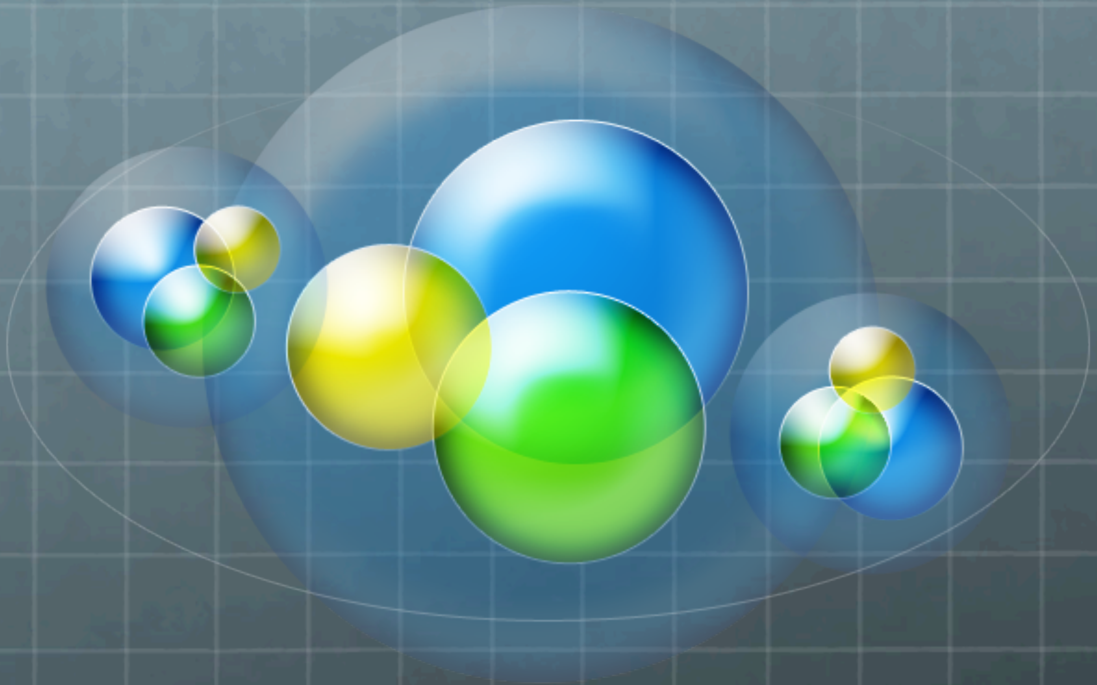


Final Review!

So how’s it all work?

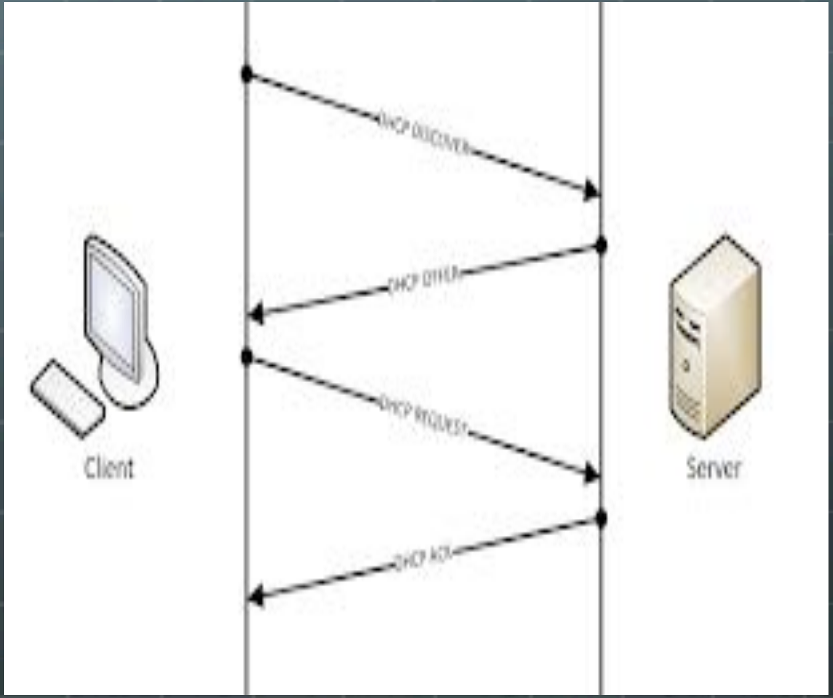
- I boot my machine
- I open my browser and type www.google.com
- The page loads
- What all just happened?

My Computer Booted

- My wifi wants to connect me to a network
- It has to find a wifi network in range
- It has to connect to that Access Point and successfully share a wifi channel/environment
  - MAC and Collision Resolution (RTS, CTS)

My Computer Found the Router

- Now I need an IP Address
  - DHCP Lease for an IP from the router
    - I ask for an IP
    - Router gives an IP
    - I tell it I will use that IP
    - It says Okay



I type google.com

- I want to make a TCP socket to send data there: can I?
  - No, I need an IP:Port to connect a socket, not a name
  - Make a DNS query to some nameserver to give me the IP that goes with google.com (173.194.33.99)
    - How do I know that IP?
    - Ideas?

I make my request

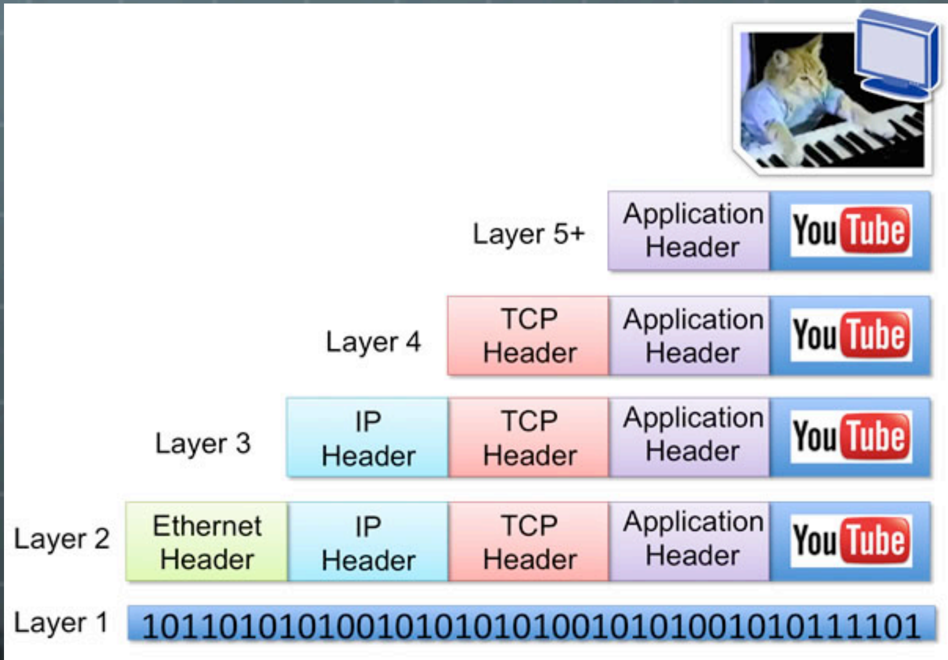
- Formulate a valid HTTP GET request to get 173.194.33.99
- Make a TCP socket that connects out to that IP on port 80.
- My router is NATing, what happens?
  - It maps my IP and port on the LAN side to some port on the WAN side
  - Alters my outgoing packet’s source IP and port accordingly

I’m still making my request

- I’m using wifi to get the router
- The router is connected to a cable-modem via Ethernet
- The cable-modem is connected to my ISP network via coaxial cable
- My ISP connects to 173.194.33.99 via fiber-optic links
- Does any of this matter?
  - Yes and No

Encapsulation

- I’m making a browser application connection
- It’s built on a TCP stream
- The TCP stream using IP routing to go from me to 173.194.33.99
- The IP packet needs a wifi/ethernet/cable/fiber-optic header on it for each hop



Layer 5+ | Application Header | YouTube

Layer 4 | TCP Header | Application Header | YouTube

Layer 3 | IP Header | TCP Header | Application Header | YouTube

Layer 2 | Ethernet Header | IP Header | TCP Header | Application Header | YouTube

Layer 1 | 1011010101001010101010010101001010111101

My connection gets there!

- Have to make a TCP connection before sending any data
- 3-Way TCP handshake of:
  - Syn (x) ->
  - <- SynAck(y, x+1)
  - Ack (y+1) ->
- Now I can finally send my data packet!

My data arrives

- 173.194.33.99 finally gets my HTTP request
- What does it have to do?
- Process request, possibly do server-side computation, then send me the webpage
- It makes the same crazy trip back, including getting retransformed by the NAT router to reach me
- May or may not need to then send more GET requests in response to get other portions of the page

Whew. Glad that all works all the time…

- What if my signal got distorted in the air, or on one of the wires?
  - Error-Correcting Codes
- What if one packet takes a slow route and the next takes a fast route?
  - TCP automatically reorders correctly: UDP not so much
- What if my packet gets dropped somewhere?
  - ACKs ensure delivery
  - Link-Layer ACKs sometimes used for high-loss mediums (wifi) to speed up recovery

What Else Might've Happened?

- My browser already has google.com cached, I just show that
- Some caching proxy is between me and 173.194.33.99 and responds to my request mid-route, rather than forwarding it to the real 173.194.33.99
- I (or my router!) use Tor so I go hopping through three other nodes before exiting to go get 173.194.33.99's page

How did my router get me the data?

- It needs to know who to route to on the wifi-layer to put the correct address on its packet
- Uses ARP to look up my MAC address from my IP address and routes my packets to me by addressing it to my MAC address

Where did the projects fit in?

- Project 0
  - Understanding how to get the basic names for networked items: IP and port
- Project 1
  - Essentially a simple DNS service, letting you specify some destination by a more convenient name and add some useful abstraction
- Project 2
  - Get to know HTTP requests, and how a program can ‘invisibly’ stand in the middle of a network connection
- Project 3
  - Understand routing and the complexities associated with creating routes as well as running multiple streams over independent connections

Other Thoughts?

- What makes network/distributed programming different from standard local programming?
- What are some traps to watch out for?
- Why do we need so many threads all the time?
- Who should be doing computation in a web app?
- How about a peer-to-peer system?