Crash Course on Networking

Davis Claiborne

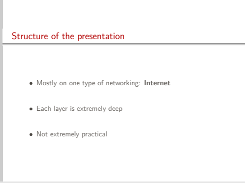
LUG @ NC State

February 16, 2021

Linux Users Group
at NC State University

Structure of the presentation

- Mostly on one type of networking: **Internet**
- Each layer is extremely deep
- Not extremely practical



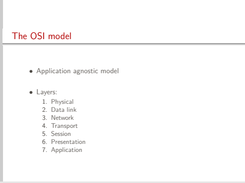
Structure of the presentation

- Mostly on one type of networking: **Internet**
- Each layer is extremely deep
- Not extremely practical

- Networking is a very broad topic, and covers more than just the internet that we typically think of - The internet is just *a* network, one of many
- That being said, I'll mostly stick to talking about the internet, just because it's the most well-known
- I'll try and keep this presentation from drifting too much into just definitions and being technical, so I'll try and throw in some examples, fun facts, etc.
- Also - each layer is extremely deep and they all have entire textbooks written about them. I'm not an expert on any of them.
- This presentation won't be extremely practical - this isn't a networking on Linux tutorial or anything like that; it's mostly on conceptual parts of networking

The OSI model

- Application agnostic model
- Layers:
  1. Physical
  2. Data link
  3. Network
  4. Transport
  5. Session
  6. Presentation
  7. Application



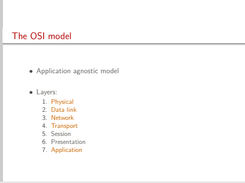
The OSI model

- Application agnostic model
- Layers:
  1. Physical
  2. Data link
  3. Network
  4. Transport
  5. Session
  6. Presentation
  7. Application

- First, we need to start with some technical jargon and definitions
- The OSI model is a 7-layer, application agnostic conceptual model used to separate different parts of communication networks
- Because it's application agnostic, the OSI model can actually be used for any computer/telecommunication protocol - it's just most commonly used for the internet
- The 7 layers are the physical layer, the data link layer, the network layer, the transport layer, the session layer, the presentation layer, and the application layer

The OSI model

- Application agnostic model
- Layers:
  1. Physical
  2. Data link
  3. Network
  4. Transport
  5. Session
  6. Presentation
  7. Application


The OSI model

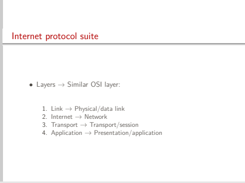
- Application agnostic model
- Layers:
  1. Physical
  2. Data link
  3. Network
  4. Transport
  5. Session
  6. Presentation
  7. Application

- I'll only be talking about the first 4 and the last one in a lot of depth for this presentation, since I think they're more important for understanding the internet
- While the presentation layer is interesting, in my opinion it's just not really necessary to understanding networking overall
- For this presentation, I'll start at the physical layer and work my way up from there

Internet protocol suite

- Layers → Similar OSI layer:

  1. Link → Physical/data link
  2. Internet → Network
  3. Transport → Transport/session
  4. Application → Presentation/application

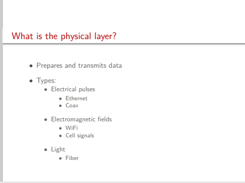

Internet protocol suite

- Layers → Similar OSI layer:
  1. Link → Physical/data link
  2. Internet → Network
  3. Transport → Transport/session
  4. Application → Presentation/application

- It's important to note, though: the internet actually doesn't follow the OSI model; it uses its own model that is roughly similar to it
- For the most part, there's a rough correlation between the two, and, in general, literature refers to the OSI model usually, but it's something important to keep in mind if you're doing your own research on this

What is the physical layer?

- Prepares and transmits data
- Types:
  - Electrical pulses
    - Ethernet
    - Coax
  - Electromagnetic fields
    - WiFi
    - Cell signals
  - Light
    - Fiber



What is the physical layer?

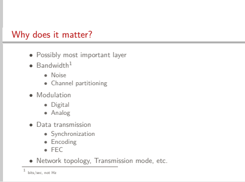
- Prepares and transmits data
- Types:
  - Electrical pulses
    - Ethernet
    - Coax
  - Electromagnetic fields
    - WiFi
    - Cell signals
  - Light
    - Fiber

- The physical layer is the first and lowest layer of the OSI model
- Essentially, the physical layer takes the data from the computer, prepares it for transmission, and then transmits it
- There are tons of different mediums that can be used for data transmission, but the most common ones are electrical pulses along wires, like in Ethernet or coaxial cables, electromagnetic fields, like in WiFi or cell signals, and light, like in optical cables

Why does it matter?

- Possibly most important layer
- Bandwidth[1]
    - Noise
    - Channel partitioning
- Modulation
    - Digital
    - Analog
- Data transmission
    - Synchronization
    - Encoding
    - FEC
- Network topology, Transmission mode, etc.

[1] bits/sec, not Hz



Why does it matter?

- Possibly most important layer
- Bandwidth[1]
  - Noise
  - Channel partitioning
- Modulation
  - Digital
  - Analog
- Data transmission
  - Synchronization
  - Encoding
  - FEC
- Network topology, Transmission mode, etc.

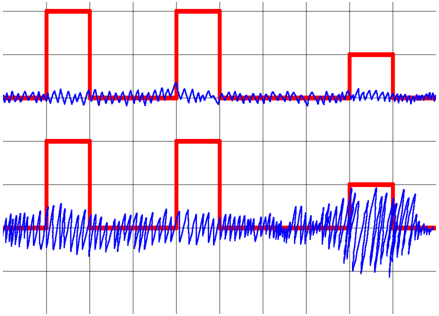
[1] bits/sec, not Hz

- The physical layer is very important - possibly the most important layer - because whatever medium you choose determines things like the bandwidth
- Certain mediums have higher bandwidths than others due to inherent physical properties, which lead to more noise, etc.
- Note that, in most instances, when talking about bandwidth, I'll be referring to the bandwidth in bits/second, **not** the bandwidth in terms of Hertz, which, confusingly, affects the bandwidth of the channel
- Other parameters determined by the physical layer are non-physical properties, like the channel partitioning and encoding scheme used, synchronization, topology, and transmission mode, all of which are determined by the standard being used rather than physical limitations

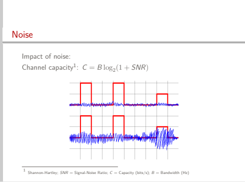
Noise

Impact of noise:

Channel capacity[1]: $C = B\log_2(1 + SNR)$



[1] Shannon-Hartley; SNR = Signal-Noise Ratio; C = Capacity (bits/s); B = Bandwidth (Hz)



Noise

Impact of noise:

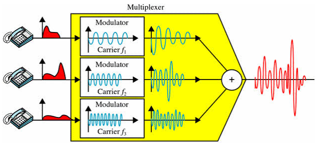
Channel capacity[1]: $C = B\log_2(1 + SNR)$

[1] Shannon-Hartley; SNR = Signal-Noise Ratio; C = Capacity (bits/s); B = Bandwidth (Hz)

- As mentioned before, a number of factors influence a given connection's bandwidth - noise is one of the bigger ones, and one of the ones that can be controlled less
- Certain mediums are more prone to noise and other interference than others
- The Shannon-Hartley capacity gives an approximate estimation of channel capacity given noise
- This can be roughly explained by the picture below - the presence of noise limits the amount of discrete voltage levels that can be detected, limiting the amount of data, limiting the bandwidth

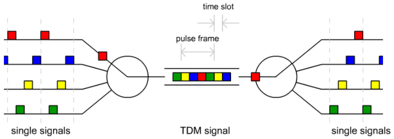
Channel partitioning - Circuit switched

Frequency division multiplexing (FDM):



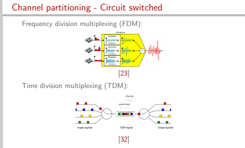
[23]

Time division multiplexing (TDM):



[32]

└Channel partitioning - Circuit switched


Channel partitioning - Circuit switched

Frequency division multiplexing (FDM):

[23]

Time division multiplexing (TDM):

[32]

- Another factor that influences the bandwidth is the channel partitioning scheme - AKA how a shared channel is divided up between its users
- There are two fundamental types of channel partitioning in circuit switched connections: frequency division multiplexing, or FDM, and time division multiplexing, or TDM
- Circuit switched connections are essentially just connections that are created or exist solely for two nodes to communicate with each other - e.g. telephone lines
- One example of FDM would be radio - each station is allocated a certain bandwidth to operate in; in fact, all frequencies are designated for particular activities
- In general, channel partitioning cuts down the total bandwidth allowed to be used by any individual transmitter to allow multiple transmitters to send data over the same medium without interference
- These are just the most simple methods and aren't very efficient - if a source isn't sending any data, that bandwidth is just being wasted
- More sophisticated methods exist to try and avoid issues like this, but I won't go into those for now

Channel partitioning - Packet switched

- Connects any number of points
- Complicated, not part of layer 1 - will discuss later

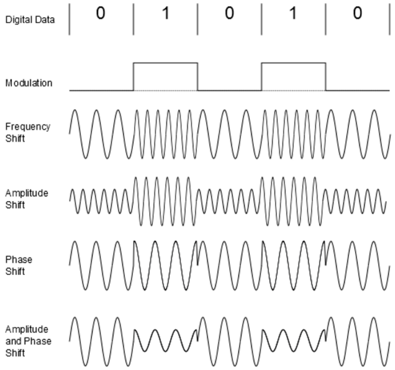
Channel partitioning - Packet switched

- Connects any number of points
- Complicated, not part of layer 1 - will discuss later

- Before, we were talking about circuit switched connections, which connect two points directly
- But what about connections that reach multiple points? For those, packet switching is used
- I'll talk more specifically about how packet switching works later, but for now, know that it reduces the available bandwidth for a source

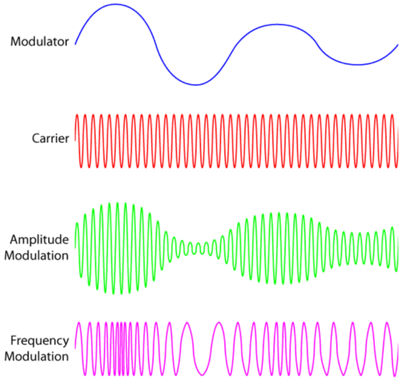
Digital vs analog

Digital modulation (shift keying):

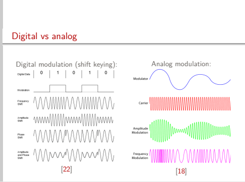


[22]

Analog modulation:



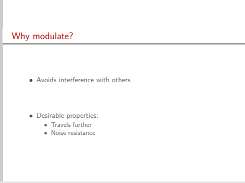
[18]



- The data that you're transmitting also heavily impacts the way you transmit the data
- On the internet, all the information being transmitted is digital - when digital data is modulated onto a carrier signal, it is sometimes called shift keying
- In shift keying, since binary data is being transmitted, only a change in the signal's properties, like its amplitude or frequency, needs to be detected to differentiate the data
- Digital data isn't the only data that needs to be transmitted, though: radio, for instance, is an analog signal that is modulated
- There are tons of different modulation methods that have various practical advantages and disadvantages

Why modulate?

- Avoids interference with others
- Desirable properties:
  - Travels further
  - Noise resistance



Why modulate?

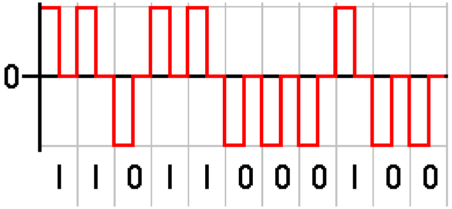
- Avoids interference with others
- Desirable properties:
  - Travels further
  - Noise resistance

- You may be wondering - why bother with modulation, anyways? It seems like it's just complicating stuff for no reason
- There's a lot of complicated math and science going on, most of which are way above my head, but the gist of it is that it has several desirable properties:
- It allows more people to transmit at once without interfering with each other. This is the main advantage - it many people to use the same transmission medium at once without losing any data
- Certain modulation schemes allow transmitted signals to travel further or with less noise

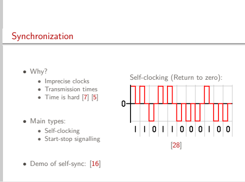
Synchronization

- Why?
  - Imprecise clocks
  - Transmission times
  - Time is hard [7] [5]
- Main types:
  - Self-clocking
  - Start-stop signalling
- Demo of self-sync: [16]

Self-clocking (Return to zero):



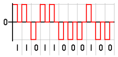
[28]



Synchronization

- Why?
  - Imprecise clocks
  - Transmission times
  - Time is hard [7] [5]
- Main types:
  - Self-clocking
  - Start-stop signalling
- Demo of self-sync: [16]

Self-clocking (Return to zero):



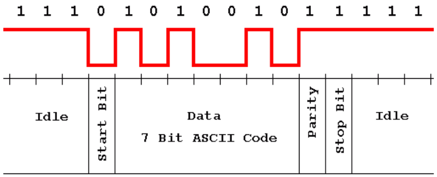
[28]

- Now that we know how much data we can send, we need to make sure that that data is received properly at the other end
- The most obvious solution would be to tell all the receivers to only read signals every so-often to ensure that all data matches up, but this fails to account for a number of problems, such as clock chip inaccuracies and propagation time
- Plus, time is hard - leap seconds and changes in how time is interpreted and defined make it hard to agree on a standard
- Maybe the most obvious solution then is to encode the time within the data being sent itself
- Here you can see one encoding method that does just that - return to zero encoding. With this method, called return to zero encoding, repeated positive or negative voltages aren't even possible, removing the need for synchronization - any signal after a zero can be interpreted as data
- You may be wondering - why is this not the only method? It seems to be pretty simple and intuitive. Well, for one, the receiver now needs to distinguish 3 different voltage levels, complicating it, plus you can't send as much data, since half the time is spent at 0

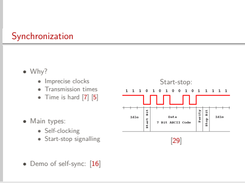
Synchronization

- Why?
  - Imprecise clocks
  - Transmission times
  - Time is hard [7] [5]
- Main types:
  - Self-clocking
  - Start-stop signalling
- Demo of self-sync: [16]

Start-stop:



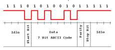
[29]



Synchronization

- Why?
  - Imprecise clocks
  - Transmission times
  - Time is hard [7] [5]
- Main types:
  - Self-clocking
  - Start-stop signalling
- Demo of self-sync: [16]

Start-stop:

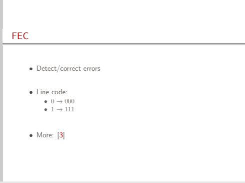


[29]

- There are, of course, more efficient self-clocking signals, but I won't go into them for now
- An alternative is start-stop signalling, which works well for asynchronous data transfer
- Implementations vary based on the standard - sometimes, like in the example here, a constant value is held to indicate an idle status, then the data is transmitted at a pre-determined rate
- In other implementations, a series of pulses is sent to allow the receiver to sync up with the transmitter before data is sent

FEC

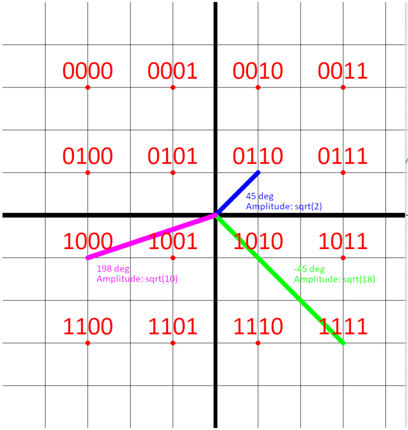
- Detect/correct errors
- Line code:
  - $0 \rightarrow 000$
  - $1 \rightarrow 111$
- More: [3]

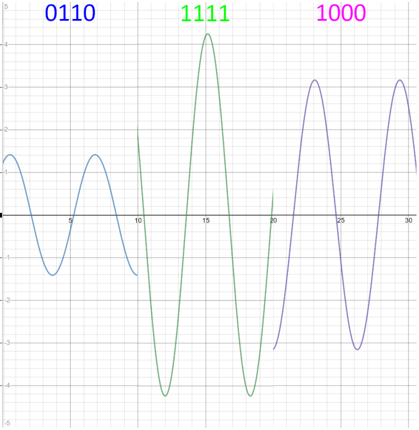

FEC

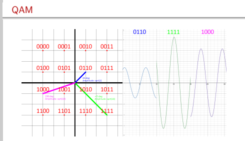
- Detect/correct errors
- Line code:
  - $0 \rightarrow 000$
  - $1 \rightarrow 111$
- More: [3]

- Once you get the signal, even if you sample it at the right time, it's possible that the signal is read incorrectly
- This can be due to noise, signal degradation, or any number of other reasons
- FEC, or forward error correction, is one way to detect and correct simple errors without needing a retransmit
- The most simple, and commonly used method of implementing FEC, is to use what's called a line code, which restricts the valid data sent to a certain range - any data sent that doesn't match a code word is guaranteed to have an error
- Though it adds more bandwidth, certain coding schemes can fix many common errors, through the use of smart encoding and statistical models, without the need to retransmit
- For a very readable introduction to FEC, check out this presentation

QAM


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
45 deg
Amplitude: sqrt(2)
198 deg
Amplitude: sqrt(10)
-45 deg
Amplitude: sqrt(18)

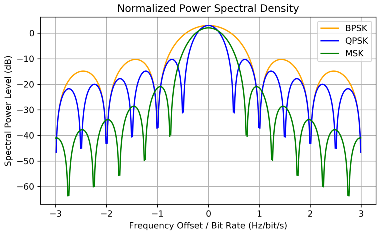

0110
1111
1000


QAM

- QAM is also a pretty interesting multi-bit encoding method
- I won't go too deep into it, but basically you have these constellation charts are used to encode the amplitude and phase offset of the sine wave, allowing you to transmit multiple bits at once

And more...

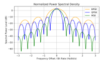
- Unique because of how many fields it covers:
  - Information theory
  - Fourier analysis
  - Material science



[26]

And more...

- Unique because of how many fields it covers:
  - Information theory
  - Fourier analysis
  - Material science

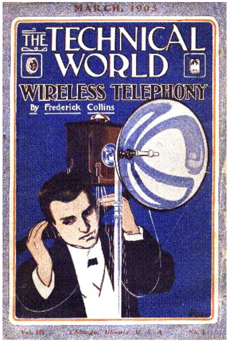


[26]

- There is so much more to the physical layer than what I've talked about so far - this is just a kind of surface-level overview of a lot of the different aspects involved in it
- I personally think layer 1 is one of the most interesting layers, because it combines so many different fields and topics, from information theory to Fourier analysis to material science
- To keep this from being just a layer 1 presentation, though, I won't go too in depth on the remaining topics, but I do want to mention them just to make you aware of the different challenges involved in data transmission
- There's a ton of stuff I didn't even delve into on power considerations; different transmission and encoding schemes use more power than others, and there are trade-offs between picking them
- I also barely even went into the process of modulating and demodulating... there's entire classes just on the math behind the Fourier transforms and all the things that go into that
- Also topology, simplex vs duplex, the actual logic behind FEC, etc.

Fun fact - Photophone

Photophone - predecessor to fiber optic communication



[9]

Fun fact - Photophone

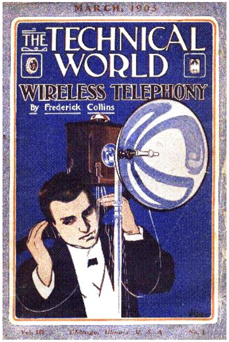
Photophone - predecessor to fiber optic communication

[9]

- Before we move on, I just want to tell a fun fact that doesn't really fit anywhere else, but it's too fun not to share:
- Alexander Graham Bell, inventor of the telephone, also created the modern predecessor to fiber optics, the photophone

Fun fact - Photophone

Photophone - predecessor to fiber optic communication



[9]

Bell wanted to name his second daughter after it [9]

[19]

Fun fact - Photophone

Photophone - predecessor to fiber optic communication

[9]

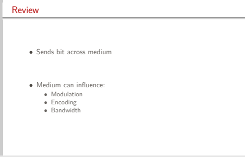
Bell wanted to name his second daughter after it [9]

[19]

- The photophone was actually the device he was the most proud of; he was so proud of it that he wanted to name his second daughter “Photophone”
- They went with “Marian” instead

Review

- Sends bit across medium
- Medium can influence:
  - Modulation
  - Encoding
  - Bandwidth



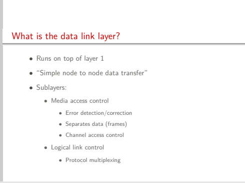
Review

- Sends bit across medium
- Medium can influence:
  - Modulation
  - Encoding
  - Bandwidth

- That was a lot, so I just want to summarize the key points if you take nothing else away from that section
- The physical layer is responsible for actually transmitting data
- Depending on the medium, distance, etc, different modulation and encoding schemes will be used

What is the data link layer?

- Runs on top of layer 1
- “Simple node to node data transfer”
- Sublayers:
  - Media access control
    - Error detection/correction
    - Separates data (frames)
    - Channel access control
  - Logical link control
    - Protocol multiplexing



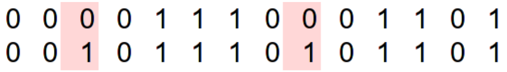
What is the data link layer?

- Runs on top of layer 1
- "Simple node to node data transfer"
- Sublayers:
  - Media access control
    - Error detection/correction
    - Separates data (frames)
    - Channel access control
  - Logical link control
    - Protocol multiplexing

- If the physical layer sends data between two points; the data link layer manages the data to ensure it's been sent properly
- Another common way to say what it does is that it provides (simple) node to node transfer
- It does this through two sublayers: the media access control layer and the logical link control layer
- Just like the physical layer, it can also provide error detection, as well as sometimes correction
- The reason both the physical and data link layers provide error detection/correction is because they both **can**, but not all implementations do, so sometimes it falls on another layer to do it
- It may chunk the data into sizes more appropriate for the physical layer implementation being used
- When the data is separated like this, it's called a "frame"
- The data link layer can also control when data is sent to the physical layer in an effort to reduce transmission collisions as much as possible
- Finally, the logical link control sublayer provides protocol multiplexing

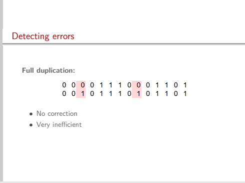
Detecting errors

Full duplication:



0	0	0	0	1	1	1	0	0	0	1	1	0	1
0	0	1	0	1	1	1	0	1	0	1	1	0	1

- No correction
- Very inefficient



Detecting errors

Full duplication:

0	0	0	0	1	1	1	0	0	0	1	1	0	1
0	0	1	0	1	1	1	0	1	0	1	1	0	1

- No correction
- Very inefficient

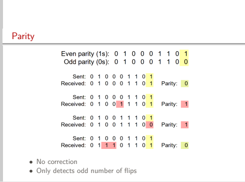
- Another important part of data transmission is ensuring that the data is received properly - noise, interference, and signal degradation can cause the signal to be read incorrectly
- In order to prevent against that, we need to come up with a way to at least detect, if not correct, data transmission errors
- The most obvious way to detect errors would be to just have every single bit of data be sent twice, but this has some major disadvantages: for one, it's impossible to know which message is actually the "right" one; it also, at best, halves the amount of data you can send, and actually quadruples the data, since you need to retransmit errors

Parity

Even parity (1s):	0	1	0	0	0	1	1	0	1
Odd parity (0s):	0	1	0	0	0	1	1	0	0

Sent:	0	1	0	0	0	1	1	0	1		
Received:	0	1	0	0	0	1	1	0	1	Parity:	0
Sent:	0	1	0	0	0	1	1	0	1		
Received:	0	1	0	0	1	1	1	0	1	Parity:	1
Sent:	0	1	0	0	1	1	1	0	1		
Received:	0	1	0	0	1	1	1	0	0	Parity:	1
Sent:	0	1	0	0	0	1	1	0	1		
Received:	0	1	1	1	0	1	1	0	1	Parity:	0

- No correction
- Only detects odd number of flips



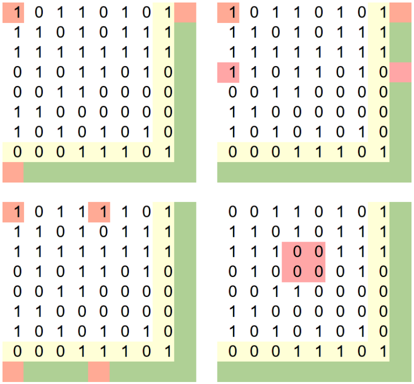
Parity

```
Even parity (1s):  0  1  0  0  0  1  1  0  1
 Odd parity (0s):  0  1  0  0  0  1  1  0  0

    Sent:  0 1 0 0 0 1 1 0 1
Received:  0 1 0 0 0 1 1 0 1    Parity:  0

    Sent:  0 1 0 0 0 1 1 0 1
Received:  0 1 0 0 1 1 1 0 1    Parity:  1

    Sent:  0 1 0 0 1 1 1 0 1
Received:  0 1 0 0 1 1 1 0 0    Parity:  1

    Sent:  0 1 0 0 0 1 1 0 1
Received:  0 1 1 1 0 1 1 0 1    Parity:  0
```

- No correction
- Only detects odd number of flips

- So what's a better way to detect errors?
- A much more efficient way to detect errors is to add parity bits - basically just count how many ones or zeros you see, and add that bit to the end
- The two types of parity - even and odd. In even parity, you add a digit so that the total number of ones is even; for odd parity, you add a number to make the total number of ones odd
- As long as the sender and receiver agree on a parity forma, this works well for small pieces of data, but can't detect situations where an even number of bits is flipped

2D parity



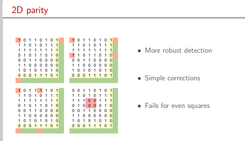
1	0	1	1	0	1	0	1
1	1	0	1	0	1	1	1
1	1	1	1	1	1	1	1
0	1	0	1	1	0	1	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1

1	0	1	1	0	1	0	1
1	1	0	1	0	1	1	1
1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1

1	0	1	1	1	1	0	1
1	1	0	1	0	1	1	1
1	1	1	1	1	1	1	1
0	1	0	1	1	0	1	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1

0	0	1	1	0	1	0	1
1	1	0	1	0	1	1	1
1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	0
1	1	0	0	0	0	0	0
1	0	1	0	1	0	1	0
0	0	0	1	1	1	0	1

- More robust detection
- Simple corrections
- Fails for even squares



- One way to make parity more robust is to do it in two directions
- By grouping blocks of data, you can take the parity of a column, allowing you to detect more types of errors, and even perform simple corrections (in some instances)
- While 2d parity is a great improvement, and doesn't add too much more overhead either, it still fails at detecting some errors, like squares with an even number of bits
- In reality, these methods aren't used too frequently on the internet, mostly because there are schemes that are more efficient and better at detecting errors
- Additionally, none of these methods detect situations where order is swapped but no other errors are present

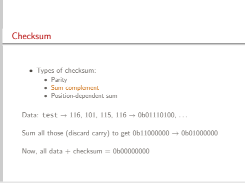
Checksum

- Types of checksum:
  - Parity
  - Sum complement
  - Position-dependent sum

Data: `test` $\rightarrow$ 116, 101, 115, 116 $\rightarrow$ 0b01110100, ...

Sum all those (discard carry) to get 0b11000000 $\rightarrow$ 0b01000000

Now, all data + checksum = 0b00000000



Checksum

- Types of checksum:
  - Parity
  - Sum complement
  - Position-dependent sum

Data: `test` → 116, 101, 115, 116 → 0b01110100, . . .

Sum all those (discard carry) to get 0b11000000 → 0b01000000

Now, all data + checksum = 0b00000000

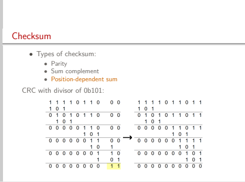
- Checksums are basically broader versions of parity checks - regular 1d parity is sometimes called a “longitudinal” checksum
- They’re literally just operations on the data used to check it and ensure that it’s right (though note that the operation done isn’t always a sum)
- Other than parity, another form of a checksum is the sum complement, where you sum all the data together, then take the twos complement, so that when the data is re-summed you get 0 if there were no errors
- Sum complement has similar issues to parity - namely it misses rearranged data, and bit flips at the same index in different words are also missed, though the odds of that happening are pretty low

Checksum

- Types of checksum:
  - Parity
  - Sum complement
  - Position-dependent sum

CRC with divisor of 0b101:

```
1 1 1 1 0 1 1 0   0 0        1 1 1 1 0 1 1 0 1 1
1 0 1                        1 0 1
-------------------          -------------------
0 1 0 1 0 1 1 0   0 0        0 1 0 1 0 1 1 0 1 1
  1 0 1                        1 0 1
-------------------          -------------------
0 0 0 0 0 1 1 0   0 0        0 0 0 0 0 1 1 0 1 1
          1 0 1                        1 0 1
-------------------    →     -------------------
0 0 0 0 0 0 1 1   0 0        0 0 0 0 0 0 1 1 1 1
            1 0   1                      1 0 1
-------------------          -------------------
0 0 0 0 0 0 0 1   1 0        0 0 0 0 0 0 0 1 0 1
              1   0 1                      1 0 1
-------------------          -------------------
0 0 0 0 0 0 0 0   1 1        0 0 0 0 0 0 0 0 0 0
```



Checksum

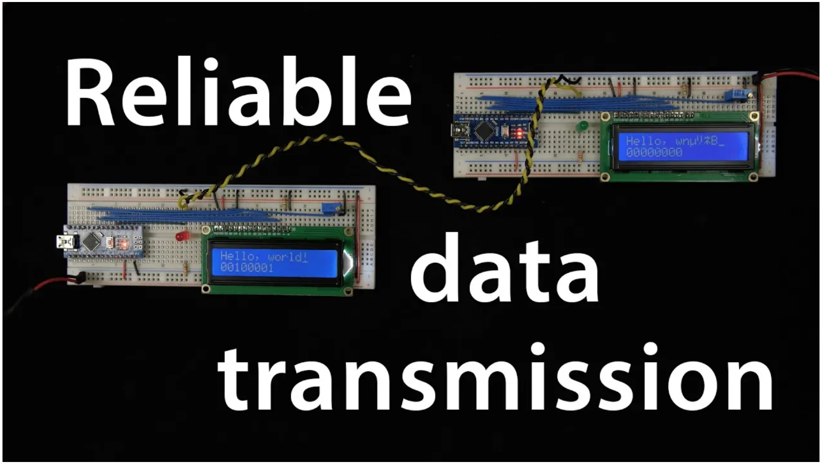
- Types of checksum:
    - Parity
    - Sum complement
    - Position-dependent sum

CRC with divisor of 0b101:

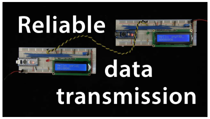
```
1 1 1 1 0 1 1 0   0 0      1 1 1 1 0 1 1 0 1 1
1 0 1                      1 0 1
-------------------      -------------------
0 1 0 1 0 1 1 0   0 0      0 1 0 1 0 1 1 0 1 1
  1 0 1                      1 0 1
-------------------      -------------------
0 0 0 0 0 1 1 0   0 0      0 0 0 0 0 1 1 0 1 1
          1 0 1                      1 0 1
-------------------  →   -------------------
0 0 0 0 0 0 1 1   0 0      0 0 0 0 0 0 1 1 1 1
            1 0   1                    1 0 1
-------------------      -------------------
0 0 0 0 0 0 0 1   1 0      0 0 0 0 0 0 0 1 0 1
              1   0 1                    1 0 1
-------------------      -------------------
0 0 0 0 0 0 0 0   1 1      0 0 0 0 0 0 0 0 0 0
```

- Another better implementation would be position-dependent sums
- These implementations avoid the problem of missing swapped words by making the order have an impact on the sum
- The CRC is one of the most common of these
- As a side note, I'm not exactly sure why it's considered a checksum, since it doesn't seem to be doing any summing to me, but what do I know
- The basic principles of the CRC is that you choose a particular prime divisor of length n, and pad your data with $n - 1$ zeros.
- The basic operation is as follows:
- While there are ones outside of the padded bits, choose the most significant 1 and XOR, set the result as the new data to operate on
- The value of the padded bits is appended to the original data; now when CRC is done with the same divisor, if the data is unchanged the result will be 0

Learn more



[12]

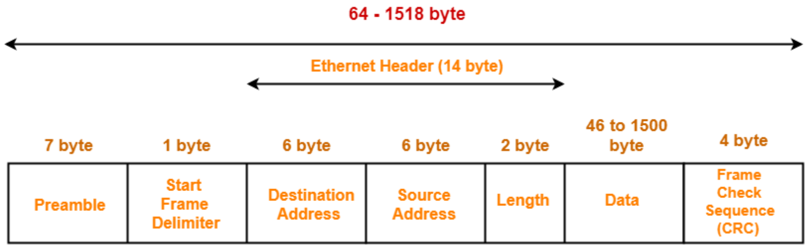


[12]

- If this is interesting to you, I recommend you check out this playlist

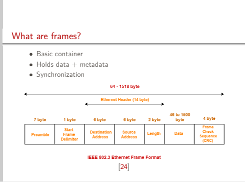
What are frames?

- Basic container
- Holds data + metadata
- Synchronization



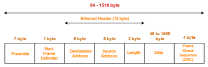
IEEE 802.3 Ethernet Frame Format

[24]



What are frames?

- Basic container
- Holds data + metadata
- Synchronization



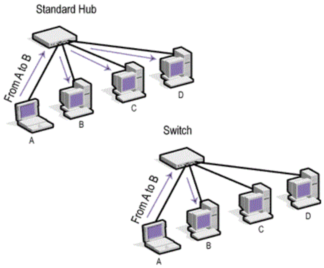
IEEE 802.3 Ethernet Frame Format

[24]

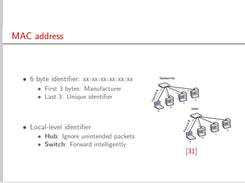
- Frames are basically just containers that hold the data being sent over the physical layer, plus a bit of extra data
- Frames can also act as a synchronization tool to help align the transmitter and receiver
- Here's an example of a frame - the Ethernet frame format
- The preamble is a series of alternating 0s and 1s to allow synchronization
- The start frame delimiter (SFD), which basically is just a pattern that marks the end of the synchronization period (even though the preamble is a consistent length, it's possible parts of the preamble may have been missed or lost)
- The source and destination addresses are the MAC address, which I'll explain in a bit
- Next, the length of the **entire** frame, including headers. This is needed because the data part of the frame can be a variable length
- Finally, at the end of the frame is the CRC taken over the addresses and onwards

MAC address

- 6 byte identifier: xx:xx:xx:xx:xx:xx
  - First 3 bytes: Manufacturer
  - Last 3: Unique identifier
- Local-level identifier
  - **Hub**: Ignore unintended packets
  - **Switch**: Forward intelligently

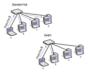


[31]



MAC address

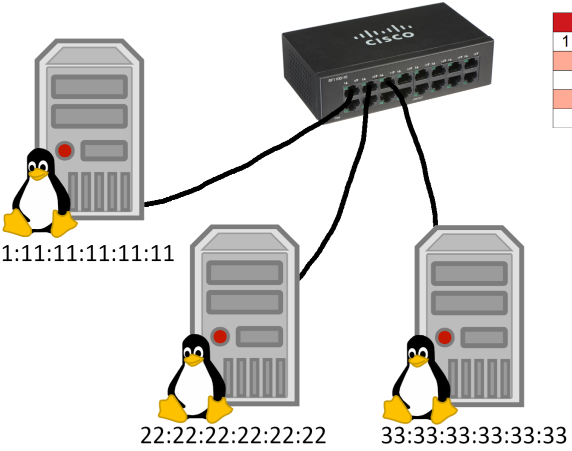
- 6 byte identifier: xx:xx:xx:xx:xx:xx
  - First 3 bytes: Manufacturer
  - Last 3: Unique identifier
- Local-level identifier
  - **Hub**: Ignore unintended packets
  - **Switch**: Forward intelligently



[31]

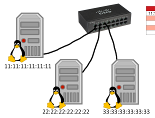
- So I talked about MAC addresses already, but I didn't really talk about what they were
- MAC addresses are 6 byte numbers assigned to network interfaces whey they're manufactured
- They're usually displayed as hex numbers by byte; the first three indicate the manufacturer, and the last 3 are just unique to the device
- The main reason for a MAC address is to coordinate traffic on a very local level
- On older networking hardware, called hubs, messages would just be broadcast to every device connected to the hub; in this scenario, nodes use the MAC address to filter to data only intended for them
- Newer hardware, called switches, are much smarter about how they forward traffic
- Note that switches and hubs themselves do not have MAC addresses; they only forward data

How do switches work?



MAC	Port
11:11:11:11:11:11	1

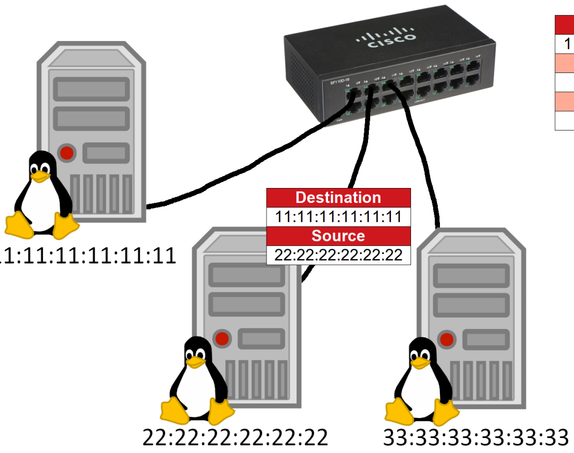
How do switches work?



MAC	Port
11:11:11:11:11:11	1

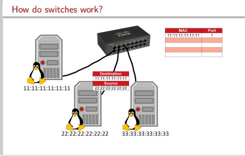
- You may be wondering - how are switches able to know which device to send data to?
- Let's look at an example - we'll assume the forwarding table already has one entry for now just for simplicity

How do switches work?



Destination
11:11:11:11:11:11
Source
22:22:22:22:22:22

MAC	Port
11:11:11:11:11:11	1

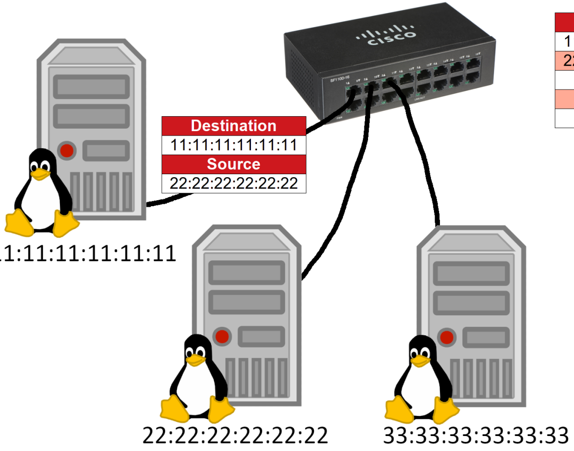


How do switches work?

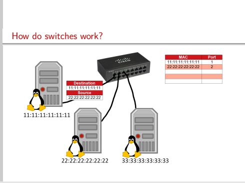
MAC	Port
11:11:11:11:11:11	1

- Switches use forwarding tables that are constructed dynamically to determine where to send frames

How do switches work?



MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2

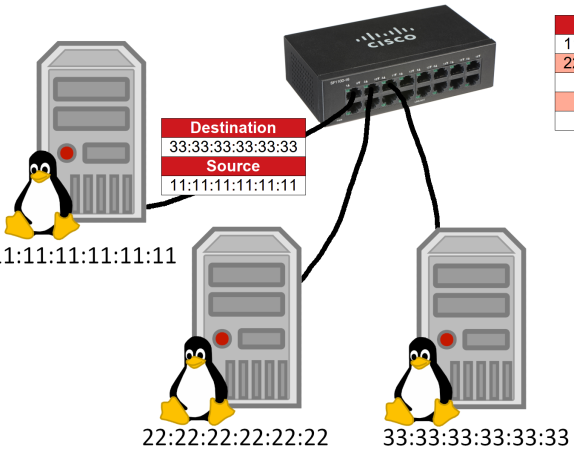


How do switches work?

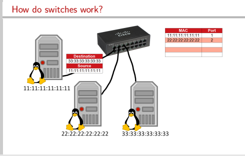
MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2

- Any time a switch receives a frame, it makes a note of the MAC that sent the data and the port which it was sent on, then sends it on to the intended destination

How do switches work?



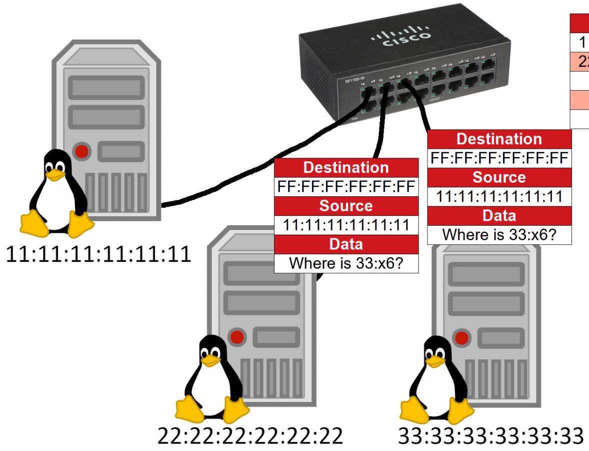
MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2


How do switches work?

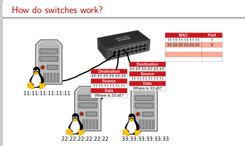
MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2

- But what about if the switch needs to send data to an unknown MAC?

How do switches work?



MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2

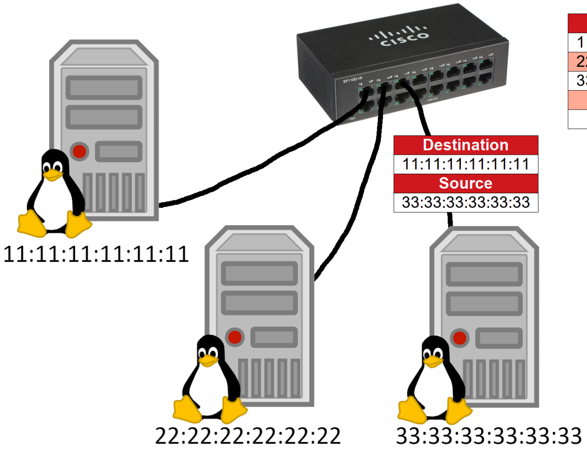


How do switches work?

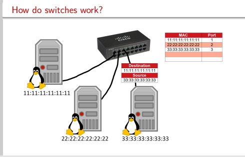
MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2

- The switch can send a broadcast message asking all connected devices if they have the given MAC address
- Broadcast messages are sent to a destination of FF:x6
- You may be wondering - if the switch already knows that 22:x6 is on port 2, why ask it anyways?
- The reason is because there can be multiple switches on a LAN, meaning there can be more than one MAC address on a single port

How do switches work?



MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2
33:33:33:33:33:33	3

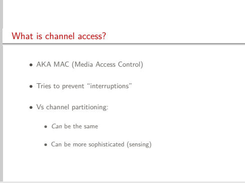

How do switches work?

MAC	Port
11:11:11:11:11:11	1
22:22:22:22:22:22	2
33:33:33:33:33:33	3

- The intended device will then respond, identifying itself
- Its response is noted and added to the forwarding table
- An additional note: each entry in the table has a certain "life expectancy," so after a certain amount of time it will be removed from the table, allowing it to be more dynamic

What is channel access?

- AKA MAC (Media Access Control)
- Tries to prevent “interruptions”
- Vs channel partitioning:
    - *Can* be the same
    - Can be more sophisticated (sensing)


What is channel access?

- AKA MAC (Media Access Control)
- Tries to prevent "interruptions"
- Vs channel partitioning:
  - *Can* be the same
  - Can be more sophisticated (sensing)

- Confusingly, sometimes called MAC (media access control)
- So now that we've talked some about how to make sure the data is transmitted to the right place with no errors, how do we make sure that two senders don't send data over top of each other? And how does this differ from the channel partitioning of layer 1?
- As it turns out, the difference to channel partitioning isn't all that much - some layer 1 implementations don't have any kind of partitioning, so that may need to be done at a higher level
- On the other hand, channel access is a bit more sophisticated, because it can attempt to sense whether something is being broadcast over it already, and will wait until that's finished to start

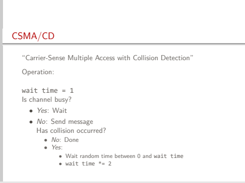
CSMA/CD

“Carrier-Sense Multiple Access with Collision Detection”

Operation:

`wait time = 1`
Is channel busy?

- *Yes*: Wait
- *No*: Send message
  Has collision occurred?
  - *No*: Done
  - *Yes*:
    - Wait random time between 0 and `wait time`
    - `wait time *= 2`



CSMA/CD

"Carrier-Sense Multiple Access with Collision Detection"

Operation:

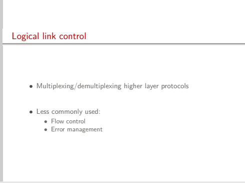
`wait time = 1`
Is channel busy?

- *Yes*: Wait
- *No*: Send message
  Has collision occurred?
  - *No*: Done
  - *Yes*:
    - Wait random time between 0 and `wait time`
    - `wait time *= 2`

- The MAC-sublayer channel access mode used in Ethernet is called CSMA/CD, which stands for Carrier-sense multiple access with collision detection
- The way it works is pretty simple: the node can detect (sense) that a signal is currently being transmitted and waits until it's finished
- At that point, it will transmit its own data and attempt to detect if a collision has occurred
- Why does it need to detect a collision if it waits until there's no transmission happening?
- Basically because two nodes can observe that the channel is empty and simultaneously broadcast a message
- Collision detection changes depending on the medium, but in general involves monitoring signal levels for unexpected/unusual signals
- After a collision is detected, both nodes will wait a random amount of time that increases on each collision before retransmitting; this is known as exponential backoff
- The random delay is very important - it prevents two nodes from just continuously colliding with each other
- This is falling out of favor now with Ethernet, due to the popularity of switches and full duplex wires

Logical link control

- Multiplexing/demultiplexing higher layer protocols
- Less commonly used:
    - Flow control
    - Error management



Logical link control

- Multiplexing/demultiplexing higher layer protocols
- Less commonly used:
  - Flow control
  - Error management

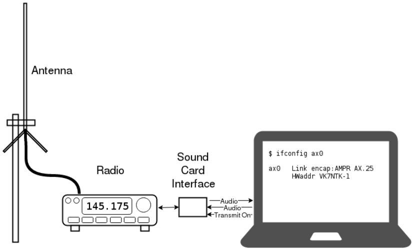
- The link layer's other sublayer - logical link control - mostly just does one thing - multiplexing the signals from layers above
- This allows multiple protocols to exist on the same network
- There are other features in the logical link layer, like flow control and error management, but since they aren't as commonly used by protocols on the internet, I won't talk about them

Fun fact - Radio networking

AKA Packet radio

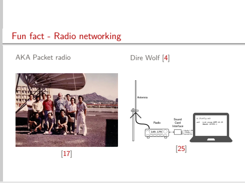
[17]

Dire Wolf [4]



```
$ ifconfig ax0

ax0    Link encap:AMPR AX.25
       HWaddr VK7NTK-1
```

[25]


Fun fact - Radio networking

AKA Packet radio

Dire Wolf [4]

[17]

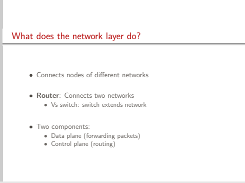
[25]

- Though the vast majority of the internet is connected with copper and fiber, it is possible to form a network with simple amateur radio equipment - this is typically called packet radio
- The first radio-based network was the ALOHAnet, made in Hawaii because the expense of making wired connections between islands was prohibitively expensive
- ALOHAnet paved the way for medium access controls and other concepts frequently found in layer 2 protocols nowadays
- Though it's not nearly as popular now, there are still HAM radio operators around the world with radio-only networks and BBS
- The most popular software to do this with is called Dire Wolf

What does the network layer do?

- Connects nodes of different networks
- **Router**: Connects two networks
    - Vs switch: switch extends network
- Two components:
    - Data plane (forwarding packets)
    - Control plane (routing)

└What does the network layer do?

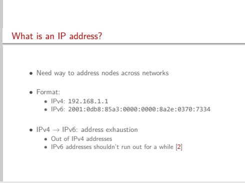

What does the network layer do?

- Connects nodes of different networks
- **Router**: Connects two networks
  - Vs switch: switch extends network
- Two components:
  - Data plane (forwarding packets)
  - Control plane (routing)

- The network layer connects nodes in different networks together
- For a bit of terminology, the networking definition of a router is something that connects two networks together
- Different from a switch, which just extends a network
- For the internet, it's often broken into two layers: the data plane and the control plane
- The data plane is responsible for forwarding packets to the right IP address
- Remember on the last layer how, when data was encapsulated it was called a frame? Well, encapsulated frames are called "Datagrams"
- The control plane is in charge of determining the routing tables

What is an IP address?

- Need way to address nodes across networks
- Format:
  - IPv4: `192.168.1.1`
  - IPv6: `2001:0db8:85a3:0000:0000:8a2e:0370:7334`
- IPv4 $\rightarrow$ IPv6: address exhaustion
  - Out of IPv4 addresses
  - IPv6 addresses shouldn't run out for a while [2]



What is an IP address?

- Need way to address nodes across networks
- Format:
  - IPv4: `192.168.1.1`
  - IPv6: `2001:0db8:85a3:0000:0000:8a2e:0370:7334`
- IPv4 → IPv6: address exhaustion
  - Out of IPv4 addresses
  - IPv6 addresses shouldn't run out for a while [2]

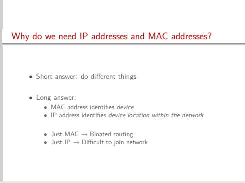
- In order to connect different networks together, we need some kind of addressing mechanism, just like the link layer
- For the internet's implementation, this address is called the IP address
- The format for this address changes depending on which protocol you're using
- The older version, IPv4, uses 32 bit number identifiers, which are usually broken up and represented as 4 8-bit decimal numbers
- IPv6 uses 128 bit number identifiers, which are usually broken up as 8 groups of four hex digits
- We've started moving from IPv4 due to something called address exhaustion, which is caused by there being more hosts on the internet than there are available IPv4 addresses
- This shouldn't ever be a problem with IPv6 (at least for an incredibly long time), as we have enough IPv6 addresses to uniquely number every single grain of sand on around 340 billion Earths

Why do we need IP addresses and MAC addresses?

- Short answer: do different things

- Long answer:
  - MAC address identifies *device*
  - IP address identifies *device location within the network*

  - Just MAC $\rightarrow$ Bloated routing
  - Just IP $\rightarrow$ Difficult to join network



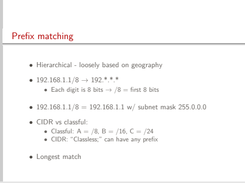
Why do we need IP addresses and MAC addresses?

- Short answer: do different things
- Long answer:
  - MAC address identifies *device*
  - IP address identifies *device location within the network*
  - Just MAC → Bloated routing
  - Just IP → Difficult to join network

- You may be wondering - why do we need both MAC and IP addresses? Why not just use the MAC address or just the IP address?
- IP addresses provide built-in information about the structure of the network, allowing routing to be more efficient; if we used MAC addresses for identifying nodes between networks, our routing software and hardware would have to be much more complex
- So then why MAC addresses? These allow people to move their computers to different networks whenever they want - you'd need to get a new IP address every time you changed networks, but you'd have no identifier to your computer, making it difficult (if not impossible) to dynamically assign IP addresses

Prefix matching

- Hierarchical - loosely based on geography
- 192.168.1.1/8 $\rightarrow$ 192.*.*.*
  - Each digit is 8 bits $\rightarrow$ /8 = first 8 bits
- 192.168.1.1/8 = 192.168.1.1 w/ subnet mask 255.0.0.0
- CIDR vs classful:
  - Classful: A = /8, B = /16, C = /24
  - CIDR: "Classless;" can have any prefix
- Longest match

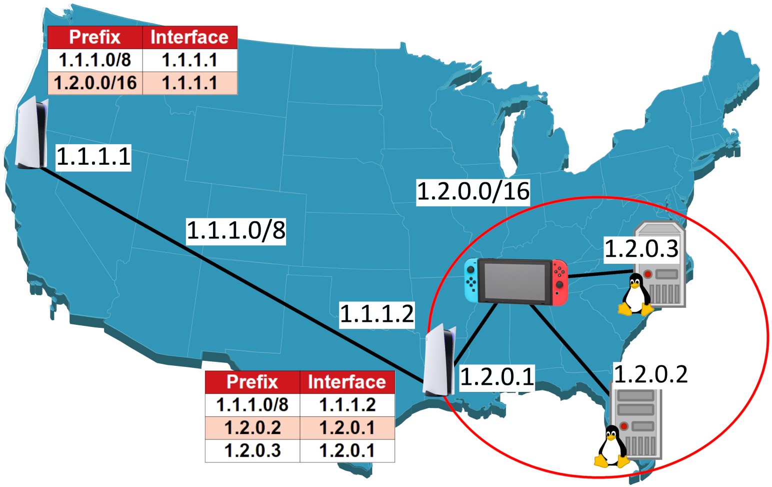


Prefix matching

- Hierarchical - loosely based on geography
- 192.168.1.1/8 → 192.*.*.*
  - Each digit is 8 bits → /8 = first 8 bits
- 192.168.1.1/8 = 192.168.1.1 w/ subnet mask 255.0.0.0
- CIDR vs classful:
  - Classful: A = /8, B = /16, C = /24
  - CIDR: “Classless;” can have any prefix
- Longest match

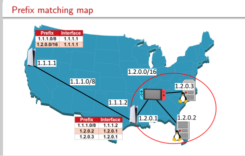
- So I just mentioned that the IP address provides built-in routing information. How does it do that?
- The IP address is hierarchical - ISPs are allocated a certain range of IP addresses to provide to their clients; these ISPs are usually broken up further by the ISP based on the geographic location of the client
- By regimenting the addresses in this way, routers can be efficient with their routing without making their routing tables massive, allowing for faster routing and cheaper equipment
- They way this hierarchy is represented is through subnets
- Subnets usually take the form of an IP address with a slash indicating the number of bits to match, though they sometimes will take on the form of binary ones that match in decimal
- The current prefix system is called CIDR, which stands for classless Inter-Domain Routing
- The previous implementation divided prefixes into designated classes and didn’t work well, since groupings were usually either too big or too large
- A router can have multiple prefixes that overlap - in this case, the longest matching prefix is used

Prefix matching map



Prefix	Interface
1.1.1.0/8	1.1.1.1
1.2.0.0/16	1.1.1.1

Prefix	Interface
1.1.1.0/8	1.1.1.2
1.2.0.2	1.2.0.1
1.2.0.3	1.2.0.1


Prefix matching map

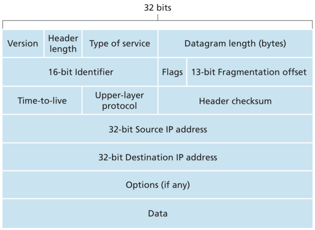
Prefix	Interface
1.1.1.0/8	1.1.1.1
1.2.0.0/16	1.1.1.1

Prefix	Interface
1.1.1.0/8	1.1.1.2
1.2.0.2	1.2.0.1
1.2.0.3	1.2.0.1

- Now let's take a look at that in practice
- A few things before we start: it's important to note that each **network interface**, not host, gets an IP address
- The circled red area has a subnet of 1.2.0.0/16 and the connection between CA and LA has a subnet 1.1.1.0/8
- Looking at the router in CA, we can immediately see the advantage of hierarchical IP addresses: instead of having to store 3 entries in its forwarding table, it only needs to store 2
- The advantage is pretty insignificant here, but it's much larger once more hosts and subnets are added
- We can also see a switch in action: the LA router always sends 1.2.*.* addresses out its 1.2.0.1 interface and the switch takes care of the rest
- One other note: routers will usually have the .1 IP address; this isn't a standard or anything, but it's a relatively common technique to make them a bit easier to find

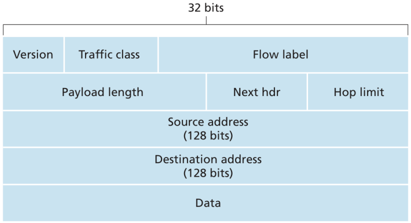
IPv4/IPv6

IPv4:



[20]

IPv6:

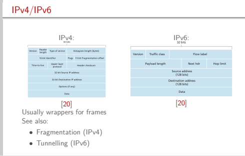


[20]

Usually wrappers for frames

See also:

- Fragmentation (IPv4)
- Tunnelling (IPv6)



IPv4/IPv6

IPv4:

[20]

IPv6:

[20]

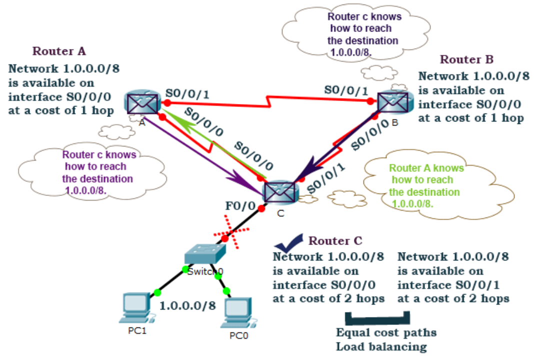
Usually wrappers for frames
See also:

- Fragmentation (IPv4)
- Tunnelling (IPv6)

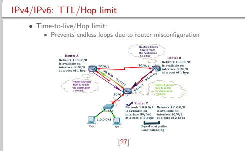
- I mostly haven't talked too much about protocols so far, mostly just because I don't want to get too into the specifics for the most part, but I do think talking about some of these protocols helps illustrate what the data plane does
- I won't go into detail on most of these fields, really only TTL, AKA hop-limit in IPv6, but fragmentation and tunnelling are both interesting solutions to problems with the IP format
- Note that, usually, the data contained within is a lower-level frame

IPv4/IPv6: TTL/Hop limit

- Time-to-live/Hop limit:
  - Prevents endless loops due to router misconfiguration

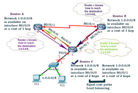

Router c knows how to reach the destination 1.0.0.0/8.
Router A
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 1 hop
Router B
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 1 hop
S0/0/1
S0/0/1
S0/0/0
S0/0/0
S0/0/0
S0/0/1
A
B
C
Router c knows how to reach the destination 1.0.0.0/8.
Router A knows how to reach the destination 1.0.0.0/8.
F0/0
Router C
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 2 hops
Network 1.0.0.0/8 is available on interface S0/0/1 at a cost of 2 hops
Equal cost paths
Load balancing
Switch0
1.0.0.0/8
PC1
PC0

[27]



IPv4/IPv6: TTL/Hop limit

- Time-to-live/Hop limit:
  - Prevents endless loops due to router misconfiguration


Router c knows how to reach the destination 1.0.0.0/8.
Router A
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 1 hop
Router B
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 1 hop
S0/0/1
S0/0/1
S0/0/0
S0/0/0
S0/0/0
S0/0/1
Router c knows how to reach the destination 1.0.0.0/8.
Router A knows how to reach the destination 1.0.0.0/8.
F0/0
Router C
Network 1.0.0.0/8 is available on interface S0/0/0 at a cost of 2 hops
Network 1.0.0.0/8 is available on interface S0/0/1 at a cost of 2 hops
Equal cost paths Load balancing
1.0.0.0/8
PC1
PC0

[27]

- TTL, which was renamed to "hop limit" in IPv6, basically is to keep data from endlessly being sent back and forth if a router is misconfigured: each time its sent somewhere, that field is decremented; when it reaches 0, the packet is dropped
- Below you can see a picture of a routing loop - if not for TTL, packets sent in this way would endlessly congest the network

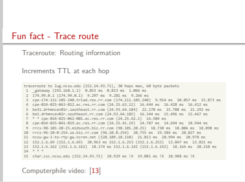
Fun fact - Trace route

Traceroute: Routing information

Increments TTL at each hop

```
traceroute to lug.ncsu.edu (152.14.93.71), 30 hops max, 60 byte packets
 1  _gateway (192.168.1.1)  0.853 ms  0.813 ms  1.066 ms
 2  174.99.0.1 (174.99.0.1)  9.297 ms  9.281 ms  9.266 ms
 3  cpe-174-111-105-240.triad.res.rr.com (174.111.105.240)  9.954 ms  10.057 ms  15.873 ms
 4  cpe-024-025-063-012.ec.res.rr.com (24.25.63.12)  16.444 ms  16.428 ms  16.412 ms
 5  be31.drhmncev01r.southeast.rr.com (24.93.64.184)  22.170 ms  15.788 ms  21.292 ms
 6  be1.drhmncev02r.southeast.rr.com (24.93.64.185)  16.344 ms  15.496 ms  15.467 ms
 7  * * cpe-024-025-062-001.ec.res.rr.com (24.25.62.1)  16.504 ms
 8  cpe-024-025-041-019.ec.res.rr.com (24.25.41.19)  14.787 ms  14.694 ms  18.944 ms
 9  rrcs-98-101-20-25.midsouth.biz.rr.com (98.101.20.25)  18.730 ms  18.806 ms  18.898 ms
10  rrcs-96-10-0-254.se.biz.rr.com (96.10.0.254)  20.755 ms  19.584 ms  20.827 ms
11  ncsu-gw-1-to-rtp-gw.ncren.net (128.109.18.110)  21.013 ms  20.994 ms  20.978 ms
12  152.1.6.69 (152.1.6.69)  20.963 ms 152.1.6.253 (152.1.6.253)  13.847 ms  13.821 ms
13  152.1.6.162 (152.1.6.162)  18.174 ms 152.1.6.142 (152.1.6.142)  18.164 ms  20.210 ms
14  * * *
15  char.csc.ncsu.edu (152.14.93.71)  18.929 ms !X  19.081 ms !X  18.988 ms !X
```

Computerphile video: [13]



Fun fact - Trace route

Traceroute: Routing information

Increments TTL at each hop

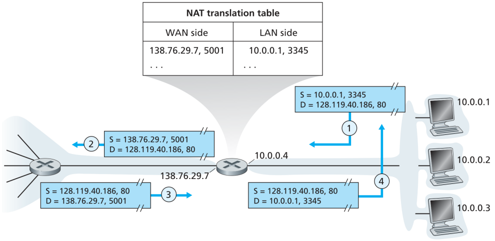
```
traceroute to lug.ncsu.edu (152.14.93.71), 30 hops max, 60 byte packets
 1  _gateway (192.168.1.1)  0.853 ms  0.813 ms  1.066 ms
 2  174.99.0.1 (174.99.0.1)  9.297 ms  9.281 ms  9.266 ms
 3  cpe-174-111-105-240.triad.res.rr.com (174.111.105.240)  9.954 ms  10.057 ms  15.873 ms
 4  cpe-024-025-063-012.ec.res.rr.com (24.25.63.12)  16.444 ms  16.428 ms  16.412 ms
 5  be31.drhmncev01r.southeast.rr.com (24.93.64.184)  22.170 ms  15.788 ms  21.292 ms
 6  be1.drhmncev02r.southeast.rr.com (24.93.64.185)  16.344 ms  15.496 ms  15.467 ms
 7  * * cpe-024-025-062-001.ec.res.rr.com (24.25.62.1)  16.504 ms
 8  cpe-024-025-041-019.ec.res.rr.com (24.25.41.19)  14.787 ms  14.694 ms  18.944 ms
 9  rrcs-98-101-20-25.midsouth.biz.rr.com (98.101.20.25)  18.730 ms  18.806 ms  18.898 ms
10  rrcs-96-10-0-254.se.biz.rr.com (96.10.0.254)  20.755 ms  19.584 ms  20.827 ms
11  ncsu-gw-1-to-rtp-gw.ncren.net (128.109.18.110)  21.013 ms  20.994 ms  20.978 ms
12  152.1.6.69 (152.1.6.69)  20.963 ms 152.1.6.253 (152.1.6.253)  13.847 ms  13.821 ms
13  152.1.6.162 (152.1.6.162)  18.174 ms 152.1.6.142 (152.1.6.142)  18.164 ms  20.210 ms
14  * * *
15  char.csc.ncsu.edu (152.14.93.71)  18.929 ms !X  19.081 ms !X  18.988 ms !X
```

Computerphile video: [13]

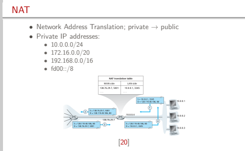
- The only reason I really brought up TTL was so I could tell this fun fact about how traceroute works
- For those of you that don't know, traceroute is a command line tool that tells you the routing information between your computer and someone else's
- It does this by abusing the TTL parameter: by starting with the TTL at 1, the request will always fail at the next hop; failure messages usually include information about the router where it failed
- So by incrementing the TTL after each failure until you get to the intended host, you can get a comprehensive map of a route
- (The asterisks are from routers that don't return information, usually for security reasons)

NAT

- Network Address Translation; private → public
- Private IP addresses:
  - 10.0.0.0/24
  - 172.16.0.0/20
  - 192.168.0.0/16
  - fd00::/8



[20]


NAT

- Network Address Translation; private → public
- Private IP addresses:
  - 10.0.0.0/24
  - 172.16.0.0/20
  - 192.168.0.0/16
  - fd00::/8

[20]

- NAT, or Network Address Translation, is another network-layer protocol
- The most frequently encountered implementation is to allow for private address ranges
- Private IP ranges were originally conceived to help prevent IP exhaustion and aren't allocated to any one group
- Any device on a LAN has to have an IP address, but it doesn't need to be connected to the interent
- That's why most pepole's IP address is just `192.168.X.X` - because it's in the private IP range
- But this creates a problem: with so many devices mapped to the same IP address, how is traffic routed?
- A router with NAT looks like a single host to the outside network
- When the data is sent to the router it acts like a new host and, using its IP address, sends the message out to the internet
- It keeps track of which host sent each request and which port it used to send that request, and when the router gets a response with the right port, it forwards it along to the right local host

ARP

- How to get MAC given IP?
- Address Resolution Protocol
- Similar concept to switch MAC discovery (FF:x6)
- Not really layer 3

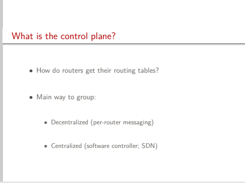
ARP

- How to get MAC given IP?
- Address Resolution Protocol
- Similar concept to switch MAC discovery (FF:x6)
- Not really layer 3

- An astute observer may have realized something: IP packets contain frames, meaning they somehow need to know the MAC address of the destination - this is handled by ARP
- ARP stands for address resolution protocol, and allows hosts to find MAC addresses given the IP address
- The way ARP works is very similar to the way switches get MAC addresses: a message is sent with the broadcast MAC address (FF:x6) to all devices on the network; the one with the matching IP will respond with its MAC
- The question does lead to something that's a bit confusing, though: to send a message to a computer on another network, you don't actually need that computer's MAC address (in fact, you wouldn't even be able to do anything with it); what you do need is your primary router's MAC address
- This protocol isn't *really* a layer 3 or layer 2 protocol; most people call it a 2, but since it uses IP addresses I've decided to group it with the layer 3 protocols

What is the control plane?

- How do routers get their routing tables?
- Main way to group:
  - Decentralized (per-router messaging)
  - Centralized (software controller; SDN)



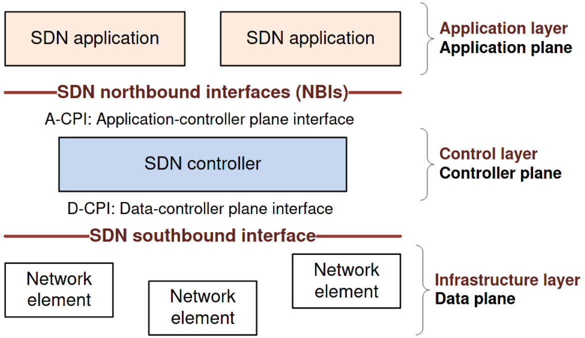
What is the control plane?

- How do routers get their routing tables?
- Main way to group:
  - Decentralized (per-router messaging)
  - Centralized (software controller; SDN)

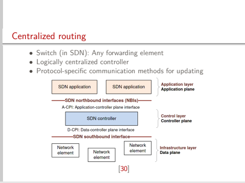
- The data plane is responsible for forwarding packets along
- Question: how do the routers get their routing tables?
- There are many ways to categorize control plane algorithms, but the broadest way is how the data is communicated: either on a per-router basis or from some centralized controller
- The centralized method, sometimes called SDN, or software defined networking, is becoming increasingly popular

Centralized routing

- Switch (in SDN): Any forwarding element
- Logically centralized controller
- Protocol-specific communication methods for updating

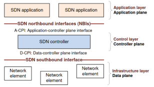


[30]



Centralized routing

- Switch (in SDN): Any forwarding element
- Logically centralized controller
- Protocol-specific communication methods for updating

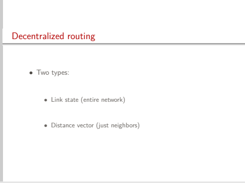


[30]

- I'll start by talking about centralized routing protocols just because I think they're a little easier to understand
- As a bit of a terminology note: in a lot of discussion on SDN, any device responsible for forwarding data is called a switch
- The gist of it is that every router communicates with a "logically centralized" controller, and that server sets each router's forwarding table appropriately based on the information sent to it
- When I say logically centralized, this basically means that the controller can physically be in several different places, but each of the individual components communicate with each other and act as if they were a single component
- This data is usually communicated by having switches send protocol-specific messages, which are then received by the controller, which then sends out another message dictating how to update each switch

Decentralized routing

- Two types:
    - Link state (entire network)
    - Distance vector (just neighbors)

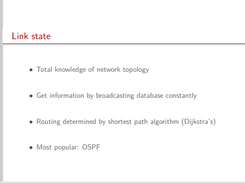

Decentralized routing

- Two types:
  - Link state (entire network)
  - Distance vector (just neighbors)

- Decentralized protocols are much more interesting, in my opinion
- There are two main categories: link state and distance vector
- Link state algorithms work with information of the entire network, while distance vector algorithms work with information just from its neighbors

Link state

- Total knowledge of network topology
- Get information by broadcasting database constantly
- Routing determined by shortest path algorithm (Dijkstra's)
- Most popular: OSPF



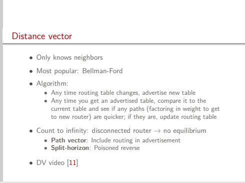
Link state

- Total knowledge of network topology
- Get information by broadcasting database constantly
- Routing determined by shortest path algorithm (Dijkstra's)
- Most popular: OSPF

- Link state algorithms, in general, work by broadcasting lots of information to each node
- The first thing a router needs to know is the routers it's connected to
- When a router is added to a network, it broadcasts to other routers that it's present with a "Hello" message; connected routers will respond, which the router will use to construct its table
- Routers on the network periodically will broadcast out their entire database to surrounding routers; these broadcasts can be used to evaluate the current state of the network, e.g. how quickly they get a response, how busy links are, if a link has gone down, etc.
- After updating their topological database, routers will then use a shortest path algorithm, like Dijkstra's, to determine which route to use for which prefix
- Almost like SDN but all run locally on routers
- The current most popular implementation is OSPF, which stands for Open Shortest Path First

Distance vector

- Only knows neighbors
- Most popular: Bellman-Ford
- Algorithm:
  - Any time routing table changes, advertise new table
  - Any time you get an advertised table, compare it to the current table and see if any paths (factoring in weight to get to new router) are quicker; if they are, update routing table
- Count to infinity: disconnected router $\rightarrow$ no equilibrium
  - **Path vector**: Include routing in advertisement
  - **Split-horizon**: Poisoned reverse
- DV video [11]



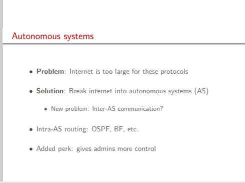
Distance vector

- Only knows neighbors
- Most popular: Bellman-Ford
- Algorithm:
  - Any time routing table changes, advertise new table
  - Any time you get an advertised table, compare it to the current table and see if any paths (factoring in weight to get to new router) are quicker; if they are, update routing table
- Count to infinity: disconnected router $\rightarrow$ no equilibrium
  - **Path vector**: Include routing in advertisement
  - **Split-horizon**: Poisoned reverse
- DV video [11]

- Unlike LS algorithms, which have total knowledge of the entire network, DV algorithms only have knowledge of their immediate neighbors
- Most popular implementation is Bellman-Ford
- People like to describe these algorithms as trying to "one-up" the other routers: each time a router's routing table changes, it will advertise to all of its neighbors its routing table
- When its neighbors receive this message, they'll compare the weights at each step (considering the weight to get to that router) and see if going through that router is more efficient than their current route; if it is, they'll update their routing table and broadcast the change
- This cycle will continue until the entire network has reached the optimal point
- This implementation is very clever, and pretty efficient, but it does have one problem: the count to infinity problem
- Basically, if a router that is only connected to the network by a single router goes down, the network will never reach a steady state since each router just assumes the other ones have a more route to it
- This can be solved or partially solved in a few ways: either by including the routing information in the advertisements, which is known as path vector, or by not advertising connections to direct neighbors on your current path, which is called split horizon
- I won't get too much into split-horizon and the problems it faces, but it does this through something called a poisoned reverse
- Weights can be calculated based on bandwidth, etc., but are usually determined by administrators
- This is something that really needs a visualization to understand, so I recommend checking out this video by Computerphile on DV

Autonomous systems

- **Problem**: Internet is too large for these protocols
- **Solution**: Break internet into autonomous systems (AS)
  - New problem: Inter-AS communication?
- Intra-AS routing: OSPF, BF, etc.
- Added perk: gives admins more control



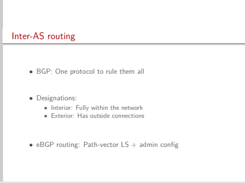
Autonomous systems

- **Problem**: Internet is too large for these protocols
- **Solution**: Break internet into autonomous systems (AS)
  - New problem: Inter-AS communication?
- Intra-AS routing: OSPF, BF, etc.
- Added perk: gives admins more control

- You might've been thinking, especially after hearing about these decentralized protocols, that these seem a bit impractical for very large networks
- And you'd be right: the internet is huge; there's no way we could ever reach convergence or any semblance of unified routing across the entire internet; by the time information propagated all the way across it would be out of date
- For that reason, the internet is divided into whare are called AS, or autonomous systems
- This does create a new problem, which is how to communicate between these AS; we'll look at this problem soon
- All the routing protocols we've looked at so far are called intra-AS routing protocols, since they working within a single AS
- These also have the added benefit of allowing network administrators to have more control over their individual networks
- This is important, because some networks may have better data rates than others, allowing administrators to save a bit of money

Inter-AS routing

- BGP: One protocol to rule them all
- Designations:
    - Interior: Fully within the network
    - Exterior: Has outside connections
- eBGP routing: Path-vector LS + admin config

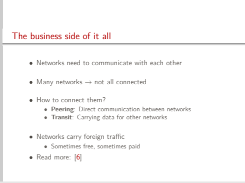

Inter-AS routing

- BGP: One protocol to rule them all
- Designations:
  - Interior: Fully within the network
  - Exterior: Has outside connections
- eBGP routing: Path-vector LS + admin config

- To communicate between AS effectively, one protocol is used; that protocol is called BGP, for border gateway protocol
- BGP divides routers within an AS into two categories: interior and exterior
- Somewhat confusingly, exterior routers are still inside the given AS; the designation just means that they can communicate outside the AS
- Intra-AS protocols communicate changes only within the AS among interior routers; if the change affects external routing, the exterior routers will communicate the change as needed
- I won't go into how it works here, but basically eBGP routing uses path-vector LS + some administrative configuration

The business side of it all

- Networks need to communicate with each other
- Many networks $\rightarrow$ not all connected
- How to connect them?
  - **Peering**: Direct communication between networks
  - **Transit**: Carrying data for other networks
- Networks carry foreign traffic
  - Sometimes free, sometimes paid
- Read more: [6]


The business side of it all

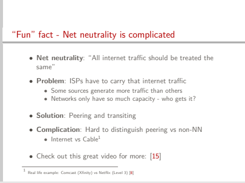
- Networks need to communicate with each other
- Many networks → not all connected
- How to connect them?
  - **Peering**: Direct communication between networks
  - **Transit**: Carrying data for other networks
- Networks carry foreign traffic
  - Sometimes free, sometimes paid
- Read more: [6]

- The fact that Autonomous Systems, usually referred to simply as "networks," must communicate with each other, leads to some interesting cases of fighting amongst these companies
- The main thing to understand is that there are many many networks, and it's impossible for most of them to be connected directly to each other
- In the case where you're requesting data from an external network, it's very likely one or more totally independent networks will have to carry the traffic from your request
- This kind of cooperation involves complicated contracts, agreements, and arrangements, called either peering or transiting
- Peering is when two networks directly share with each other, while transiting is when an intermediate network has to carry data
- Usually, if the amount of data two networks exchange is roughly equivalent, these networks will agree to trade data for free, as it's mutually beneficial, as it provides their clients access to more services
- Sometimes, though, companies will have to pay to access another network's content, or pay for their content to be sent via that network

"Fun" fact - Net neutrality is complicated

- **Net neutrality**: "All internet traffic should be treated the same"
- **Problem**: ISPs have to carry that internet traffic
  - Some sources generate more traffic than others
  - Networks only have so much capacity - who gets it?
- **Solution**: Peering and transiting
- **Complication**: Hard to distinguish peering vs non-NN
  - Internet vs Cable[1]
- Check out this great video for more: [15]

[1] Real life example: Comcast (Xfinity) vs Netflix (Level 3) [8]



“Fun” fact - Net neutrality is complicated

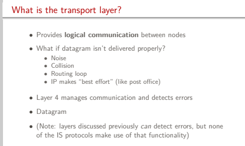
- **Net neutrality**: “All internet traffic should be treated the same”
- **Problem**: ISPs have to carry that internet traffic
  - Some sources generate more traffic than others
  - Networks only have so much capacity - who gets it?
- **Solution**: Peering and transiting
- **Complication**: Hard to distinguish peering vs non-NN
  - Internet vs Cable[1]
- Check out this great video for more: [15]

[1] Real life example: Comcast (Xfinity) vs Netflix (Level 3) [8]

- This is probably the least fun fact possible, but it is an interesting discussion nonetheless
- What is net neutrality? The easiest, most simple definition of net neutrality is that all internet traffic should be treated the same
- This definition seems fine from a consumer’s POV, but runs into a problem when viewed from an ISP’s perspective, because they need to carry all this traffic
- There are two main problems that networks run into: networks have a finite amount of capacity they can provide, and they need to decide how to allocate that capacity if it’s exceeded
- Since some sources generate more traffic than others, it only seems fair (from the network’s POV) to either restrict the amount of data sent, or request more to serve more - this is typically handled as part of the peering/transiting negotiations
- One major complication, however, is how traffic-intensive applications, like YouTube and Netflix, directly conflict with traditional cable companies (who are the primary ISPs), so often it can be hard to distinguish between an ISP being hard to work with “because of data usage” vs defending their market

What is the transport layer?

- Provides **logical communication** between nodes
- What if datagram isn't delivered properly?
    - Noise
    - Collision
    - Routing loop
    - IP makes "best effort" (like post office)
- Layer 4 manages communication and detects errors
- Datagram
- (Note: layers discussed previously *can* detect errors, but none of the IS protocols make use of that functionality)

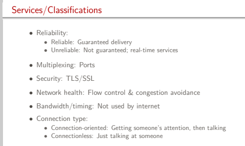

What is the transport layer?

- Provides **logical communication** between nodes
- What if datagram isn't delivered properly?
  - Noise
  - Collision
  - Routing loop
  - IP makes "best effort" (like post office)
- Layer 4 manages communication and detects errors
- Datagram
- (Note: layers discussed previously *can* detect errors, but none of the IS protocols make use of that functionality)

- The transport layer provides logical communication between nodes across the network
- Consider what happens if a datagram isn't delivered correctly?
- There are dozens of reasons why a datagram may not have been delivered properly: too much noise; collision; caught in routing loop
- No matter what, though: if a datagram is lost, it usually needs to be resent
- So far, none of the facilities we've discussed on the internet have any kind of way to even determine if a datagram has been delivered or not; IP services only offer best effort delivery, which means basically that the network will make its best effort to send the data as soon as it can, but makes no promises (like post office - sometimes loses/mis-sends mail)
- That's where the transport layer comes in: it manages the sending and receiving of data to ensure that all the appropriate data is received properly, and in the right order (if applicable)
- When layer 4 encapsulates data, this is called a datagram
- Note that several of the lower levels are capable of detecting that their data wasn't properly delivered; it's just that none of the internet suite protocols make use of them

Services/Classifications

- Reliability:
  - Reliable: Guaranteed delivery
  - Unreliable: Not guaranteed; real-time services
- Multiplexing: Ports
- Security: TLS/SSL
- Network health: Flow control & congestion avoidance
- Bandwidth/timing: Not used by internet
- Connection type:
  - Connection-oriented: Getting someone's attention, then talking
  - Connectionless: Just talking at someone

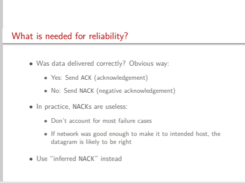

Services/Classifications

- Reliability:
  - Reliable: Guaranteed delivery
  - Unreliable: Not guaranteed; real-time services
- Multiplexing: Ports
- Security: TLS/SSL
- Network health: Flow control & congestion avoidance
- Bandwidth/timing: Not used by internet
- Connection type:
  - Connection-oriented: Getting someone's attention, then talking
  - Connectionless: Just talking at someone

- One way to look at the transport layer is through the different types of services it offers
- The most important one, in my opinion, is the type of reliability you want
- These can be divided into two categories: do you want your data delivery to be guaranteed to be delivered (reliable), or have the possibility that it may not be delivered (unreliable)?
- You may be thinking, "Why would I ever send data if I don't care if it's delivered or not?" But a lot of real-time services, like live-video chatting, use unreliable service
- It makes sense for services like that: if it's late, we don't even want or need it anymore, since it's out of date anyways
- The transport layer also offers multiplexing; in the internet, this is done through the use of ports, which allow multiple connections to run at once on a single host; think of it like an IP/MAC address for a given connection
- Some security is also offered through TLS and SSL, which I'll talk more about later
- Two other things I'll talk more about in a bit are flow control and congestion avoidance, which contribute to the overall network health
- Some TL protocols make promises about the amount of bandwidth allocated, or how soon the data will be sent, but not none of the protocols in the internet do
- One type of classification would be the connection type
- Some TL services need to establish a connection before communication can begin; others don't

What is needed for reliability?

- Was data delivered correctly? Obvious way:
  - Yes: Send ACK (acknowledgement)
  - No: Send NACK (negative acknowledgement)
- In practice, NACKs are useless:
  - Don't account for most failure cases
  - If network was good enough to make it to intended host, the datagram is likely to be right
- Use "inferred NACK" instead


What is needed for reliability?

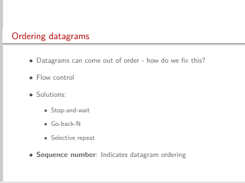
- Was data delivered correctly? Obvious way:
  - Yes: Send ACK (acknowledgement)
  - No: Send NACK (negative acknowledgement)
- In practice, NACKs are useless:
  - Don't account for most failure cases
  - If network was good enough to make it to intended host, the datagram is likely to be right
- Use "inferred NACK" instead

- So what all is needed to guarantee reliable delivery?
- First of all, at the very least you need a way to at least know if the data was delivered successfully or not
- The most obvious way to do this is to send a message signalling that the data was either received properly or not - either a positive acknowledgement that the data was received properly (ACK), or a negative acknowledgement (NACK), meaning the data was received incorrectly
- The only problem is that NACKs are kind of useless in most situations if you think about it:
- Considering the datagrams even makes it to the right host (which means the network has no routing loops and low enough noise), what errors are you even solving?
- So most TL protocols use what's called an "inferred NACK," where, if an ACK isn't received after a certain amount of time, it's just assumed that the data was lost/corrupted

Ordering datagrams

- Datagrams can come out of order - how do we fix this?
- Flow control
- Solutions:
  - Stop-and-wait
  - Go-back-N
  - Selective repeat
- **Sequence number**: Indicates datagram ordering

2021-04-27

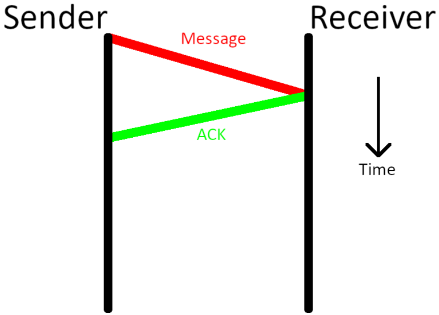

Ordering datagrams

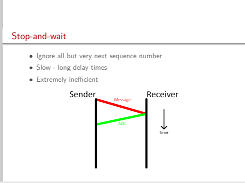
- Datagrams can come out of order - how do we fix this?
- Flow control
- Solutions:
  - Stop-and-wait
  - Go-back-N
  - Selective repeat
- **Sequence number**: Indicates datagram ordering

- Because data sent over the internet can go one of many different routes, there's no guarantee that it will come in the right order
- Flow control is the name of the TL concept responsible for ordering datagrams
- In order to account for that, the receiver needs to ensure that the data comes in the right order
- There are a number of ways to do this: stop and wait, go-back-n, and selective repeat
- Each solution has different pros and cons, but requires a way to keep track of the order datagrams were sent in; this is usually done by identifying each datagram with a sequence number

Stop-and-wait

- Ignore all but very next sequence number
- Slow - long delay times
- Extremely inefficient



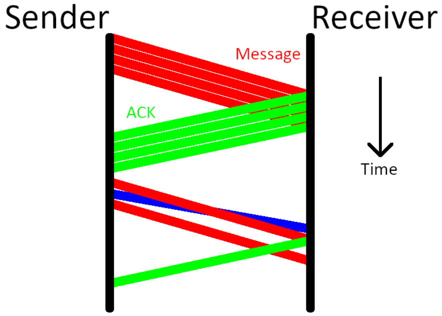

Stop-and-wait

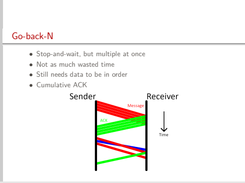
- Ignore all but very next sequence number
- Slow - long delay times
- Extremely inefficient

- The simplest, but also least efficient, of the order-ensuring techniques is for the receiver to ignore all datagrams except the one with the very next sequence number it's waiting for
- The major problem with this implementation is that it's very slow, especially for long connections
- It's also very inefficient: all the resources spend most of their time waiting on responses

Go-back-N

- Stop-and-wait, but multiple at once
- Not as much wasted time
- Still needs data to be in order
- Cumulative ACK





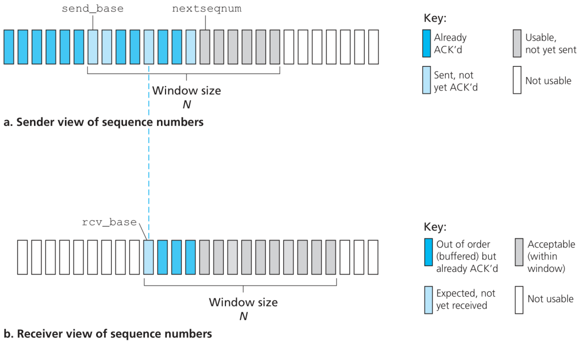
Go-back-N

- Stop-and-wait, but multiple at once
- Not as much wasted time
- Still needs data to be in order
- Cumulative ACK

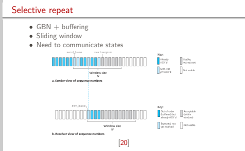
- Go-back-N is a bit more sophisticated: like stop-and-wait except multiple messages are sent at once
- Datagrams still need to be ACKd in order, but at least now the receiver isn't spending so much time sitting around idle
- Here in the picture you can see an example of how GBN works: in this case, 4 datagrams were sent out at a time; at the first time all 4 were received correctly and in order
- The second time, however, the second datagram arrived before the first, preventing all following datagrams from being ACKd, even if they arrived in order
- What would happen in this case is that the sender would transmit 4 datagrams: the 3 that were rejected before, and 1 new one
- One important aspect of GBN is that it offers what's called a cumulative ack: when it sends an ACK for a given sequence number, that number and all that came before it have been received correctly

Selective repeat

- GBN + buffering
- Sliding window
- Need to communicate states


send_base
nextseqnum
Window size
N
a. Sender view of sequence numbers
Key:
Already ACK'd
Sent, not yet ACK'd
Usable, not yet sent
Not usable
rcv_base
Window size
N
b. Receiver view of sequence numbers
Key:
Out of order (buffered) but already ACK'd
Expected, not yet received
Acceptable (within window)
Not usable

[20]



Selective repeat

- GBN + buffering
- Sliding window
- Need to communicate states

a. Sender view of sequence numbers

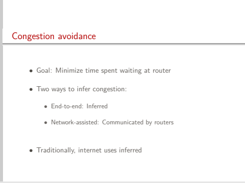
b. Receiver view of sequence numbers

[20]

- Finally, there's selective repeat, which is like GBN if it could buffer datagrams received in the wrong order
- This one is the most useful, but also the hardest to understand, because they sender and receiver can be in such fundamentally different states
- The receiver will accept datagrams within its sliding window range, but will deny datagrams outside of that window
- The receiver's window will slide up whenever the lowest previously unacknowledged datagram is received
- In order to keep in sync, the sender and receiver need to have some state communication
- The window size and base of each host can be included in each message to keep in sync

Congestion avoidance

- Goal: Minimize time spent waiting at router
- Two ways to infer congestion:
    - End-to-end: Inferred
    - Network-assisted: Communicated by routers
- Traditionally, internet uses inferred



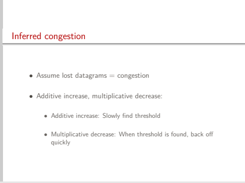
Congestion avoidance

- Goal: Minimize time spent waiting at router
- Two ways to infer congestion:
  - End-to-end: Inferred
  - Network-assisted: Communicated by routers
- Traditionally, internet uses inferred

- The goal of congestion avoidance is to minimize the amount of time a datagram sends waiting at a router
- There are two main ways to determine congestion: end-to-end, where total network congestion has to be inferred, or network-assisted, where routers can communicate congestion
- The internet has traditionally relied on inferred congestion, though network-assisted is becoming more popular

Inferred congestion

- Assume lost datagrams = congestion
- Additive increase, multiplicative decrease:
  - Additive increase: Slowly find threshold
  - Multiplicative decrease: When threshold is found, back off quickly



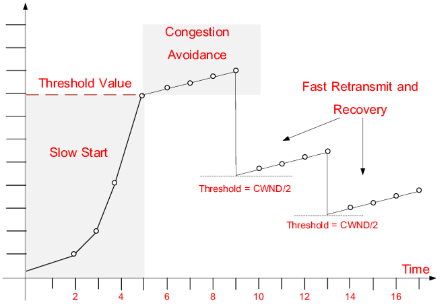
Inferred congestion

- Assume lost datagrams = congestion
- Additive increase, multiplicative decrease:
  - Additive increase: Slowly find threshold
  - Multiplicative decrease: When threshold is found, back off quickly

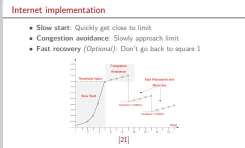
- In inferred congestion avoidance, the internet has to guess the state of the network based only on the minimal signals it has received, and tries to act accordingly
- This is usually done through lost datagrams; when datagrams are lost, it is assumed they were lost because of congestion
- The primary method used is called additive increase, multiplicative decrease, because of how it works
- The idea is to be conservative and reactionary: by slowing increasing the amount of data sent, the network won't be as “swingy”; by decreasing multiplicatively the router backs off quickly, ensuring congestions ends ASAP

Internet implementation

- **Slow start**: Quickly get close to limit
- **Congestion avoidance**: Slowly approach limit
- **Fast recovery** *(Optional)*: Don't go back to square 1

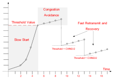


[21]



Internet implementation

- **Slow start**: Quickly get close to limit
- **Congestion avoidance**: Slowly approach limit
- **Fast recovery** *(Optional)*: Don't go back to square 1

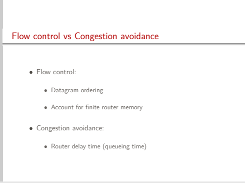


[21]

- The actual implementation used on the internet is a bit more sophisticated than that, though
- The state it starts off in is slow start

Flow control vs Congestion avoidance

- Flow control:
  - Datagram ordering
  - Account for finite router memory
- Congestion avoidance:
  - Router delay time (queueing time)



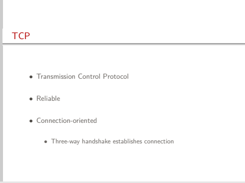
Flow control vs Congestion avoidance

- Flow control:
  - Datagram ordering
  - Account for finite router memory
- Congestion avoidance:
  - Router delay time (queueing time)

- Flow control and congestion avoidance are two oft-confused services in the TL that aim to improve the overall health and performance of the network
- The goal of flow control is to keep any router from getting so many datagrams at once that it loses some of them, either because it can't process all them at once, or because it's run out of room
- Congestion avoidance is a similar, but solves a slightly different problem: even if a router had unlimited memory, if datagrams are coming faster than it can put them out, a backup will occur

TCP

- Transmission Control Protocol
- Reliable
- Connection-oriented
    - Three-way handshake establishes connection

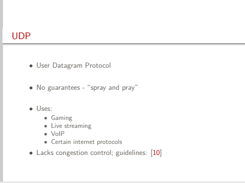

TCP

- Transmission Control Protocol
- Reliable
- Connection-oriented
  - Three-way handshake establishes connection

- TCP, short for Transmission Control Protocol, is the primary reliable protocol for the internet
- It's also connection-oriented. This means that, before any data can be sent over it, some kind of setup process must occur to ensure that the two hosts can actually communicate
- For TCP, this setup process is known as a three-way handshake, which I won't go into here

UDP

- User Datagram Protocol
- No guarantees - “spray and pray”
- Uses:
  - Gaming
  - Live streaming
  - VoIP
  - Certain internet protocols
- Lacks congestion control; guidelines: [10]

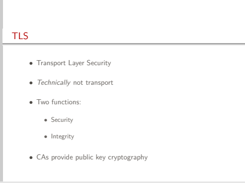

UDP

- User Datagram Protocol
- No guarantees - "spray and pray"
- Uses:
  - Gaming
  - Live streaming
  - VoIP
  - Certain internet protocols
- Lacks congestion control; guidelines: [10]

- UDP, or the User Datagram Protocol, is the other main protocol used on the internet
- Unlike TCP, UDP provides no guarantees about how, when, or even *if* data will arrive at all
- This means it lacks many of the attributes described before, like frame ordering and acknowledgements
- This may sound like it's way worse than TCP, but it has its uses:
- Because UDP is so much simpler than TCP, it makes it easier and "lower-cost" to implement than TCP connections, making it ideal for scenarios where either response time is important (gaming), or where it's okay if some packets are dropped (live streaming and VoIP)
- Additionally, its lower overhead makes it ideal for certain types of requests - more on that later
- Notably, UDP lacks native congestion control, meaning it's possible for UDP to "hog" the network while TCP continues dialing back its windows
- Based on my research, there's no enforced standard for managing UDP traffic, but it's in everybody's best interest to follow the established guidelines for best service

TLS

- Transport Layer Security
- *Technically* not transport
- Two functions:
  - Security
  - Integrity
- CAs provide public key cryptography



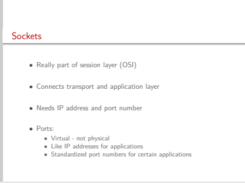
TLS

- Transport Layer Security
- *Technically* not transport
- Two functions:
  - Security
  - Integrity
- CAs provide public key cryptography

- TLS, or transport layer security, is another protocol in the transport layer - kind of
- Because it sits over top of messages in the transport layer, you'd expect it to be in the presentation or application layer, but most applications just treat it like it's part of the transport layer
- TLS provides two main features: data security (in other words, unintended people can't read it), and integrity (in other words, the data received is the same as the data set)
- CAs, or certificate authorities, provide the backbone of TLS' functionality by acting as public key exchanges

Sockets

- Really part of session layer (OSI)
- Connects transport and application layer
- Needs IP address and port number
- Ports:
  - Virtual - not physical
  - Like IP addresses for applications
  - Standardized port numbers for certain applications


Sockets

- Really part of session layer (OSI)
- Connects transport and application layer
- Needs IP address and port number
- Ports:
  - Virtual - not physical
  - Like IP addresses for applications
  - Standardized port numbers for certain applications

- One last thing I should talk about before I go on are sockets
- While most people consider sockets part of the session layer of the OSI model, I'm grouping them with the transport layer here since I'm not planning to talk about the session layer
- In basic terms, sockets are basically what connect the transport and application layer
- Sockets are formed using an IP address and a port number
- That may make you wonder - what are port numbers?
- Ports (in this context) are entirely virtual
- You can think of them kind of like IP addresses for specific applications
- Certain ports have been standardized to enable easier communication, otherwise you'd need to guess every port

(Attempted) Humor

I'd tell you a joke about UDP... but you might not get it

(Attempted) Humor

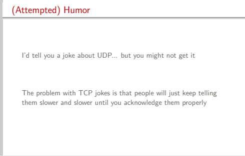
I'd tell you a joke about UDP... but you might not get it

- I couldn't think of a great fun fact for the transport layer, so instead you'll have to settle for some bad jokes
- I'd tell you a joke about UDP
- But you might not get it
- (Because UDP makes no guarantees on delivery)

(Attempted) Humor

I'd tell you a joke about UDP... but you might not get it

The problem with TCP jokes is that people will just keep telling them slower and slower until you acknowledge them properly



(Attempted) Humor

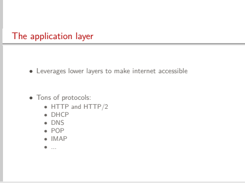
I'd tell you a joke about UDP... but you might not get it

The problem with TCP jokes is that people will just keep telling them slower and slower until you acknowledge them properly

- The problem with TCP jokes is that people will just keep telling them slower and slower until you acknowledge them properly
- (Because of AIMD)

The application layer

- Leverages lower layers to make internet accessible
- Tons of protocols:
  - HTTP and HTTP/2
  - DHCP
  - DNS
  - POP
  - IMAP
  - ...



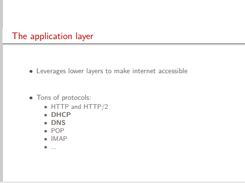
The application layer

- Leverages lower layers to make internet accessible
- Tons of protocols:
  - HTTP and HTTP/2
  - DHCP
  - DNS
  - POP
  - IMAP
  - ...

- The application layer is the last layer that I'm going to talk about in this presentation
- Basically, what it does is it ties everything together to make using the internet easier

The application layer

- Leverages lower layers to make internet accessible
- Tons of protocols:
  - HTTP and HTTP/2
  - **DHCP**
  - **DNS**
  - POP
  - IMAP
  - ...

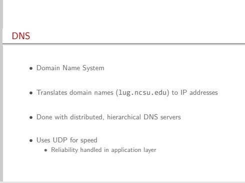

The application layer

- Leverages lower layers to make internet accessible
- Tons of protocols:
  - HTTP and HTTP/2
  - **DHCP**
  - **DNS**
  - POP
  - IMAP
  - ...

- There are tons of protocols in use in the application layer, but I've decided to only talk about a few that I think are both interesting and (potentially) useful to know about
- Those two specifically being DHCP and DNS

DNS

- Domain Name System
- Translates domain names (`lug.ncsu.edu`) to IP addresses
- Done with distributed, hierarchical DNS servers
- Uses UDP for speed
    - Reliability handled in application layer



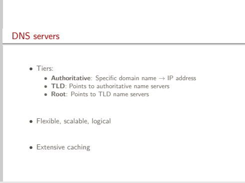
DNS

- Domain Name System
- Translates domain names (`lug.ncsu.edu`) to IP addresses
- Done with distributed, hierarchical DNS servers
- Uses UDP for speed
  - Reliability handled in application layer

- DNS, or the Domain Name System, is the protocol responsible for translating domain names, like `www.google.com` or `lug.ncsu.edu` to the corresponding IP address
- This is accomplished through the use of a series of distributed, hierarchical DNS servers scattered around the world that are queried as part of the DNS protocol
- DNS requests are sent as UDP packets because they're faster
- DNS requests need to be done quickly; this means that UDP is ideal for these messages, since TCP takes a long time (relatively) to establish a connection
- Obviously, DNS still needs reliability, though; this is handled by the application layer: it simply resends UDP requests until it gets a response

DNS servers

- Tiers:
  - **Authoritative**: Specific domain name $\rightarrow$ IP address
  - **TLD**: Points to authoritative name servers
  - **Root**: Points to TLD name servers
- Flexible, scalable, logical
- Extensive caching


DNS servers

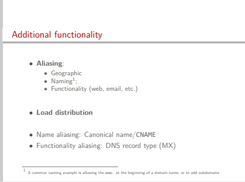
- Tiers:
  - **Authoritative**: Specific domain name → IP address
  - **TLD**: Points to authoritative name servers
  - **Root**: Points to TLD name servers
- Flexible, scalable, logical
- Extensive caching

- Earlier I said that DNS servers are hierarchical and distributed… but what exactly does that mean?
- There are three main tiers of DNS servers: at the lowest level there are authoritative DNS servers, which associate specific domain names with specific IP addresses
- But how does your computer know how to access these authoritative name servers? It usually gets this info from a TLD, or top level domain server, which has IP address information for many authoritative name servers
- Top level domains are the last part of a domain name, for instance .com, .org, etc
- Just like how the IP addresses of the authoritative name servers are known by the TLD name servers, the TLD IP addresses are given by the root server
- The organization of this system was chosen because it allows for the greatest amount of flexibility and resilience for the system
- One main feature of DNS is its use of caching: each name server, as well as your own computer, will keep a cache of results, allowing it to bypass certain requests, improving speed and reducing traffic

Additional functionality

- **Aliasing**:
  - Geographic
  - Naming[1];
  - Functionality (web, email, etc.)
- **Load distribution**
- Name aliasing: Canonical name/`CNAME`
- Functionality aliasing: DNS record type (MX)

[1] A common naming example is allowing the `www.` at the beginning of a domain name, or to add subdomains



Additional functionality

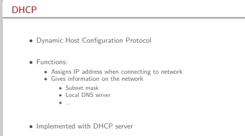
- **Aliasing**:
  - Geographic
  - Naming[1];
  - Functionality (web, email, etc.)
- **Load distribution**
- Name aliasing: Canonical name/`CNAME`
- Functionality aliasing: DNS record type (MX)

[1] A common naming example is allowing the `www.` at the beginning of a domain name, or to add subdomains

- DNS covers more than just simply associating domain names to IP address: it also handles aliasing and load distribution
- There are two main types of aliasing: geographic aliasing, i.e. allowing a website to serve you the domain closest to you, giving faster load times; naming aliasing, i.e. making it so that users can access the same IP using multiple domain names; functionality aliasing, i.e. allowing you to serve your website, mail server, and more all from a single domain name
- The other main function DNS provides is handling load distribution: it's possible for the DNS record to associate a specific domain name to *multiple* IP addresses, allowing for backups in case one is slow or overused

DHCP

- Dynamic Host Configuration Protocol
- Functions:
  - Assigns IP address when connecting to network
  - Gives information on the network
    - Subnet mask
    - Local DNS server
    - ...
- Implemented with DHCP server

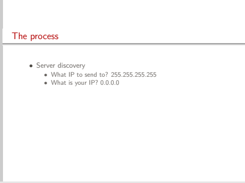

DHCP

- Dynamic Host Configuration Protocol
- Functions:
  - Assigns IP address when connecting to network
  - Gives information on the network
    - Subnet mask
    - Local DNS server
    - ...
- Implemented with DHCP server

- Another protocol that you've likely interacted with without realizing is DHCP, or the Dynamic Host Configuration Protocol
- This protocol is what allows you to easily connect to a new network quickly and painlessly by assigning new hosts on a network an IP address, making it so you don't have to worry about accidentally choosing the same IP address as someone
- As part of the IP assignment process, it also tells your host information about the network, such as its subnet mask, the local DNS server, etc.
- DHCP is implemented with a DHCP server; on most home networks, this server is actually built in to your router, but on more sophisticated networks, specific servers exist to handle DHCP

The process

- Server discovery
    - What IP to send to? 255.255.255.255
    - What is your IP? 0.0.0.0


The process

- Server discovery
  - What IP to send to? 255.255.255.255
  - What is your IP? 0.0.0.0

- I won't go into too much depth on how this whole process works, but I do want to explain a little about how it works
- The first step is for the new host to send a DHCP request to the DHCP server - but hang on: how does it know where to find the DHCP server?
- The answer: it doesn't, so it needs to ask all the devices on the network "Are you the DHCP server?" using a broadcast IP message (255.255.255.255) and wait for a response
- This poses another tricky question, though: if you remember correctly, IP messages need a sending and return IP address; if DHCP is for getting an IP address, what do you list as the return IP?
- For unassigned IPs, 0.0.0.0 is used

The process

- Server discovery
  - What IP to send to? 255.255.255.255
  - What is your IP? 0.0.0.0
- Server offers
- Request
- Acknowledgement

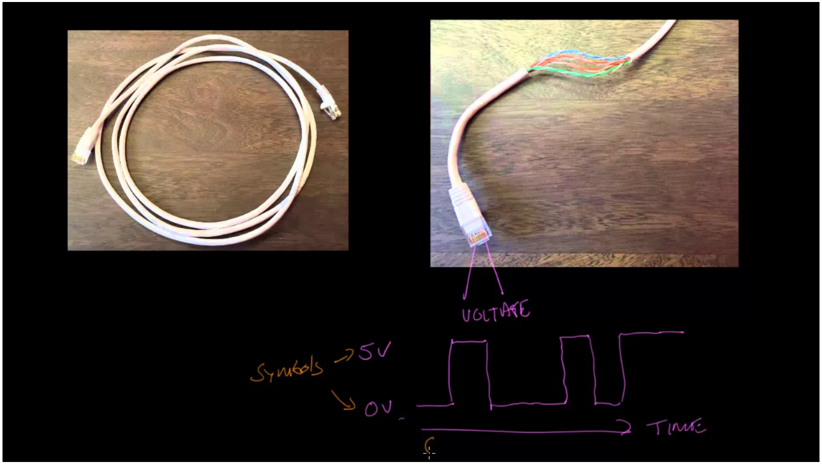
The process

- Server discovery
  - What IP to send to? 255.255.255.255
  - What is your IP? 0.0.0.0
- Server offers
- Request
- Acknowledgement

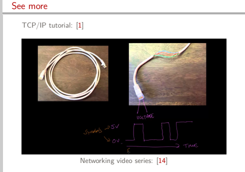
- Okay, so now that your DHCP discovery request is sent, the DHCP server(s) on the network will respond
- If you get multiple responses, you can either look at their offers and choose the best one, choose the first one, etc.
- You send a request to the DHCP server you've chosen, which will then send an acknowledgement
- The acknowledgement is important, as it's possible the DHCP server offered an IP address that was no longer available by the time you requested it

See more

TCP/IP tutorial: [1]



Networking video series: [14]


See more

TCP/IP tutorial: [1]

Networking video series: [14]

- There's just way too much information for me to go super in-depth on all of it
- Here are some good references that you can use if you'd like to learn more

References I

Text

[1] A TCP/IP Tutorial
https://tools.ietf.org/html/rfc1180

[2] Can every grain of sand be addressed in IPv6?
https://skeptics.stackexchange.com/a/4509

[3] Communication and Networking - Forward Error Correction Basics
https://spinlab.wpi.edu/courses/ece2305_2014/forward_error_correction.pdf

[4] Dire Wolf User Guide
https://raw.githubusercontent.com/wb2osz/direwolf/dev/doc/User-Guide.pdf

[5] Gravitational time dilation
https://en.wikipedia.org/wiki/Gravitational_time_dilation

[6] How Peering Agreements Affect Netflix, YouTube, and the Entire Internet
https://www.howtogeek.com/182905/
how-peering-agreements-affect-netflix-youtube-and-the-entire-internet/

[7] Leap second
https://en.wikipedia.org/wiki/Leap_second

[8] Level 3 takes spat with Comcast public
https://www.cnet.com/news/level-3-takes-spat-with-comcast-public/

[9] Photophone
https://en.wikipedia.org/wiki/Photophone

[10] UDP Usage Guidelines
https://tools.ietf.org/html/bcp145

References II

Videos/Playlists

[11] Distance Vector Algorithm (Bellman-Ford) - Computerphile
https://www.youtube.com/watch?v=NdKcjKfJocE

[12] Error detection - Ben Eater
https://www.youtube.com/playlist?list=PLowKtXNTBypFWff2QjXCWuSfJDWcvE0Vm

[13] How Traceroute Works - Computerphile
https://www.youtube.com/watch?v=75yKT3OuE44

[14] Networking tutorial - Ben Eater
https://www.youtube.com/playlist?list=PLowKtXNTBypH19whXTVoG3oKSuOcw_XeW

[15] The case against Net Neutrality? - Ben Eater
https://www.youtube.com/watch?v=hKD-lBrZ_Gg

[16] UTF-8 - Papers We Love San Diego
https://www.youtube.com/watch?v=mhvaeHoIE24&t=1736s

References III

Images

[17] ALOHAnet
https://www.eng.hawaii.edu/about/history/alohanet/

[18] AM-FM
https://www.quora.com/Whats-the-difference-between-amplitude-modulation-and-phase-modulation

[19] Bell's family
https://en.wikipedia.org/wiki/Alexander_Graham_Bell

[20] Computer Networking, 8th Edition - James Kurose / Keith Ross

[21] Congestion avoidance
https://www.researchgate.net/figure/Slow-Start-and-Congestion-Avoidance-in-TCP_fig6_256197047

[22] Digital modulation
https://www.5gtechnologyworld.com/digital-modulation-basics-part-1/

[23] FDM
https://www.watelectronics.com/what-is-multiplexer-and-types/

[24] Frame format
https://www.gatevidyalay.com/ethernet-ethernet-frame-format/

[25] Packet radio
https://opensource.com/article/17/9/packet-radio

[26] Power spectral density
https://en.wikipedia.org/wiki/Minimum-shift_keying

References IV

[27] Routing loop
https://www.computernetworkingnotes.com/ccna-study-guide/routing-loops-explained-with-examples.html

[28] Self-clocking - RTZ
https://en.wikipedia.org/wiki/Self-clocking_signal

[29] Start-stop signalling
https://reviseomatic.org/rOmV4/rOmV4/page/314/Signal_Serial_and_Parallel

[30] Software defined networking
https://opennetworking.org/wp-content/uploads/2013/02/TR_SDN_ARCH_1.0_06062014.pdf

[31] Switch vs hub
https://security.stackexchange.com/a/102097

[32] TDM
http://ecedunia.blogspot.com/2016/03/multiplexing.html