage.

This gets you to the right destination, but remembering that long string of digits would be really annoying. It's much easier to remember "google.com". So the internet has a special service that maps these domain names to addresses. It's like the phone book for the internet. And it's called the Domain Name System, or DNS for short.

You can probably guess how it works. When you type something like "youtube.com" into your web browser, it goes and asks a DNS server - usually one provided by your ISP - to lookup the address. DNS consults its huge registry and replies with the address - if one exists.

In fact, if you try mashing your keyboard, adding ".com", and then hit enter in your browser, you'll likely be presented with an error that

says DNS failed. That's because the site doesn't exist, so DNS couldn't give your browser an address. But, if DNS returns a valid address, which it should for "youtube.com", then your browser shoots off a request over TCP for the website's data.

There's over 300 million registered domain names, so to make that DNS Lookup a little bit more manageable, it's not stored as one gigantically long list, but rather in a tree data structure.

What are called Top Level Domains, or TLDs, are at the very top. These are huge categories like .com and .gov. Then, there are lower level domains that sit below that, called second level domains; Examples under .com include google.com and dftba.com. Then, then there are even lower level domains, called subdomains, like images.google.com, and store.dftba.com.

And this tree is absolutely HUGE! Like I said, more than 300 million domain names, and that's just second level domain names, not all the subdomains. For this reason, the data is distributed across many DNS servers, which are authorities for different parts of the tree.

Okay, I know you've been waiting for it . . . We've reached **A New Level Of Abstraction!**

[NEW LEVEL OF ABSTRACTION music plays]

Over the past two episodes, we've worked up from electrical signals on wires, or radio signals transmitted through the air in the case of wireless networks. This is called the Physical Layer.

MAC addresses, collision detection, exponential backoff and similar low protocols that mediate access to the physical layer are part of the Data Link Layer. Above this is the Network Layer, which is where all the switching and routing technologies that we discussed operate.

And today, we mostly covered the Transport layer, protocols like UDP and TCP, which are responsible for point to point data transfer between computers, and also things like error detection and recovery when possible.

We've also grazed the Session Layer - where protocols like TCP and UDP are used to open a connection, pass information back and forth, and then close the connection when finished - what's called a session. This is exactly what happens when you, for example, do a DNS Lookup, or request a webpage.

These are the bottom five layers of the Open System Interconnection (OSI) model, a conceptual framework for compartmentalizing all these different network processes. Each level has different things to worry about and solve, and it would be impossible to build one huge networking implementation.

As we've talked about all series, abstraction allows computer scientists and engineers to be improving all these different levels of the stack simultaneously, without being overwhelmed by the full complexity. And amazingly, we're not quite done yet . . .

The OSI model has two more layers, the Presentation Layer and the Application Layer, which include things like web browsers, Skype, HTML decoding, streaming movies, and more. Which we'll talk about next week. See you then!

Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like Physics Girl, Deep Look and PBS SpaceTime. This



## The Internet: Crash Course Computer Science #29

Crash Course: Computer Science

<https://youtube.com/watch?v=AEaKrQ3SpW8>

<https://nerdfighteria.info/v/AEaKrQ3SpW8>

---

episode was filmed at the Chad and Stacy Emigholz Studio in Indianapolis, Indiana and it was made with the help of all of these nice people and our wonderful graphics team Thought Cafe. That's where we're gonna have to halt and catch fire. See you next week.