The Basics of Computer Networking

What is a Computer Network?

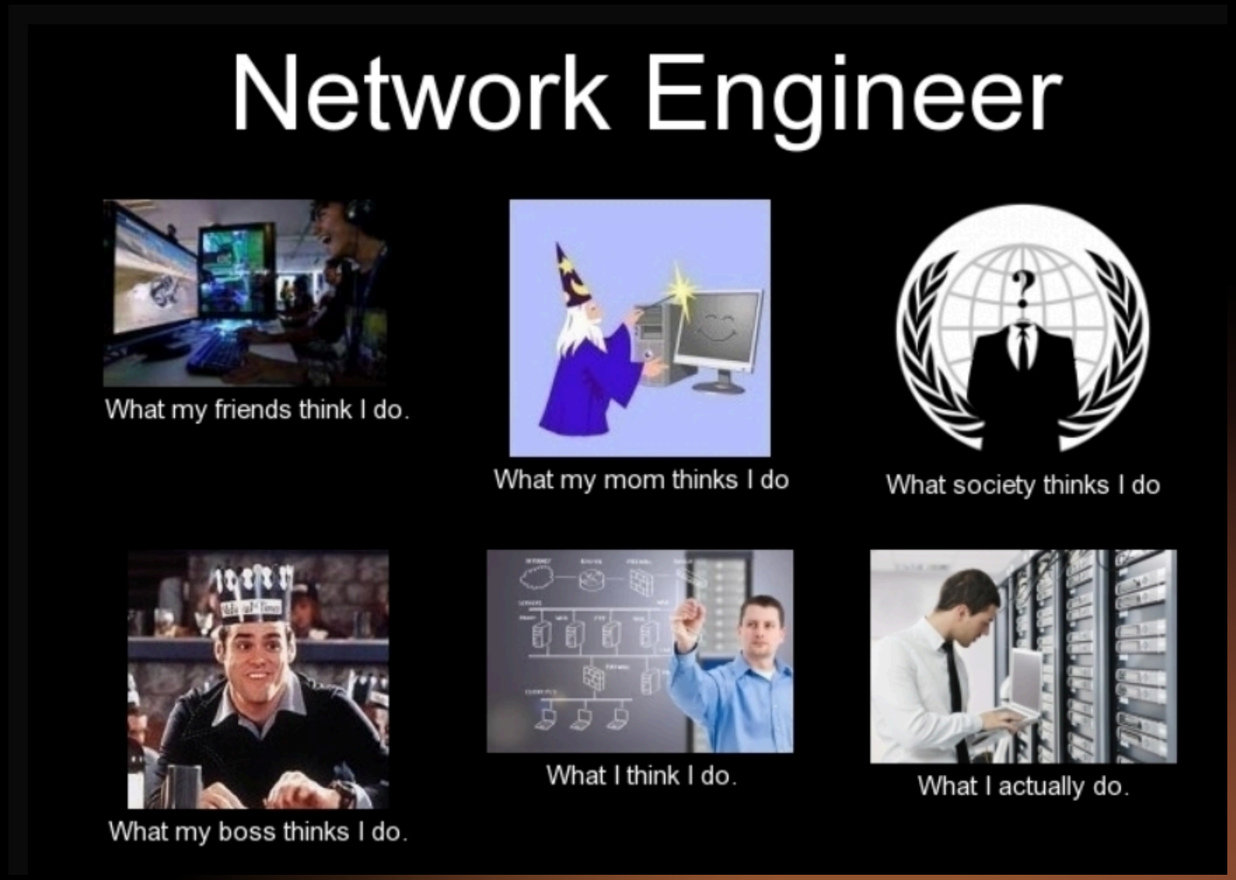
Interconnected Devices

- A network is simply a collection of connected devices that can share resources and data.
- Think of it like a road system connecting different cities.

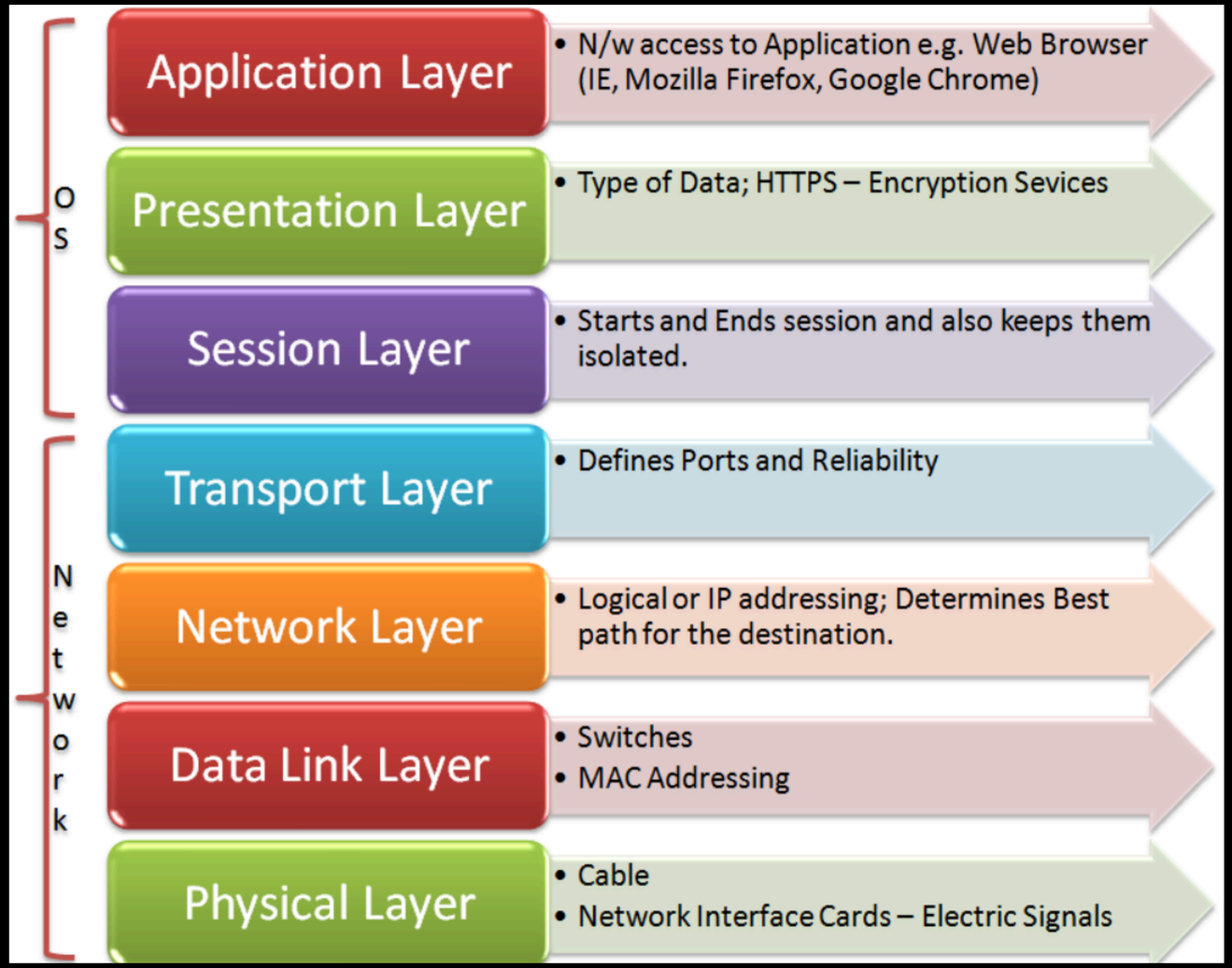
The Internet is the biggest network of all—a global network of computers.

Data Exchange

Networks allow the exchange of data, files, and information between the connected devices, enabling collaboration, resource sharing, and access to online services.



Layers of a Network



The Role of the Transport Layer

Main Job: Provide logical communication between applications running on different hosts.

The Transport Layer (TCP/UDP):

- Responsibility: Process-to-process communication
- What it does: Ensures data gets from a specific application/service on the source device to the correct application/service on the destination device
- Key point: It handles the "last mile" delivery to the right application

Analogy: IP gets a letter to the correct apartment building. The Transport Layer gets the letter to the correct person (or apartment number) inside that building.

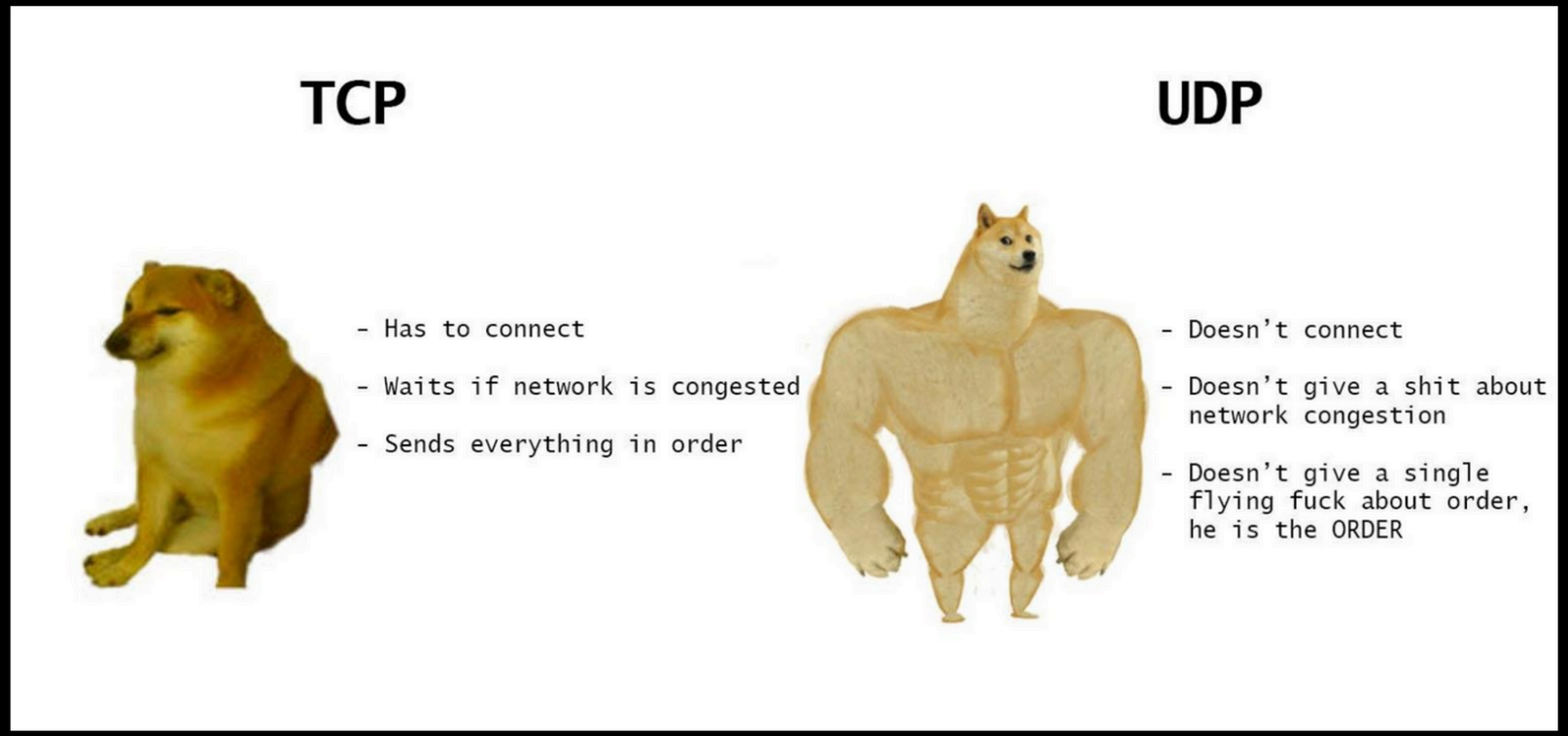
How does it know which program to go to?

Ports!

- A port is just a 16-bit number (0-65535) used to identify a specific application process.
- An IP Address + a Port Number = a unique destination.
- Example Well-Known Ports:
  - HTTP (Websites): Port 80
  - HTTPS (Secure Websites): Port 443
  - FTP (File Transfer): Port 21
  - DNS (Domain Name System): Port 53

Meet the Protocols: TCP & UDP

The Transport Layer has two main "messengers" (protocols) you can choose from.
They have very different ways of getting the job done. Choosing the right one is crucial for application performance.

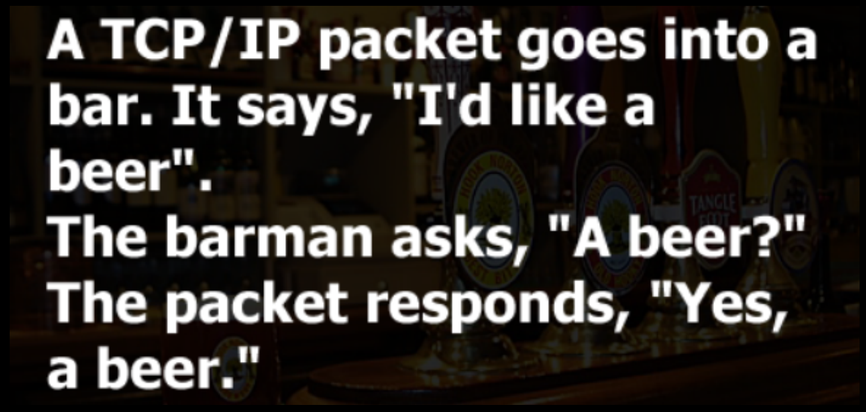


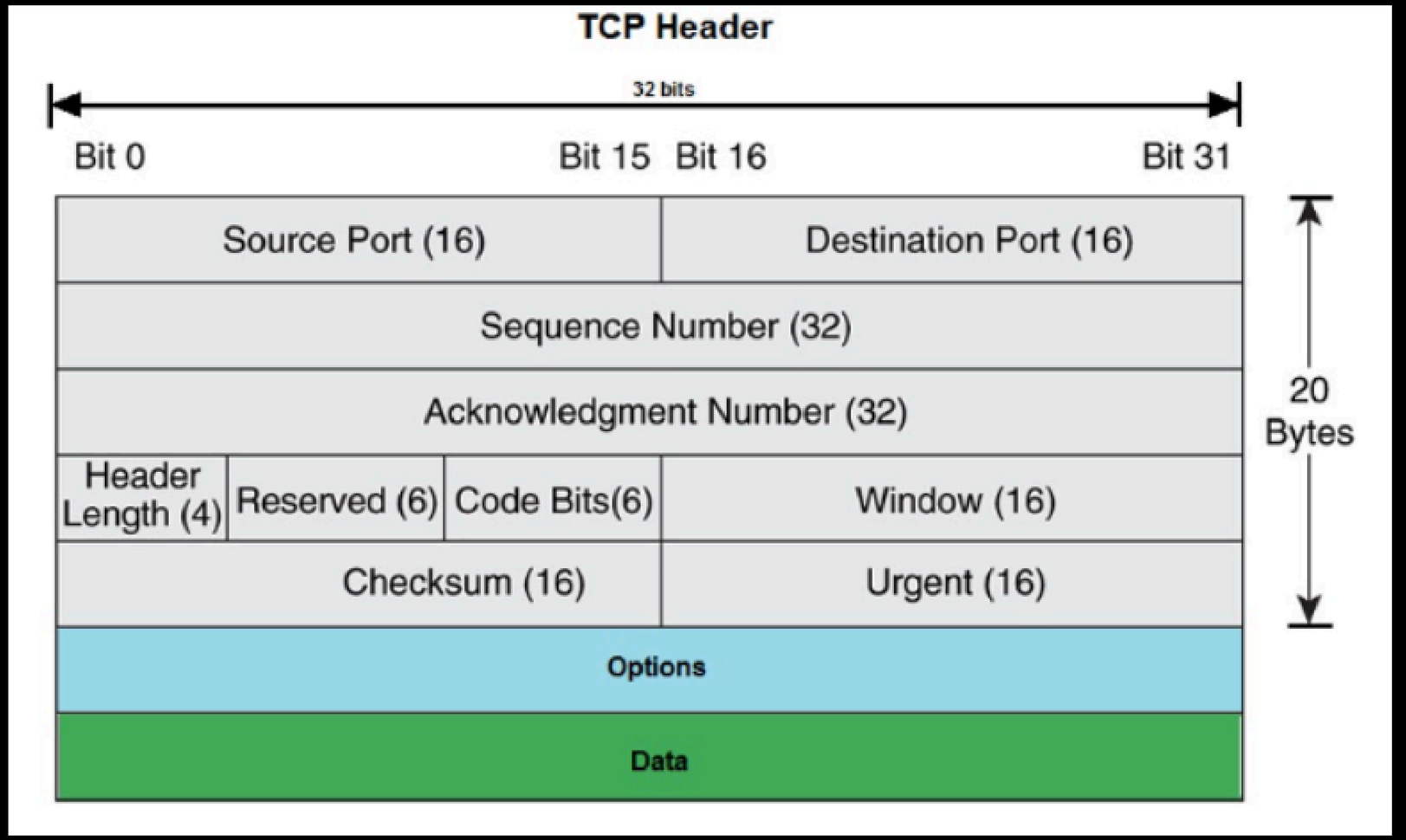
TCP - The Reliable Messenger

Think of TCP as a registered postal service with tracking and delivery confirmation.

- Connection-Oriented: It establishes a dedicated connection before sending data. This is done via a Three-Way Handshake (SYN, SYN-ACK, ACK).
- Reliable Delivery: Guarantees that data will arrive, and in the correct order. It uses sequence numbers and acknowledgements (ACKs). If a piece of data gets lost, it's re-sent.
- Flow Control: Ensures a sender doesn't overwhelm a receiver with too much data at once.

Congestion Control: Helps prevent the entire network from getting clogged up.




TCP Header
32 bits
Bit 0
Bit 15
Bit 16
Bit 31
Source Port (16)
Destination Port (16)
Sequence Number (32)
Acknowledgment Number (32)
Header Length (4)
Reserved (6)
Code Bits(6)
Window (16)
Checksum (16)
Urgent (16)
Options
Data
20 Bytes

When to use TCP?

When you need every piece of data to arrive correctly.

- Web Browsing (HTTP/S)
- Email (SMTP, IMAP)
- File Transfers (FTP)

UDP - The Speedy Messenger

Think of UDP as a standard postcard. You write it, you send it, and you hope for the best.

- Connectionless: No handshake, no pre-existing connection. You just send the data. It's "fire and forget."
- Unreliable: No guarantees.
    - Packets (datagrams) might get lost.
    - They might arrive out of order.
    - They might arrive duplicated.

Very Low Overhead: Because it doesn't do all the extra work of TCP, it's very fast and lightweight.

When to use UDP?

When speed is more important than perfect accuracy. A small amount of data loss is acceptable.

- Live Video/Audio Streaming: Losing a single frame or a millisecond of audio is better than pausing the whole stream to wait for a lost packet.
- Online Gaming: You need the latest game state now. Old data is useless.

UDP Datagram Header Format

Bit #	0 – 7	8 – 15	16 – 23	24 – 31
0	Source Port		Destination Port	
32	Length		Header and Data Checksum	

Intro to Socket Programming

How do our applications actually use TCP or UDP?

Through Sockets!

- A socket is one endpoint of a two-way communication link between two programs running on the network.
- It's an API (Application Programming Interface) that lets our code send and receive data over the network.
- When you create a socket, you specify whether you want to use TCP or UDP

Socket Programming: Conceptual Flow

Step	Server Side	Client Side
1	`socket()` – Create a new socket	`socket()` – Create a new socket
2	`bind()` – Assign IP address and port	`connect()` – Connect to server
3	`listen()` – Listen for incoming connections	
4	`accept()` – Accept client connection	
5	`recv()` / `send()` – Communicate with client	`send()` / `recv()` – Communicate with server
6	`close()` – Close the client connection	`close()` – Close the connection

Questions?