- Sources
- System design - Youtube (neetcode, TechPrep)
- **Brendan Gregg - Systems Performance_ Enterprise and the Cloud (2020, Pearson)**
- geeksforGeeks
- Basic
- Os and nic and ports - how it handles packets
- Nic - network interface card; used to receive/send packets
    - Each has
        - Mac address
        - Two rings(circular buffers) for receiving/sending
        - Ability to generate interrupts
        - Memory mapped registers
            - For settings, like transmit register, receive control register
            - Also has status for received/not or interrupt triggered
            - Buffer loc, interrupt enable/mask
- Dma - nic can save to mem/buffer without cpu (direct memory access). Thus faster perf since cpu more free
- Promiscuous mode - no dma; packets forwarded to cpu
- Interrupt mechanisms
    - Usually have multilevel queue or coalesce related interrupts
        - Coalesce determined by timing (Batches), hardware capabilities/driver, packet-count (up to x before interrupt)
    - Message Signaled Interrupts (MSI/MSI-X): This is an advanced method where interrupts are signaled through memory writes instead of traditional IRQ lines. It enables more efficient handling and scalability for multi-core systems.
    -
    - Direct Interrupt Assignment to CPU Cores: On multi-core systems, interrupt handling can be assigned to specific CPU cores (using affinity). This allows better parallelism, where each core can handle interrupts related to different network operations (e.g., reception, transmission, error reporting) without bottlenecks.
- Rings are circular buffers t receive packet descriptors, which point to memory buffers with actual data
    - Prevents overflow + overwrite maliciously
    - No fragmentation
    - Dma used for accessing/saving ring
- Filter incoming packets by its mac
- Virtual nic - allows for multiple ncis with their own properties on a nic
- Nics offload checksum verification and other tasks to hardware, reducing cpu usage but introducing debug challenges. Tcp segmentation offload too

- Receive side scaling - put specific packets in different ring buffers, allowing for proepr ordering. Improves load distirubiton and avoids performance issues from out of orderpackets
- Nics handle millions of interrupts per sec, which overwhelms cpus. Have throttling (slower interrupt freq) or polling (no interrupts but have thread keep on asking at interval (where not overwhelmed) if there is new one. Cpu intensive)
-
- ## How ports are assigned
- Have phsycial port in nic, and these are for sending/receiving packets
- Software port is logical number designated for receiving specific types of data
    - ranging from 0 to 65535
- Assigned statically and dynamically. Statically for contacting on a server. Dynamic for client
- 0-1023 - protocol related ports (http is 80, https is 443, 22 is ssh, 53 is dns)
- 1024-49151 - registered ports for apps (like mysql: 3306), given by iana (internet assigned numbers authority)
- 49152-65535 - dynamic ports assigned yb os for client receiving/sending
    - Assigned by sequential or random, depends on os
        - Linux does random with hash checks to ensure not in use
        - Uses hash table to keep track ofo nes in use
    - Can change the range in sysctl (ip_local_port_range)
        - Sometimes there are conflicts so would need to restart process
- Can forward ports so that way may receive on x but then forwarded on y (same device or diff device)
- Multiplexing depe depends on multiple client ports
- ## How scale
- Load balancers (see below)
- Cdn to deliver geographically. Also have edge function in edge server of this network to perform other tasks like login
- Sharding - doine by username or some other column (like this db has a-h, this has h-m etc.). Hash based BEST
- Master-slave - best for read heavy
    - If have multiplemasters, slightly better for writes but more complex
- Quorum - have replicas but must agree in majority before write happens. Kinda slow but higher consistency
- log-based/event - aka just queues. Best for writes!
    - Kafka for example
- Nosql - best for distributed writes (like just use cassandra)
- Distributed cache
- Make it stateless. Has auth? then:
    - No Sticky sessions cuz ruins load balancing + distributed layers of cache (cache fragmentation).

        - Makes better use of local cache in server though but that's negligible. Also this is best for sTATEFUL
    - Jwt (no session stored on server. True stateless)
- What happens when you type google.com (or curl it)
- Keyboard shenanigans with interrupt
- Url parsing - get http protocol (https vs http) and domain name
- Local cache dns
- Dns, recursive server dns; otherwise recursive lookup
- Arp to send to router in level 2, data linking. Router nat ur private to its public ip and forwards your request through networks outside of your own
- Reaches server. Form tcp connection with tcp handshake (application layer 4, transport). Form tls connection via tls handshake (application layer 6, presentation)
- Http protocol - send get request with headers specifying compression preference, cookies, host (who it's for; domain name), user-agent (cient id), auth, preferred media
    - Response - Header has set-cookie, server, content type, length
    - Basically cookies/auth, who/to, preferences
-
- What happens to go from edge to destination server
-
- Ipsec - used by vpns to encrypt ip packets (lower level than tls)

IPSec Summary for Interviews

IPSec (Internet Protocol Security) is a protocol suite used to secure IP communications by authenticating and encrypting IP packets. It is commonly used for **VPNs**.

Key Components

1. **Protocols:**
    - **AH (Authentication Header)**: Provides integrity and authentication, but no encryption.
    - **ESP (Encapsulating Security Payload)**: Provides encryption, integrity, and authentication.
2. **Modes:**
    - **Transport Mode**: Encrypts only the data (used for host-to-host communication).
    - **Tunnel Mode**: Encrypts the entire IP packet (used for site-to-site VPNs).
3. **Security Associations (SA):** Defines the relationship for securing traffic, with one SA for inbound and one for outbound traffic.

4. **Key Management:**

    - **IKE (Internet Key Exchange)**: Protocol to establish SAs and exchange keys. **IKEv2** is more secure and efficient than **IKEv1**.

Advantages

- **Confidentiality**: Data is encrypted.
- **Integrity**: Ensures data hasn't been tampered with.
- **Authentication**: Verifies sender identity.
- **Anti-replay**: Prevents replay attacks.

Common Use Cases

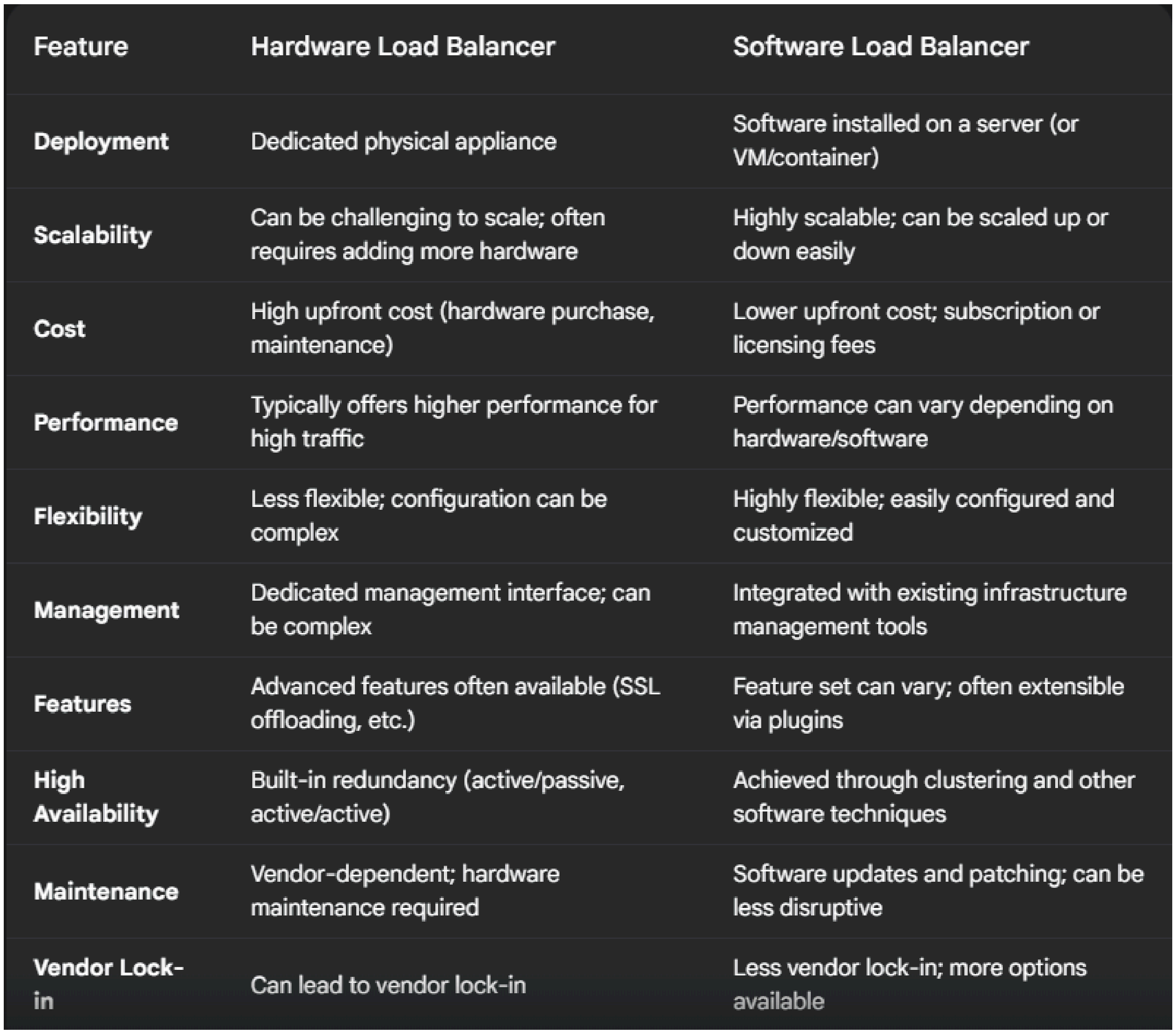
- **VPNs**: Secures remote access and site-to-site communication.

Interview Questions

1. **Difference between AH and ESP?**
    - AH: Authentication and integrity only. ESP: Provides encryption, authentication, and integrity.
2. **Difference between Transport and Tunnel Mode?**
    - Transport: Encrypts only the payload (host-to-host). Tunnel: Encrypts the entire packet (site-to-site).
3. **What is IKE and its versions?**
    - IKE is used for establishing SAs and key exchange. **IKEv2** is more efficient and secure than **IKEv1**.

-
- ## Load balancing
- Balance work load between servers so have relatively equal work loads
- Algorithms:
    - Round robin - circular order
    - Least number of connections
    - Fastest response time
    - Least workload/most amount of cpu capacity
    - Hash ip address
    - Random

- Hardware vs software load balancing



Feature	Hardware Load Balancer	Software Load Balancer
Deployment	Dedicated physical appliance	Software installed on a server (or VM/container)
Scalability	Can be challenging to scale; often requires adding more hardware	Highly scalable; can be scaled up or down easily
Cost	High upfront cost (hardware purchase, maintenance)	Lower upfront cost; subscription or licensing fees
Performance	Typically offers higher performance for high traffic	Performance can vary depending on hardware/software
Flexibility	Less flexible; configuration can be complex	Highly flexible; easily configured and customized
Management	Dedicated management interface; can be complex	Integrated with existing infrastructure management tools
Features	Advanced features often available (SSL offloading, etc.)	Feature set can vary; often extensible via plugins
High Availability	Built-in redundancy (active/passive, active/active)	Achieved through clustering and other software techniques
Maintenance	Vendor-dependent; hardware maintenance required	Software updates and patching; can be less disruptive
Vendor Lock-in	Can lead to vendor lock-in	Less vendor lock-in; more options available

-
- Layer 4 is hardware; layer 7 is app
- 4 is more expensive. Just based on ip address, destination ip, etc. 7 is much more like region, etc.
    - Hardware much faster though and has ssl offloading
    - Both take into account server load/connections or other algo (see above general)
- Level 7 is less expensive and takes into account http headers and cookies andl ocation too

Dns setup

- Buy domain
- Must have authoritative server -> name server -> top level -> root -> recursive server
- Must have zone file with A/AAAA records
    - May have these records (or more)
        - A (Address) Record: Maps a domain to an IPv4 address (e.g., example.com -> 192.0.2.1).

            - AAAA Record: Maps a domain to an IPv6 address (e.g., example.com -> 2001:0db8::ff00:42:8329).
            - CNAME (Canonical Name) Record: Alias of one domain to another (e.g., www.example.com -> example.com).
            - Mx record - for mail servers
            - Txt record - for security
- Once update zone file, depending on ttl, caches mauy take a while to empty out before serving new values (once fetched from authoritative)
    - Dns propogation
- Verify with dig or nslookup
- ## Dns
- When load page, need to resolve web address
- Machine checks local cache if resolution is in there
- If not, then: machine goes to /etc/resolv.conf to find dns server
    - Program usually uses glibc, etc. to read file
    - Network manager, etc. automatically update in some distributions of linux; otherwise, manual
- Contacts dns server, gets web address resolved to ip address
    - Usually recursive, on port 53. Ask recursive server. The recursive server only has cache.
        - Iterative is each nameserver tells u where to go next (next server's ip) like a scavenger hunt
    - If cached, return.
    - Otherwise, the recursive server (recS) will do these steps: go to root and perform the scavenger hunt for you. It will ask which nameserver is associated with next part of url (a zone. Servers can have multiple zones) hat is associated with the current path (ip of next server), split by period. Url is deciphered left to right, with each part being delimited by a period (www.example.com(.))   <-
        - it will go through many servers (so spread out work across servers like sharding almost. Also resilience/redundancy).
    - Root sends recS to top level server (com, net). Top level dns server gets queried, and this gives TLD (top level domain (like .com))
        - a.root-servers.net
    - Go through name servers translating each part of url separated by .
        - Second level domain server (example in example.com), then subdomains (www, api)
    - Authoritative nameserver given by tld, and that one gives actual ip address
        - Does so through A record (for Ipv4) and AAAA record (for ipv6)
            - A record is: full domain name mapped to ip address (also mentions in the entry the type (either AAAA or A)
                - Has TTL, which is taken by cache when cached and that's how long it lasts for
        - Round robin dns:

            - Good since no need for hardware/software for load balancing elsewhere. Simple to implement
            - Bad because if one goes down, then dns will fail for one of x servers. Also, if dns is cached, then this means no round robin until expires (no load balancing)
        - Dynami dns - dhcp dynamic host config protocol, where isp's lease out ip's. Flexible ip in case your ip changes but you want same domain
- Dns client gives ip address to app. App contacts ip address
    - Cache it in recursive resolver server
    - Name servers don't have cache
- Connects to server (likely edge) with tcp. Tcp handshake. Then tls
- Then edge server goes through load balancers, etc. to a backend server
- Server gives info and page is rendered at client level
-
- Dns can have spoofing and dns amplification attacks (see attacks section)
- Diff dns resolvers
- Stub - sends to recursive but does not cache. Is in OS
- Caching - a resolver that caches
- Forwarding resolver - usually a router that knows actual dns servers
- ## Webpage rendering
- Dom rendered from html; parent has child components and this makes the tree's downward relationshiops
- Cssom is made from css for styling and how these html elements should be
- Javascript parsed and can modify dom/cssom
- Cssom + dom combined
- Layout reflow: calculate positions/dimensions of elements
- Paint: render tree to screen
- ## Protocols
- ### Tcp
- Handshake
    - Client sends synch request
    - Server sends synch-ack
    - Client sends ack and connection is established
- Control congestion
    - How it can be improved
- Header
- Retransmission
- General process
    -
- Nagle's algorithm

- Udp
- Header

A UDP packet consists of a **header** and **data**:

1. **Header (8 bytes)**:
    - **Source Port (2 bytes)**: Identifies the sending application.
    - **Destination Port (2 bytes)**: Identifies the receiving application.
    - **Length (2 bytes)**: Specifies the length of the UDP header and the data.
    - **Checksum (2 bytes)**: Used for error-checking the header and data (optional in IPv4, required in IPv6).
2. **Data**:
    - The actual data being transferred.

- Uses socket to send to a destination port at ip (combind is socket). No connection. Easy to forge since have no idea who sender really is. No guaranteed order
-
- Process
- The application writes the data to the socket.
- The kernel constructs the UDP datagram (with source, destination ports, checksum, etc.).
    - Checksum is so can do mathematical operations on it and verify nothing changed
- The system adds the necessary IP header (from the IP layer).
- The entire packet (UDP + IP) is then sent to the network layer for transmission.
-
- The network layer extracts the IP packet and forwards the UDP datagram to the appropriate socket.
- The kernel examines the destination port in the UDP header to determine which application should receive the data.
- The application reads the data from the UDP socket.
- UDP does not guarantee data ordering or delivery, so the application has to handle any issues if needed.
-
- http/rest api definition
- Http is a type of rest api
    - Has put post patch get create delete
    - Stateless
    - Http header and body; stateless so has info for identifier in header
    - 1.2+ is persistant
- Other types include graphql
- Rest api is stateless, ahs unique uris to identify resources (like paths), cacheable, client-server architecture (either client renders or server does)

- Quic (html3)
- Why quic is beter than tls + tcp
- so quic has 1 RTT handshake with tls built in but it uses udp and it has independent streams for multiplexing meaning it's not blocked by lost packets in one strea
    - Others need 2-3
    - implements its own retransmission and congestion control, making it reliable.
        - Multistreaming
        - Encryption
    - Faster conection changes
- Http
- **What is HTTP?**
    - HTTP is a protocol used for transferring hypertext requests and information on the World Wide Web.
    - It defines the rules for how clients (like web browsers) and servers communicate over the web.
    - HTTP is a **request-response protocol** where the client sends a request, and the server sends a response.
- **Stateless Protocol**:
    - Each request-response cycle is independent of others.
    - The server doesn't store information about the client's previous requests (unless the client maintains state via cookies, sessions, etc.).

---

2. HTTP Methods (Verbs)

- **GET**:
    - Requests data from a specified resource.
    - **Idempotent**: Can be repeated without causing side effects.
- **POST**:
    - Sends data to the server to create or update a resource.
    - Not idempotent.
- **PUT**:
    - Updates an existing resource or creates it if it doesn't exist.
    - **Idempotent**: Multiple requests should have the same effect.
- **DELETE**:
    - Removes the specified resource.
    - **Idempotent**: Deleting the same resource multiple times has no effect after the first request.
- **PATCH**:
    - Partially updates a resource.
- **HEAD**:
    - Similar to GET, but only retrieves the headers, not the body.

- **OPTIONS**:
    - Describes the communication options for the target resource.

---

3. HTTP Request Structure

1. **Request Line**:
    - **Method**: GET, POST, etc.
    - **Request URI**: The specific resource being requested.
    - **HTTP Version**: Specifies the version of HTTP being used (e.g., HTTP/1.1, HTTP/2).
2. **Headers**:
    - Provide metadata about the request. Examples: `Content-Type`, `User-Agent`, `Accept-Encoding`.
3. **Body**:
    - Optional in requests like GET, but used in POST, PUT, etc. to send data to the server.

---

4. HTTP Response Structure

1. **Status Line**:
    - **HTTP Version**: The version of HTTP used.
    - **Status Code**: A 3-digit code indicating the result (e.g., `200 OK`, `404 Not Found`, `500 Internal Server Error`).
    - **Status Message**: A human-readable explanation of the status code.
2. **Headers**:
    - Provide metadata about the response. Examples: `Content-Type`, `Content-Length`, `Set-Cookie`.
3. **Body**:
    - Contains the data being returned (e.g., HTML, JSON, image, etc.).

---

5. HTTP Status Codes

1. **1xx - Informational**:
    - `100 Continue`: The server has received the request and the client can continue sending the rest.
2. **2xx - Success**:
    - `200 OK`: The request was successful and the server has returned the requested data.

    - `201 Created`: The request was successful and a new resource was created.
3. **3xx - Redirection**:
    - `301 Moved Permanently`: The resource has been permanently moved to a new URL.
    - `302 Found`: The resource has been temporarily moved.
4. **4xx - Client Errors**:
    - `400 Bad Request`: The request is malformed.
    - `404 Not Found`: The requested resource could not be found.
    - `403 Forbidden`: The server understands the request but refuses to authorize it.
5. **5xx - Server Errors**:
    - `500 Internal Server Error`: A generic server-side error.
    - `502 Bad Gateway`: The server received an invalid response from an upstream server.

---

6. HTTP Headers

- **Request Headers**:
    - `User-Agent`: Identifies the client (browser, mobile app, etc.).
    - `Accept`: Specifies the media types that the client can process (e.g., `Accept: text/html`).
    - `Authorization`: Contains credentials for authenticating the user.
- **Response Headers**:
    - `Content-Type`: Specifies the media type of the response body (e.g., `text/html`, `application/json`).
    - `Cache-Control`: Directives for caching mechanisms (e.g., `no-cache`).
    - `Location`: Used in redirection responses to specify the new URL.
- Hsts - enforces https
- ## ARP - address resolutionp rotocol

Purpose: ARP is used to map a known **IP address** to a **MAC (Media Access Control) address** on a local network.
How it works:

- A device needs to send a packet to another device in the same local network, but it only knows the **IP address** of the destination.
- The device broadcasts an ARP request to all devices on the network, asking, "Who has IP X.X.X.X?"
- The device with the matching IP address responds with its MAC address.
- The requesting device stores this information in its **ARP cache** for future communications.

Example: If a computer (A) wants to send data to another computer (B) on the same network, it will first use ARP to resolve B's IP address to its MAC address.

- Arp needed so that way subnet hardware (Switches) knows how to route; ip addresses are for between networks
- Ftp

Purpose: FTP is used to transfer files between a client and a server over a TCP/IP network.
How it works:

- FTP uses two separate channels: a **control channel** (port 21) for commands and responses, and a **data channel** (usually port 20) for transferring the file content.
- **Active Mode**: The client opens a port and tells the server to connect to it for data transfer.
- **Passive Mode**: The server opens a random port and the client connects to it for data transfer (this mode is often used behind firewalls or NAT).

Example: A user might use an FTP client to upload files to a web server by connecting to port 21, and the server will send files back over the data channel.

-
- Icmp
- **Purpose**: ICMP is used for sending control and error messages in a network, commonly used for diagnostic tools like **ping** and **traceroute**.
- **How it works**:
  - ICMP messages are encapsulated in IP packets, where the **type** of message indicates the purpose (e.g., echo request, destination unreachable).
  - **Ping** (ICMP Echo Request and Echo Reply) is one of the most common uses, which sends an echo request to a host and waits for an echo reply to check connectivity.
  - ICMP is used for network diagnostics, error reporting, and troubleshooting (like when routers inform hosts that a destination is unreachable).
- **Example**: A user running a **ping** command sends an ICMP Echo Request to a server. The server responds with an ICMP Echo Reply, confirming that it is reachable.

. ICMP Header (Encapsulated within the IP Header)

The **ICMP header** contains control information about the message being sent.

Echo request - used in ping to test reachability of host

Reply tells that it is alive

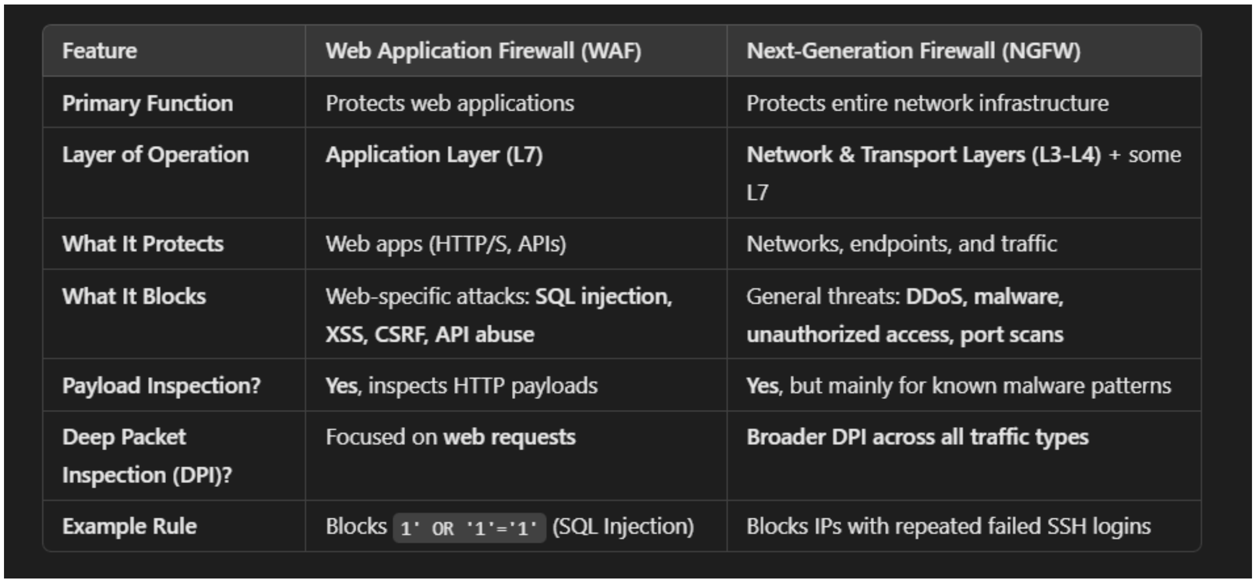
Traceroute uses tll of icmp

Used for debugging. For instance:

- **Type** (8 bits): Specifies the type of ICMP message (e.g., Echo Request, Echo Reply, Destination Unreachable).
    - **0** = Echo Reply
    - **8** = Echo Request
    - **3** = Destination Unreachable
- **Code** (8 bits): Provides additional information depending on the Type. For example:
    - For Echo Request, the code is **0**.
    - For Destination Unreachable, the code might specify whether it's a network, host, or port unreachable.
- **Checksum** (16 bits): Used for error-checking the ICMP message.
- **Rest of the Header** (varies):
    - **Echo Request/Reply** messages contain a **Identifier** and **Sequence Number** to match requests with replies.
    - **Destination Unreachable** messages contain the **IP header** of the original packet that caused the error.
- ## Bgp - border gateway protocol
- Provides directions go get from one ip to another. Is basically rouers with routing tables, but given by AS, autonomous system
    - Shortest path
- Autonomous system is system of n etworks/subnets run by a singular org. They have similar rules
- Advertisement + Peering - AS upon business agreement allow syou to peer into what they can reach (sequence to reach is AS path). Also give cost
- When an AS receives route advertisements from its neighbors, its BGP routers store this information in their Routing Information Base (RIB). finds best routes that it can give with this info, aiming for shortest +S cheapest
- Then advertise this
- so bpg hijacking is AS fakes its ownership saying it owns ip's it doesn't and if not filtered they now control a specific ip set and can steal traffic
    - Prevent with RIR (regional internet registries) to assign subip addresses(prefixes) and IANA (internet assigned numbers authority) to assign ip addresses in record and for ASs to check it with filter when other ASs announce
    -
- Border Gateway Protocol (BGP) is the Internet's routing protocol, determining the most efficient path for data transmission between autonomous systems (ASes), similar to how the postal service routes mail.
-
- Autonomous Systems (ASes)
- The Internet consists of multiple ASes, which are networks managed by ISPs, tech companies, universities, and government institutions. ASes exchange routing information via peering sessions to keep routing data updated.
-
- BGP Routing

- BGP selects paths based on factors like hop count, cost, and policy preferences. Each AS advertises available routes, allowing BGP to dynamically update paths as network conditions change.
-
- Types of BGP
-
- External BGP (eBGP): Used between different ASes for Internet-wide routing.
- Internal BGP (iBGP): Used within an AS for internal traffic management.
- BGP Attributes
- Routers determine the best path using attributes like weight, local preference, and AS path length. These attributes help optimize network efficiency.
-
- BGP Security Issues
-
- BGP Hijacking: Incorrect route advertisements (intentional or accidental) can disrupt Internet traffic. Examples include incidents in Turkey (2004), Pakistan (2008), and Verizon (2019).
- Attacks: Phishing, DoS, on-path attacks, and impersonation can exploit BGP vulnerabilities.
- BGP Security Solutions
    - Resource Public Key Infrastructure (RPKI): Uses cryptographic records to validate route announcements.
    - Cloudflare Route Leak Detection: Alerts customers about unauthorized route advertisements.
- ### NAT (Network Address Translation)
- - **Purpose**: A method used to modify the source or destination IP addresses of packets as they pass through a router or firewall.
- - **How it works**: Commonly used for converting private IP addresses within an internal network to a public IP address when sending data to the internet (and vice versa). It helps conserve IP address space and provides a layer of security.
- Translates private ip to public ip
    - Private ip is just not accessible to public. Can be in house network, etc.
- Router intercepts request you are sending outside and translates your device's ip address with router's public ip address
    - This is why people on same private network have same public ip address
- Layer 3 (modifies ip source in ip headers)
-
- ## Firewalls
- Is table with rules. 3 actions: accept, deny, drop. Default is configurable; usually accept/reject
    - Reject - icmp or tcp rst (termination abruptly) to say that was rejected. Drop is just silent rejection
- Packet filtering - can filter based on ip, ports, protocols

    - One row for each combination of source, dest ip, source/dest port
    - Inspect
- State filtering - packets' history are stored in state table. May be rejected based on history (like not ana active tcp connection)
- Application layer (waf) - can block specific content and if protocols like http are being misused. Can be done through proxy firewall so goes through here before forwards/reaches server.
    - Just layer 7 (http/s)
    - Has known patterns like 1' OR 1 == 1 for sql injection (true statement from user input that is used for sql so can select all)
    - Excessive api calls, unusual logins
    - Rate limiting
    - Malicious scripts in payloads
    - Correct headers in http
- Next generation - deep packet inspection, application inspection, ssl/ssh inspection
    - Dep packet - not just headers but inside. Malware, data, etc.
    - Can look at more than just ports. Knows what a legit http connection is
    - NGFWs can decrypt, inspect, and re-encrypt traffic to detect threats that traditional firewalls would miss.
    - 3,4,7 layer
- Circuit level - session; no packet inspection. Less computation. Verifies tcp handshake/session by connecting tcp to client AND server; acts as proxy. Verifies tcp on both sides that both are communicating with who they think they are
- Software - is software running on OS. only protects machine is on. Blocks at app level, 3, 4, too. Free or low cost. Adds ovehread. Can be disabled by malware
- Hardware - expensive but faster than software. Basic packet, stateful. Can be ngfw
- Cloud - is sold as a service (redirected to company's servers, then back to yours)
    - Bot detection, ips, dpi
- Other security: Ips vs ids - ips is intrusion prevention system; ids is intrusion detection system. Former blocks/acts while ids just alerts. These can be built into firewalls.
- general:
- Pros: protection, network recording
- Cons: needs fine tuning due to threats at varying levels, costs, complexity, slower



Feature	Web Application Firewall (WAF)	Next-Generation Firewall (NGFW)
Primary Function	Protects web applications	Protects entire network infrastructure
Layer of Operation	**Application Layer (L7)**	**Network & Transport Layers (L3-L4)** + some L7
What It Protects	Web apps (HTTP/S, APIs)	Networks, endpoints, and traffic
What It Blocks	Web-specific attacks: **SQL injection, XSS, CSRF, API abuse**	General threats: **DDoS, malware, unauthorized access, port scans**
Payload Inspection?	**Yes**, inspects HTTP payloads	**Yes**, but mainly for known malware patterns
Deep Packet Inspection (DPI)?	Focused on **web requests**	**Broader DPI across all traffic types**
Example Rule	Blocks `1' OR '1'='1'` (SQL Injection)	Blocks IPs with repeated failed SSH logins

Firewalls: Reasons You May Get Blocked

A firewall is a security device (hardware/software) that filters network traffic based on rules. You might get blocked due to:

1. **IP Blacklisting:** Your IP is in a deny list (e.g., spam, malicious activity, or a misconfigured rule).
2. **Port Restrictions:** The firewall blocks certain ports (e.g., only allowing ports 80/443 for web traffic).
3. **Protocol Filtering:** Only specific protocols (TCP, UDP, ICMP) are allowed.
4. **Rate Limiting:** Too many requests in a short time may trigger a block (DoS/DDoS protection).
5. **Geo-Blocking:** Traffic from certain countries is restricted.
6. **Deep Packet Inspection (DPI):** Blocks based on payload contents (e.g., detecting malicious payloads or P2P traffic).
7. **Spoofed/Malformed Packets:** If your packets don't conform to expected structures, the firewall may drop them.
8. **Expired/Revoked Certificates (HTTPS/TLS):** Some firewalls block expired or self-signed certificates.

Firewall vs. Web Application Firewall (WAF)

Feature	Regular Firewall	Web Application Firewall (WAF)

Focus	Network-level filtering	Application-layer filtering (Layer 7)
Works at	OSI Layers 3 (Network) & 4 (Transport)	OSI Layer 7 (Application)
Blocks	IPs, ports, protocols	SQL injections, XSS, CSRF, API abuse
Protection Scope	Whole system/network	Web applications only
Example Use Cases	Prevent unauthorized network access, DDoS protection	Protect web apps from injection attacks, bot traffic
Common Tools	iptables, firewalld, pfSense, Cisco ASA	Cloudflare WAF, AWS WAF, ModSecurity

---

How a Regular Firewall Works

A firewall works by examining incoming and outgoing packets and applying predefined rules.

1. **Packet Filtering (Stateless or Stateful)**
    - **Stateless:** Checks each packet independently against rules.
    - **Stateful:** Tracks connections (e.g., TCP handshake) and ensures responses are expected.
2. **Network Address Translation (NAT)**
    - Allows multiple devices to share a single public IP.
    - Prevents direct access to internal systems.
3. **Proxy-Based Filtering**
    - Some firewalls act as proxies to inspect and relay traffic.
4. **Deep Packet Inspection (DPI)**
    - Analyzes packet contents (e.g., detecting malware, blocking P2P traffic).
5. **Logging & Monitoring**
    - Logs network activity for security audits.

-

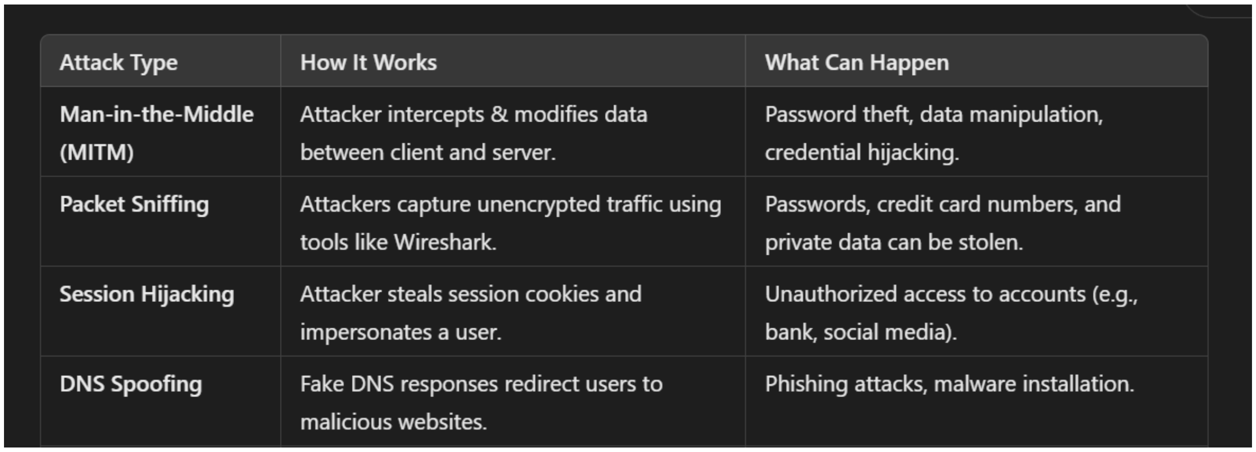
- Protocol trivia
- Tls handshake
- Tls 1.2
  - Hello, server hello (cipher decide), exchange key + certificate, generate shared secret, confirm encryption from client, http request/response
- Tls 1.3
  - Client hello with key, server hello with certificate and key share, generate shared key, http request/response
- More of the handshake happens in initial for 1.3
- Certificate's signature verified with list of valid CA signatures in browser by client
- Key share happens so:
  - Share public keys in ecdhe and compute private key independently; no decryption/encryption needed
  - Use own private + public key of each other's to generate shared secret
- Rsa only: Throughout handshake, have public key encrypt to send to x and then x's private key to decrypt. We send pre master secret to server this way. Server + client uses this pre master secret to generate
- Udp vs tcp
- Udp fast but connectionless and lose packets
  - Good for streaming and video games
- Tcp slower but ensures packets are not lost due to retrasnmit. Has congestion control
  - Good for email, and used to be for http
- Misc
- DHCP, SLAAC, IPv4, IPv6, BGP, OSPF, iBGP, NAT, QUIC
- Here are brief definitions for each of the terms you mentioned:
-
- ### **1. DHCP (Dynamic Host Configuration Protocol)**
- - **Purpose**: A network protocol used to automatically assign IP addresses and other network configuration details (like gateway and DNS server) to devices on a network.
- - **How it works**: A device (DHCP client) sends a broadcast request (DHCPDISCOVER) to a DHCP server, which then assigns an IP address and other details to the client.
-
- ### **2. SLAAC (Stateless Address Autoconfiguration)**
- - **Purpose**: A method used in IPv6 networks for devices to automatically configure their own IP addresses without the need for a DHCP server.
- - **How it works**: Devices generate their own address using a combination of a network prefix (provided by routers) and their own unique interface identifier, often derived from their MAC address.
-
- ### **3. IPv4 (Internet Protocol version 4)**

- - **Purpose**: The fourth version of the Internet Protocol, which is widely used to assign unique addresses to devices on a network.
- - **Characteristics**:
-   - 32-bit address length.
-   - Supports around 4.3 billion unique IP addresses.
-   - Notable for running out of available IP addresses due to increasing internet-connected devices.
-
- ### **4. IPv6 (Internet Protocol version 6)**
- - **Purpose**: The successor to IPv4, designed to address the limitations of IPv4, particularly the limited address space.
- - **Characteristics**:
-   - 128-bit address length.
-   - Supports approximately 340 undecillion unique addresses.
-   - Designed to accommodate the growing number of devices on the internet.
-
- ### **5. BGP (Border Gateway Protocol)**
- - **Purpose**: The protocol used to exchange routing information between different autonomous systems (ASes) on the internet.
- - **How it works**: BGP helps routers decide the best path for routing packets across the internet, based on factors like policy, network reachability, and routing preferences.
-
- ### **6. OSPF (Open Shortest Path First)**
- - **Purpose**: A link-state routing protocol used within an autonomous system (AS) to find the best path for data across a network.
- - **How it works**: OSPF routers share information about the network topology, allowing each router to independently calculate the shortest path to all reachable destinations within the AS using the Dijkstra algorithm.
-
- ### **7. iBGP (Internal BGP)**
- - **Purpose**: A variant of BGP used for routing between routers within the same autonomous system (AS).
- - **How it works**: iBGP ensures that all routers within the AS are aware of the best routes to external destinations by distributing BGP routing information internally across the AS.
-
-
- ## Misc
- TCP control bits
    - URG (Urgent): Signals that the data in the segment is urgent and should be processed immediately, bypassing normal queuing.
    -
    - ACK (Acknowledgment): Used to acknowledge the receipt of a segment. It's a key part of TCP's reliability mechanism.

    -
    - PSH (Push): Tells the receiving application to deliver the data to the application immediately, rather than buffering it.
    -
    - RST (Reset): Used to reset the connection, often due to an error or an abnormal termination.
    -
    - SYN (Synchronize): Used to initiate a connection (part of the three-way handshake).
    -
    - FIN (Finish): Signals the end of the data transmission, used to gracefully close the connection.

- ## Osi

- Layer 1 - physical
    - Includes your cables, 5g, data sent as bit stream
- Layer 2 - data link
    - Data broken up into frames
    - Arp
        - Arp to find mac if in same lan
- Layer 3 - network
    - Segments split up into packets. Ip info added via headers
        - Ip header:
            - Destination, source, ttl, protocol which, checksum, fragment numbers, lelngth, ip version
                - Better summary: source, destination, options, protocol, info for fragmentation, ttl
    - icmp
- Layer 4 - transport
    - Data broken up into segment
    - Headers added (related to protocol)
    - tcp
- Layer 5 - session
    - Connection start/end defined here
    - Checkpoint stored so that way if crash don't have to resend whole set of packets
    - Parts of tcp happen here
- Layer 6 - presentation
    - Compression, encryption happens here
    - tls
- Layer 7 - application
    - User sends http request

- Scalable system, and security
- Facebook auth
-
- Security issues
- Open wifi - easier due to anybody can join + no additional encryption
    - Allows for listening (packet sniffing, man in the middle, session hijacking, dns amplfiication and spoofing, reaquest replay, ssl stripping, fake wifi twin, arp spoofing)
        - Only dns amplification/spoofing if open dns
- Due to no tls
- 

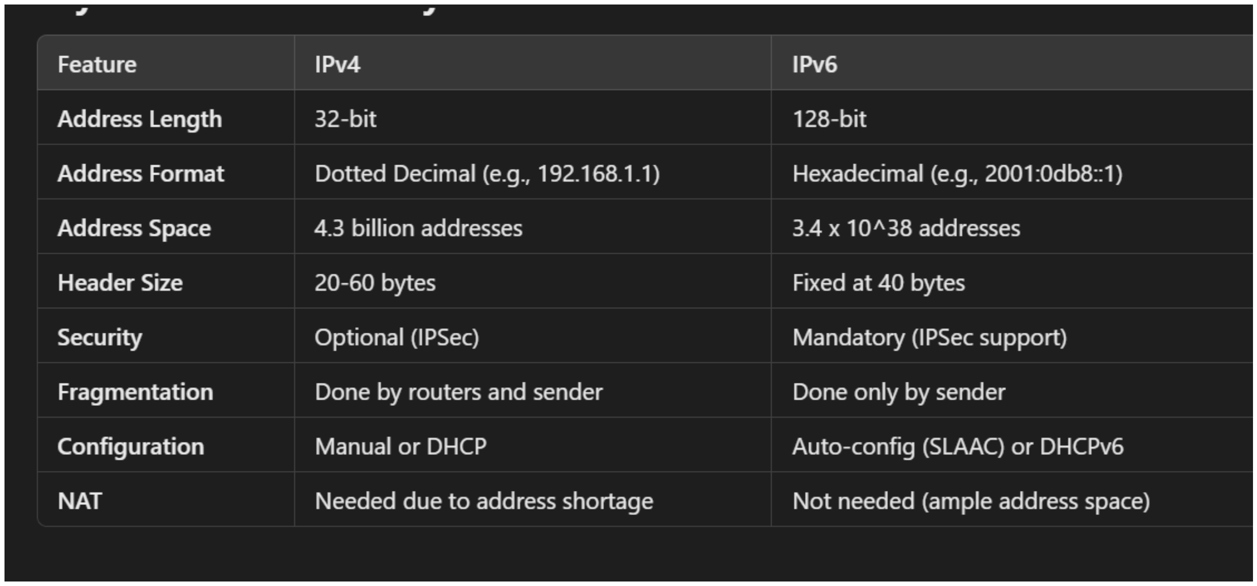
Attack Type	How It Works	What Can Happen
Man-in-the-Middle (MITM)	Attacker intercepts & modifies data between client and server.	Password theft, data manipulation, credential hijacking.
Packet Sniffing	Attackers capture unencrypted traffic using tools like Wireshark.	Passwords, credit card numbers, and private data can be stolen.
Session Hijacking	Attacker steals session cookies and impersonates a user.	Unauthorized access to accounts (e.g., bank, social media).
DNS Spoofing	Fake DNS responses redirect users to malicious websites.	Phishing attacks, malware installation.

- Capture and replay (very similar to packet sniffing but replay it) and replay it again to get same effect twice. Ex: X gives money to attacker. Attacker replays it to bank so they get 2x amount
- Inject bad code into request so server/client is compromised (usually client)
-
- On path - more sever than man in the middle b/c already hijacked/on path or in between client/server as an actual middleman (like think a fake worker at post office). Common for email
    - Example is fake public wifi
- general
- 1. DDoS (Distributed Denial of Service)
- 🔹 What it is: Flooding a server with excessive traffic to overwhelm it.
- 🔹 How it works:
-
- Attackers use botnets (compromised devices) to send massive amounts of requests.
- Server resources (CPU, bandwidth, memory) get exhausted, causing downtime.
- 🔹 Types of DDoS:
- Volumetric Attacks → Overload bandwidth (e.g., UDP flood).
- Protocol Attacks → Exploit weaknesses in TCP, SYN floods.
- Application-Layer Attacks → Overload specific services (e.g., HTTP GET flood).
- 🔹 Mitigation:

- Rate limiting, CAPTCHAs, load balancers, DDoS protection services (Cloudflare, AWS Shield).
- 2. Memcached DDoS (Reflection Attack)
- 🔹 What it is: Amplified DDoS attack using memcached servers.
- 🔹 How it works:
-
- Attacker spoofs victim’s IP and sends small UDP requests to misconfigured memcached servers.
- Memcached replies with huge responses (up to 51,000x amplification).
- The victim gets flooded with massive data, causing a DDoS.
- 🔹 Mitigation:
- Disable UDP on memcached unless needed.
- Use firewalls to block unnecessary traffic.
- Rate-limit responses from memcached.
- 3. ARP Spoofing (MITM Attack)
- 🔹 What it is: Attacker tricks a network into sending traffic through their machine.
- 🔹 How it works:
-
- Attacker spoofs ARP replies to tell devices that the attacker's MAC address is the gateway.
- Victim unknowingly sends all traffic through the attacker.
- Can be used for MITM (Man-in-the-Middle), sniffing, and session hijacking.
- 🔹 Mitigation:
- Use static ARP entries for critical devices.
- Enable Dynamic ARP Inspection (DAI) on network switches.
- Use encrypted protocols (HTTPS, SSH, VPNs) to prevent sniffing.
- 4. Traffic Spikes (Legitimate vs. Attack)
- 🔹 What it is: A sudden surge in traffic that may be legitimate (viral event, sale) or an attack (DDoS, botnet traffic).
- 🔹 How to handle:
-
- Monitor traffic patterns (e.g., sudden vs. gradual increases).
- Use autoscaling for legitimate spikes.
- Rate-limit unexpected traffic bursts to prevent overload.
- Implement WAF (Web Application Firewall) to block malicious bots.
-
-
- Cross site scripting
    - Abuses existing cookies in browser for auth
    - Attacker gets victim to click embedded link (with script, usually in email) that immediately sends request to a website to perform an auth-only request (like send money in a bank website)
    - It’s like asking someone to press a button saying the button will give them free money but you need to put debit card in the reader. Instead of paying it takes from you
        - Preventive methods:
            - same origin policy (SOP) and cross origin resource sharing (CORS)

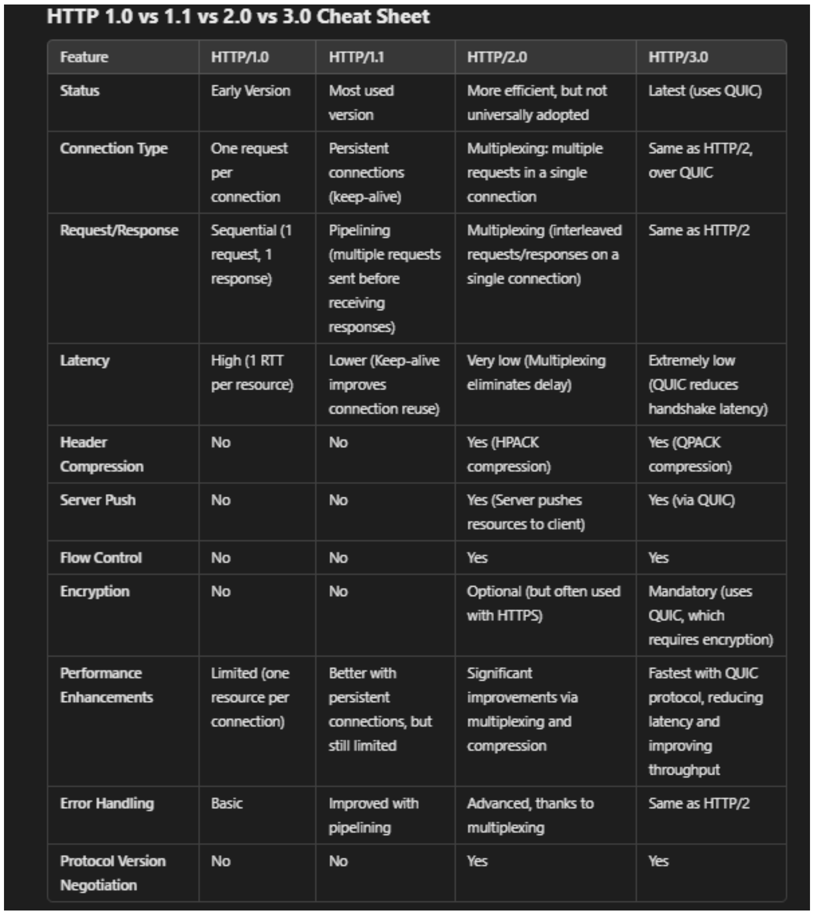
            - Requests must come from same origin and can't have different origins unless whitelisted, esp for post since that changes data
- Sql injection - In url or textbox, execute partial sql statements in section of input that would be executed in sql, like if url says studentID= in it to get student and then you write or 1=1 so that way you always have statement as true and short circuit and get all student data
    - Have prepared statements (fill in blank sql statements) and sanitization (and setting proper lowest possible permissions for DB; aka least privilege)
- ## Dns
- Dns spoofing (cache poisoning)
    - Since dns uses udp, you can forge requests to act as authoritative name server
    - When recursive server starts asking servers and eventually asks authoritative name server for domain->ip, attacker can forge udp datagrams and tell the dns that the ip address is X, when it really is Y. X would be ip to the attacker's website so whoever goes there gets data stolen/hacked
    - Dns server then caches this so whenever that domain name is resolved again it just gives the cache (poisoned value) instead of the authoritative name server
        - But, attacker must know these things to forge udp:
            - Port udp is receiving on for dns server (used to be set but now is random, sent by recS and then authoritative sends back on there)
            - Which domain is not currently cached (will pick this to start atk)
            - Request id number
                - It's sent in request and is expected in response so know that you are server that was originally asking
            - Which authoritative nameserver will be giving the answer
        - This is very hard so it's rare
    - Domain Name System Security Extensions, (DNSSE) is alternative but not widely adopted. Uses public keys like tls to encrypt data (and verify sender)
- Domain spoofing - when you fake a domain to make it sound real like google.net
- on-path
- Dns amplification
    - Clogs victim's network
    - It's like ordering everything from a restaurant and then sending to victim's house
    - Basically make dns query for everything (nothing super complex but response is) and keep spamming htis. Fake your ip address so that way it's sent to victim (ip address you faked) and clogs their network bandwidth
        - Fix by havingi less open servers (only trust X)
        - Source ip verification
        - Blackholing - specific ips, network patterns, etc. -> dump packets and don't deliver
- Dns tunneling - pass malware into dns queries to hijack
- Dns hijack - hijacked dns server gives bad ip's to a domain resolution to hack victims (who are doing dns)
- Nxdomain - ddos so nobody can use dns

- Phantom domain attack - attacker makes slow domain fake servers so recursive server cannot get dns
-
- ## Ipv4 vs 6



Feature	IPv4	IPv6
Address Length	32-bit	128-bit
Address Format	Dotted Decimal (e.g., 192.168.1.1)	Hexadecimal (e.g., 2001:0db8::1)
Address Space	4.3 billion addresses	3.4 x 10^38 addresses
Header Size	20-60 bytes	Fixed at 40 bytes
Security	Optional (IPSec)	Mandatory (IPSec support)
Fragmentation	Done by routers and sender	Done only by sender
Configuration	Manual or DHCP	Auto-config (SLAAC) or DHCPv6
NAT	Needed due to address shortage	Not needed (ample address space)

-
- Ipsec for 6 was mandatory, such as auth header and encryption
- Basically just more addresses with ipv6
- ## Http 1.1 vs 2 vs 3
- 1 was slow since only 1 request per conection. 1.1 is persistent while 1.0 isn't. 2 had multiplexing, priority sorting, and server push, header compression. 3 ditches tcp unlike first two and uses quic + udp so even faster
    - Multiplexing - one tcp connection, multiple data streams
    - Priority sorting sorthat way most important page resources sent first; this allows for most important parts to load first
    - Server push - preemptively send to client
    - 2 has pushing, which allows servers to push extra content ahead of time and let client know what that content is. Is like giving a map before giving amusement park

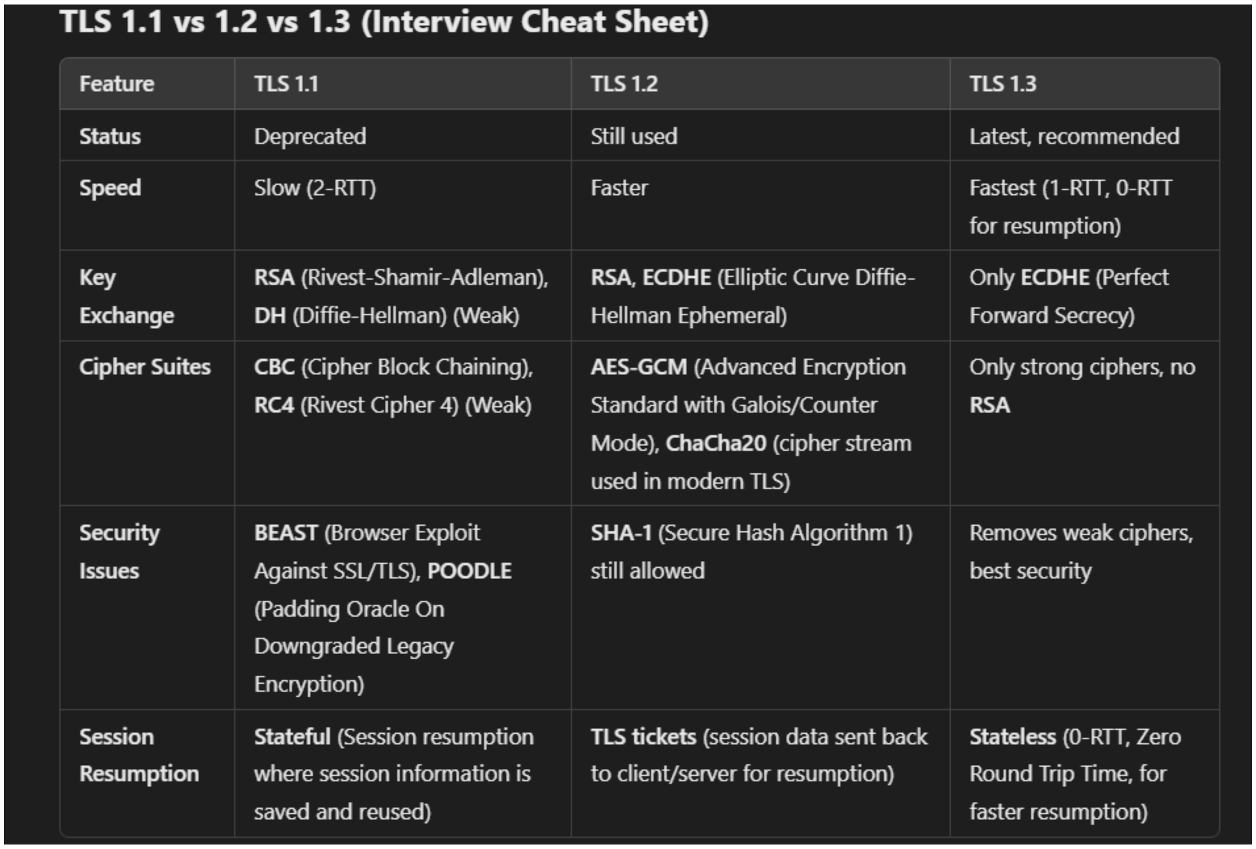
- 

HTTP 1.0 vs 1.1 vs 2.0 vs 3.0 Cheat Sheet

Feature	HTTP/1.0	HTTP/1.1	HTTP/2.0	HTTP/3.0
Status	Early Version	Most used version	More efficient, but not universally adopted	Latest (uses QUIC)
Connection Type	One request per connection	Persistent connections (keep-alive)	Multiplexing: multiple requests in a single connection	Same as HTTP/2, over QUIC
Request/Response	Sequential (1 request, 1 response)	Pipelining (multiple requests sent before receiving responses)	Multiplexing (interleaved requests/responses on a single connection)	Same as HTTP/2
Latency	High (1 RTT per resource)	Lower (Keep-alive improves connection reuse)	Very low (Multiplexing eliminates delay)	Extremely low (QUIC reduces handshake latency)
Header Compression	No	No	Yes (HPACK compression)	Yes (QPACK compression)
Server Push	No	No	Yes (Server pushes resources to client)	Yes (via QUIC)
Flow Control	No	No	Yes	Yes
Encryption	No	No	Optional (but often used with HTTPS)	Mandatory (uses QUIC, which requires encryption)
Performance Enhancements	Limited (one resource per connection)	Better with persistent connections, but still limited	Significant improvements via multiplexing and compression	Fastest with QUIC protocol, reducing latency and improving throughput
Error Handling	Basic	Improved with pipelining	Advanced, thanks to multiplexing	Same as HTTP/2
Protocol Version Negotiation	No	No	Yes	Yes

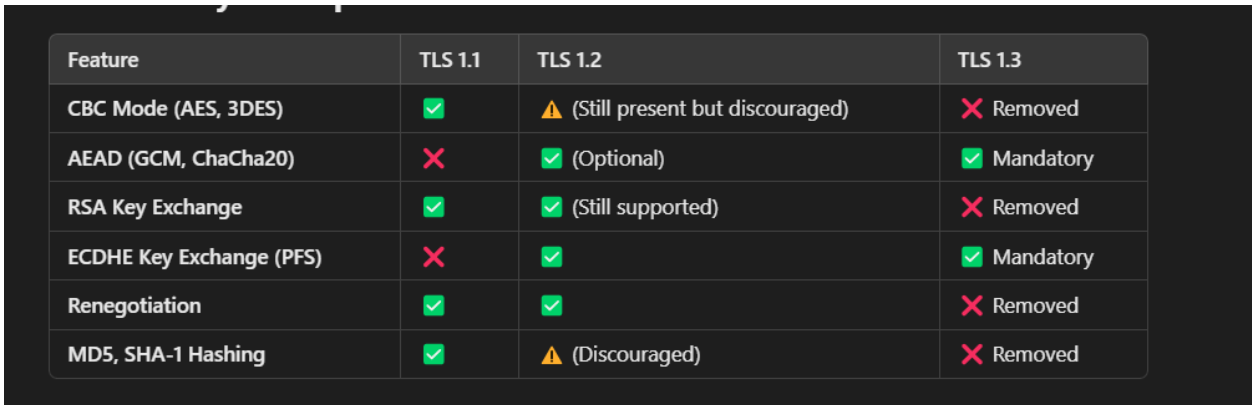
-
- ## Tls 1.1 vs 1.2 vs 1.3
- Tls 1.1 is obsolete security and stateful and slow, 1.2 is standard security and faster, 1.3 is fastest/most secure. Stateless, 1 handshake round trip (0 for re) while others are 2
- Dh vs ecdhe
  - **elliptic curve cryptography** instead of modular arithmetic with large prime numbers.
- Better ciphers used in 1.2, 1.3 (aead only for 1.3). 1.2 introduces aead, ecdhe but allows for older like cbc
  - but only 1.3 fully removes cbc, rsa, renegotiation (on security strength, re-auth, parameters), md5/sha-1 hashing (weak)
    - Why cbc weak: can find info about plain text by manipulating cipher text and seeing padding error. Iv can be reused, changes to cipher text go undetected
    - Renegotiation weak: allowed to inject malicious code when renegotiating
    - Why md5 / sha-1 hashing weak: 128-bit hash and is considered cryptographically broken; can have collisions. Sha-1 same but 160-bit hash

                - Why rsa weak: 2048 bit keys is too small; computational power (quantum) increases and this is easy to crack. Must be larger, but this makes enc/dec slow. No forward secrecy
                    - How it works: 2 large primes, find modulus, totient function, calculate public d, private e. Private = modulus and private e. Public = n and public e
                    - Decrypt with private; encrypt with public
                    - Server sends public; keeps private. Client encrypts with public. Vise versa
                - Diffie hellman focuses on Discrete Logarithm Problem instead of factoring large primes
- Essentially: 1.1 uses rsa or diffie hellman, which is outdated
- 1.3 forces ephemeral diffie hellman with perfect forward secrecy while 1.2 doesn't. 1.2 allows for static RSA and ecdhe
    - Perfect secrecy: unique key every session; even if leaked. Random values to session + ephemeral during key derivation causes this, even if same algo used across sessions
    - Basically keys every session differ
- Essentially: 1.1, 1.2 have 2 rtt. 1.3 has 0 rtt resumption, 1 rtt start
- Essentially: 1.1 is stateful, 1.2 introduces tls tickets, 1.3 enforces tickets and is stateless
    - Stateful - store on server, take up memory o(n)
    - Ticket is almost like an encrypted cookie; use to resume session. PSKs in the ticket and is shared secret used to derive session keys for encryption during session

- 

TLS 1.1 vs 1.2 vs 1.3 (Interview Cheat Sheet)

Feature	TLS 1.1	TLS 1.2	TLS 1.3
Status	Deprecated	Still used	Latest, recommended
Speed	Slow (2-RTT)	Faster	Fastest (1-RTT, 0-RTT for resumption)
Key Exchange	**RSA** (Rivest-Shamir-Adleman), **DH** (Diffie-Hellman) (Weak)	**RSA, ECDHE** (Elliptic Curve Diffie-Hellman Ephemeral)	Only **ECDHE** (Perfect Forward Secrecy)
Cipher Suites	**CBC** (Cipher Block Chaining), **RC4** (Rivest Cipher 4) (Weak)	**AES-GCM** (Advanced Encryption Standard with Galois/Counter Mode), **ChaCha20** (cipher stream used in modern TLS)	Only strong ciphers, no **RSA**
Security Issues	**BEAST** (Browser Exploit Against SSL/TLS), **POODLE** (Padding Oracle On Downgraded Legacy Encryption)	**SHA-1** (Secure Hash Algorithm 1) still allowed	Removes weak ciphers, best security
Session Resumption	**Stateful** (Session resumption where session information is saved and reused)	**TLS tickets** (session data sent back to client/server for resumption)	**Stateless** (0-RTT, Zero Round Trip Time, for faster resumption)

- 

Feature	TLS 1.1	TLS 1.2	TLS 1.3
CBC Mode (AES, 3DES)	✅	⚠️ (Still present but discouraged)	❌ Removed
AEAD (GCM, ChaCha20)	❌	✅ (Optional)	✅ Mandatory
RSA Key Exchange	✅	✅ (Still supported)	❌ Removed
ECDHE Key Exchange (PFS)	❌	✅	✅ Mandatory
Renegotiation	✅	✅	❌ Removed
MD5, SHA-1 Hashing	✅	⚠️ (Discouraged)	❌ Removed

- Tls vs ssl
- Ssl is predecessor
- Tls works; ssl doesn't
- Ssl doesn't work because
    - No pfs
    - Used to just encrypt then send data
        - Still used certificates signed by ca
        - When a client wanted to establish a secure connection with a server, an SSL handshake occurred:
        - Client Hello: The client sends a request to the server to start a secure session.

        - Server Hello: The server responds with its digital certificate and public key.
        - Session Key Generation: The client and server use the public key to securely exchange a session key, which is used to encrypt the communication between them.
        - Secure Data Exchange: Once the session key is established, all further communication is encrypted with that session key.
    - SSL 2.0 and 3.0 had several weaknesses in their encryption algorithms and cipher suites.
    - Weak ciphers like RC4 were supported, which are susceptible to attacks.
    - SSL 3.0 was found to be vulnerable to attacks like the BEAST attack, which made it easy for attackers to decrypt encrypted traffic.
    - 2. Deprecation of SSL 2.0 & 3.0
    - SSL 2.0 was deprecated in 2011 due to its severe security flaws, such as a lack of proper message authentication and weak ciphers.
    - SSL 3.0, although an improvement over 2.0, still had several weaknesses:
    - Vulnerable to the POODLE attack (Padding Oracle On Downgraded Legacy Encryption), which could allow an attacker to decrypt SSL 3.0 traffic using a chosen ciphertext attack.

Common ports

- Common port names for different networking services.
- HTTP: 80, HTTPS: 443, FTP: 21, SSH: 22, DNS: 53, SMTP: 25 (465 for SSL and 587 tor TLS), POP3: 110, IMAP: 143, MySQL: 3306, PostgreSQL: 5432, NFS: 2049, RDP: 3389, quic 443
- 1443 for MSSQL, Kerebos 88, 389/636 for LDAP, telnet: 23, Redis (TCP 6379), ES (TCP 9200, 9300), SMB (TCP 445), Syslog (514), 161/162 for snmp

Cryptography

- Encryption is protecting data either symmetric or asymmetric keys by scrambling data (with cryptography, which is math that focuses heavily on modular arithmetic)
-
- Todo: cryptography (pblic key), asymmetric encryption, quic, tcp , firewalls, tls handshake (changecyberspec x2 at end forgot), why is http not secure

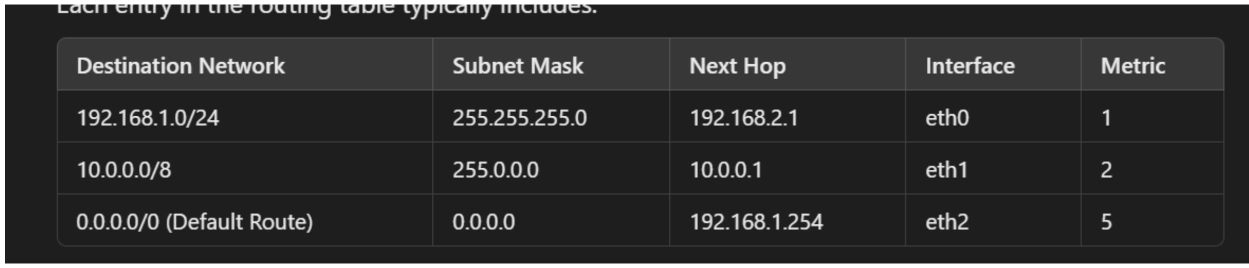
Misc

Troubleshooting scenarios

- Packet loss / connection issues -> ping
    - Then if need to debug further and connection routing may be weird, do traceroute. Can do mtr if need to monitor packet loss at same time over time (continuous) instead of just once
- Dig - to see what dns resolution is happening. Nslookup is less specific but more widely available (like not just linux)
    - Nslookup is interactive

- See what ports are lisenintg - ss
- What is my network interface, mac? Ifconfig, arp
- Something wrong with my packets themselves - tcpdump
- Want to see http request response/headers -> curl
- Looka t firewalls -> iptables
  - Tells you which firewall rules you have and what routes they're active in
- ### Troubleshooting schema
- Check physical connections. Ensure all cables, routers, and other hardware are properly connected.
-
- Ping test. Use the ping command to test whether or not your network stack is working.
-
- Check IP configuration. Perform the ip or the ifconfig commands to check that your network interface is correctly configured.
-
- Check DNS resolution. Execute the nslookup or the dig commands to verify your DNS server settings.
-
- Check routing. Use the ip route show command to show and verify the routing table.
-
- Firewall. Perform the ufw or the iptables commands to temporarily disable your firewall to see if it's blocking traffic.
-
- Check network services. Ensure that the necessary network services are running properly.
-
- Network interface reset. Restart the network interface to resolve some network issues.
-
- Check logs. Check system logs to find network-related errors that shed light on the issue.
-
- Use network monitoring tools. Use these tools to pinpoint issues at the packet level.
- ### Subnets
- Smaller subnet masks (like /16, /8) cover more addresses but are less specific.
-
- Larger subnet masks (like /24, /32) cover fewer addresses but are more precise.
- The longest (most specific) prefix match is chosen. So larger subnet mask + prefix match
-
- /16 means first 16 bits are fixed;
- ### Networks
- MAC Address (Device Level) < IP Address (Network/Device Level) AND Subnet (Part of the Network) < LAN (Local Network) < WAN (Wide Network)

- Lan
    - Higher bandwidth/less latency but shorter distance covered
    - Uses switches usually
    - No need for gateway
- Routers
    - Routers are gateways from your local network to other networks
        - Important so don’t have to keep network tables to connect to other routers/networks around the world in your pc. Less overhead for ur pc
        - Thus it knows to go to router x to eventually get to ip address Y
- Casting
    - Unicasting - one to one
    - Anycasting - send to any
    - Broad - send to all
    - Multi - send to subset of all
- Subnet vs switch vs hub
    - Subnet is a logical subdivision in ip network
        - Is a broadcast domain so can help mitigate too big broadcast
        - Is smaller than a lan
        - Is helpful to define an ip further than just the ip of that general area
            - Subnet mask used to define subnet. is a 32 bit mask that masks ip address to identify which specific area in general area
    - Switch - better
        - Get mac of intended recipient. Forwards packet to mac
        - Full duplex (not half) so no collisions when send msg to each other at same time
        - Faster and less bandwidth than hub
        - Send only to intended recipient
    - Hub - bad, obselete
        - Repeats whatever receives to all other nodes. Easier to implement
        - Can cause collisions (where two devices send msg to each other at same time)
        - Security - all can see the msgs from hub
        - Inefficient since broadcast everything at once
    -
- Network topology - physical/logical arrangement of devices (nodes) in network
    - Star - all devices connect to hub/switch. Most common
        - Single point of failure and expensive for large networks due to requiring lots of cabling
        - Easy to install/troubleshoot. centralized
    - Ring - all in circle
        - Easy to install. Good for traveling one direction
        - Difficult to add/remove, troubleshoot. One point of failure
    - Bus - a line. All hear but only the one msg was intended for listens
        - Simple, inexpensive

        - Single point of failure
        - Increase devices = slower (more collisions, traveling)
    - Mesh - all connected
        - Redundant (can survive failures). Reliable
        - Expensive and complex
    - Tree - connected to hub/switch and travel in one direction downwards in multiple paths
        - Scalable, easy to expand, hiearligical
        - Single point of failure
    - Hybrid - combination. Often more complex but combines strengths
- Asymmetric clustering
    - Nodes in network perform different tasks
        - Easier to manage, resource optimization, failure tolerant (standby can take over, but complex since need to dedicate a standby for each job)
            - Uneven workload, bottlenecks if one tha tis unique is too slow
    - Symmetric - all same tasks in same network
        - Best for even workload, failure tolerant and easy to implement
- Routing tables and how routers work
    - Router works at networks level (lvl 3)
    - What it does:
        - Receives packet
        - Gets destination ip address
        - Looks up routing table, and finds next best hop
        - Sends packet on way
- Network table
    - 

      Each entry in the routing table typically includes:

Destination Network	Subnet Mask	Next Hop	Interface	Metric
192.168.1.0/24	255.255.255.0	192.168.2.1	eth0	1
10.0.0.0/8	255.0.0.0	10.0.0.1	eth1	2
0.0.0.0/0 (Default Route)	0.0.0.0	192.168.1.254	eth2	5

    - **Has destination prefix and next best hop**
    - Interface is the port to send to and meric is cost (lower better)
- Cidr
    - In CIDR notation, the number after the / (like /24 or /16) represents the number of bits that make up the network portion of the IP address. This is called the prefix length.
        - Tells the number that is fixed, so like /3 means first 3 are fixed. Thus represents the prefix since those are the ones we know for sure. Everything else is a wildcard
- Routing packets

Packet routing is a crucial concept in networking, where data (in the form of packets) is sent from a source computer to a destination computer through various network devices like routers, switches, and gateways. The process involves determining the most efficient path for the packets to travel across the network.

How Does the Source Computer Know Where to Route Packets?

When a source computer sends a packet, it needs to know the **next hop** on the way to the destination. The routing decision is based on the following factors:

1. **Destination IP Address**:
   - Every device on a network has a unique IP address. The source computer uses the destination IP address in the packet header to determine where the packet needs to go.
2. **Routing Table**:
   - The source computer checks its **routing table** (or **routing cache**) to find the best route to reach the destination. This table contains entries with information about **destination networks**, **subnet masks**, and **next hop IP addresses**.
   - The routing table is built and updated by various routing protocols (e.g., **RIP**, **OSPF**, **BGP**) or manually configured static routes.
3. **Subnet Mask**:
   - The source uses the **subnet mask** to determine whether the destination IP is in the same local network or a different one. If it's in the same network (i.e., the source and destination IP addresses share the same network portion), the packet can be sent directly to the destination without going through a router.
   - If the destination is on a different network, the packet is forwarded to the default gateway, which is typically a router responsible for sending the packet to the correct network.
4. **Default Gateway**:
   - If the source computer determines that the destination is not in the same local network, it forwards the packet to the **default gateway** (usually a router), which is responsible for routing the packet to the appropriate destination network.
5. **Next Hop**:
   - The source computer's routing table specifies the **next hop**, which is the IP address of the next router that will forward the packet closer to the destination.

How Do Packets Move Across a Network?

Once the source computer determines the correct route for the packet, it is transmitted across the network using various devices that handle the routing process:

1. **Network Interface and Frames**:
   - The packet is encapsulated in a **frame** with link-layer information (e.g., Ethernet MAC addresses). The source computer sends the frame to its **network interface** (e.g., Ethernet adapter).
2. **Switching**:
   - If the source and destination computers are on the same local network, switches forward the frames based on **MAC addresses**. Switches do not examine IP addresses but use **MAC address tables** to forward frames within the same network.
   - If the source computer is on a different network, the packet is sent to a **router**.
3. **Routing**:
   - Routers inspect the **IP header** of the packet to determine the next hop in the network. Based on the **routing table**, the router determines the most efficient route to the destination.
   - If the router knows the destination, it forwards the packet to the next hop. If it doesn't know the destination, it may send the packet to another router, which continues the forwarding process until the destination is reached.
4. **Intermediary Routers**:
   - Routers play a critical role in packet routing. The packet may pass through multiple routers, especially if the destination is far or across different networks (e.g., across the internet).
   - At each router, the IP packet's **destination address** is examined, and the router decides where to forward it based on its own routing table. This process is repeated until the packet reaches the final destination.
5. **Final Destination**:
   - Once the packet reaches the destination network, it is delivered to the correct machine. If it's a local delivery (within the same network), it will be delivered directly to the destination device (based on its MAC address).
   - If the packet is being delivered to a device on another network, it will again pass through routers and switches, ultimately arriving at the destination computer.
6. **Packet Reassembly**:
   - In some cases, packets are fragmented when passing through networks with smaller maximum transmission units (MTU). The destination computer will **reassemble** the fragments into the original packet.

Routing Methods and Protocols

- **Static Routing**: Manual configuration of routes in the routing table. This method doesn't change unless manually updated by the network administrator.

- **Dynamic Routing**: Routing protocols automatically update the routing table based on changes in the network. Common dynamic routing protocols include:

    - **RIP (Routing Information Protocol)**: A distance-vector protocol that determines the best route based on hop count.
    - **OSPF (Open Shortest Path First)**: A link-state protocol that builds a map of the entire network and calculates the shortest path using Dijkstra's algorithm.
    - **BGP (Border Gateway Protocol)**: Used for routing between different autonomous systems (ASes), often in the context of the internet.

Key Concepts in Packet Routing:

1. **Routing Table**: A table that stores route information, specifying the destination network, the next hop, and the cost (metric) of the route.
2. **Hop**: A single step in the route from the source to the destination, usually through a router.
3. **IP Address**: The unique address that identifies a device on a network.
4. **MAC Address**: The hardware address used to route frames within a local network (Data Link Layer).
5. **Gateway**: A device (usually a router) that forwards packets between different networks.

Summary of the Routing Process:

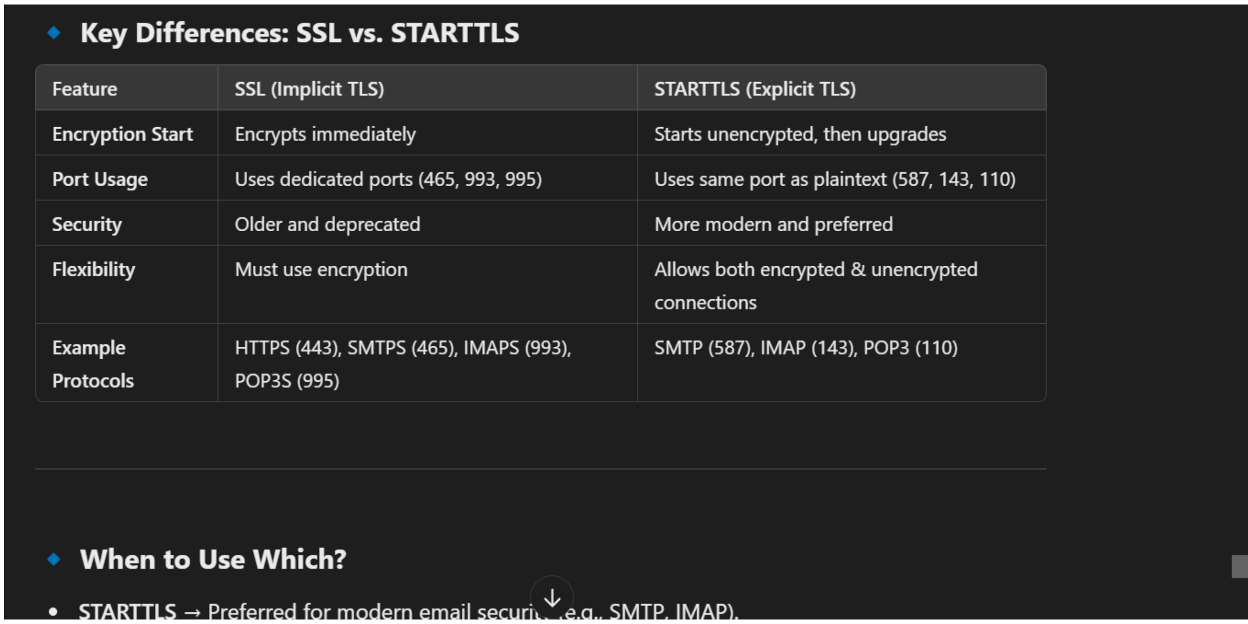
1. The source computer determines whether the destination is on the same network or a different network using the destination IP and subnet mask.
2. If the destination is local, the packet is sent directly; otherwise, it is forwarded to the **default gateway**.
3. Routers forward the packet based on their **routing tables**, examining the destination IP address at each hop.
4. The packet continues to be forwarded until it reaches the destination network.
5. The final device receives the packet and processes it.

By using routing tables, IP addresses, and routing protocols, the source and intermediary devices ensure the packet moves across the network, hop by hop, to reach the correct destination.

    -

- misc
- Port assignment

- Os of client - is dynamically assigned; 49152 to 65535
- Server - known ports as shown in above sect (80 for http, 443 for https)

- Socket vs port
    - Port is nubmered doorway on pc (important for delegating specific protocol or purpose to them). Is just identification for specific service/protocol on a pc
    - Socket is combination of ip address and port number. Uniquely identifies endpoint
- Ssl vs starttls
    - Ssl is used for encryption. Is outdated.
        - Ssl emails include smtps, imaps, pop3s
        - Ehlo -> encrypt with starttls cmd. Negotiate via usual tls (agree on version, ciphers, certificate validity, shared key). Then restart with encryption
    - Updated version is now startls
        - Includes smtp, imap, pop3
    - 

Key Differences: SSL vs. STARTTLS

Feature	SSL (Implicit TLS)	STARTTLS (Explicit TLS)
Encryption Start	Encrypts immediately	Starts unencrypted, then upgrades
Port Usage	Uses dedicated ports (465, 993, 995)	Uses same port as plaintext (587, 143, 110)
Security	Older and deprecated	More modern and preferred
Flexibility	Must use encryption	Allows both encrypted & unencrypted connections
Example Protocols	HTTPS (443), SMTPS (465), IMAPS (993), POP3S (995)	SMTP (587), IMAP (143), POP3 (110)

When to Use Which?

- STARTTLS → Preferred for modern email securi e.g., SMTP, IMAP).

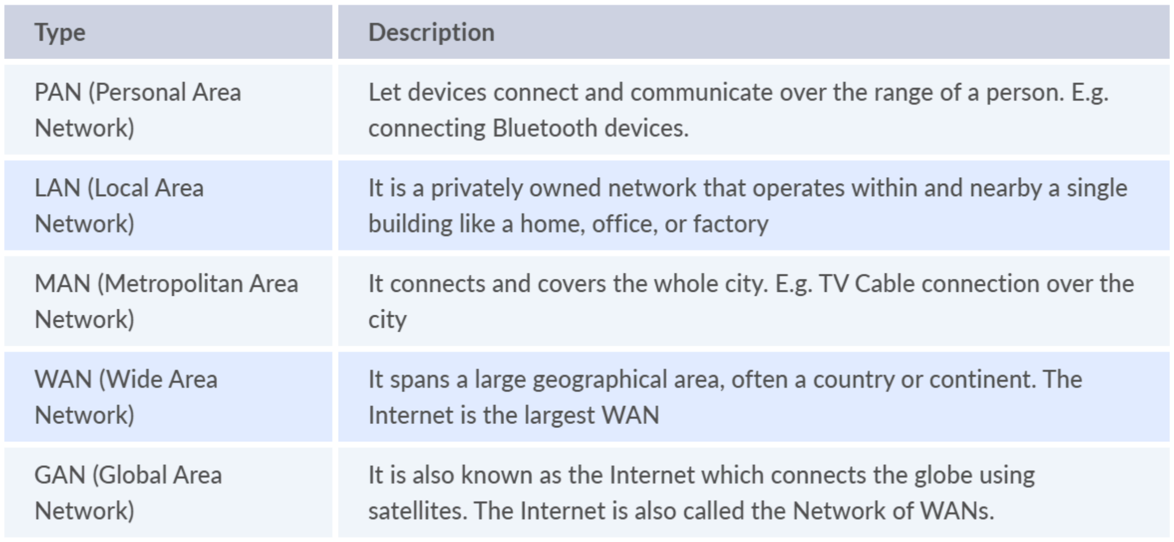
- Cn - common name
    - Field used in tls/ssl certificates and ldap (lightweight directory access protocol) to identify entities
    - Cn is domain name in tls/ssl
        - Can have wildcards to support subdomains
    - In ldap:
        - Represents user, device, service (accompaniesd by ou or organization unit and dc or domain component)
- How certificates work - certificate signed by ca and browser gets certificate and verifies ca signature is leegit against its ca list
    - Let's break down how digital certificates, Certificate Authorities (CAs), and certificate signing work, and then we'll cover what "CN" (Common Name) means in this context.

-
- **1. What is a Digital Certificate?**
-
- A digital certificate is like an electronic ID card. It's a file that binds a *public key* to the identity of an entity (a website, a person, a device). It's cryptographically signed by a trusted third party (a CA), proving the authenticity of the public key.
-
- Think of it as a guarantee: "This public key belongs to this entity."
-
- A certificate typically includes:
-
- * **Subject:** The entity the certificate is issued to (e.g., a website's domain name, a person's name).
- * **Public Key:** The public key of the subject. This is used to encrypt messages sent *to* the subject.
- * **Issuer:** The name of the Certificate Authority (CA) that issued the certificate.
- * **Signature:** A digital signature from the CA, verifying the certificate's authenticity.
- * **Validity Period:** The time period during which the certificate is valid.
-
- **2. What is a Certificate Authority (CA)?**
-
- A CA is a trusted organization that issues digital certificates. They act as a trusted third party, verifying the identity of the certificate holder *before* issuing a certificate. They are like the official ID card issuers.
-
- CAs have several key responsibilities:
-
- * **Identity Verification:** They verify the identity of the entities requesting certificates. This might involve checking domain ownership, business registration, or other forms of identification.
- * **Certificate Issuance:** They issue digital certificates containing the entity's information and public key.
- * **Certificate Revocation:** They revoke certificates that are no longer valid (e.g., if a private key is compromised, or if a website changes ownership).
-
- **3. How Does Certificate Signing Work?**
-
- Here's the process:
-
- 1. **Certificate Signing Request (CSR):** The entity (e.g., a website) that wants a certificate generates a CSR. This file contains the entity's information (name, domain name, etc.) and its *public key*. Critically, the entity *also* generates a

private key at this stage, but the private key is kept *secret* and is *not* included in the CSR. The CSR is like filling out an application for an ID card.

-
- 2. **Submission to CA:** The entity submits the CSR to a CA.
-
- 3. **Identity Verification:** The CA verifies the identity of the entity.
-
- 4. **Certificate Issuance:** If the verification is successful, the CA issues a digital certificate. The CA takes the information from the CSR (including the public key) and adds its own information (issuer, validity period). The CA then *signs* the certificate using its own *private key*. This digital signature is what guarantees the certificate's authenticity. The CA's signature is mathematically linked to its *public key*.
-
- 5. **Certificate Distribution:** The entity receives the signed certificate and can then use it to prove its identity.
-
- **4. How Does Certificate Verification Work?**
-
- When someone (e.g., a web browser) receives a certificate, it verifies the certificate's validity and trustworthiness.
-
- 1. **Obtain CA's Public Key:** The verifier (browser) needs the *CA's public key* to verify the CA's signature. Browsers come pre-loaded with the public keys of well-known CAs.
-
- 2. **Verify Signature:** The verifier uses the CA's public key to *decrypt* the digital signature on the certificate. If the signature is valid, it proves that the certificate was indeed issued by the CA and hasn't been tampered with.
-
- 3. **Check Validity Period:** The verifier checks if the certificate is still within its validity period.
-
- 4. **Check Revocation Status (Sometimes):** The verifier might also check if the certificate has been revoked by the CA. This is done through Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP).
-
- **5. What is CN (Common Name)?**
-
- In older certificates (especially those conforming to the X.509 v1 standard), the `CN` (Common Name) field was used to store the primary identifier of the certificate holder. For websites, this was typically the domain name (e.g., `www.example.com`).
-

    - **Important Note:** The `CN` field is now considered *deprecated* for website certificates. Modern best practices recommend using the `Subject Alternative Name (SAN)` extension instead. The SAN extension allows a certificate to contain multiple domain names, IP addresses, and other identifiers. While you might still see CNs in some certificates, especially older ones, the SAN extension is the preferred way to specify website identities in current certificates. Browsers often check the SAN field first and might display a warning if the CN doesn't match the expected domain.
    -
    -
    -
-
- Diff types of networks

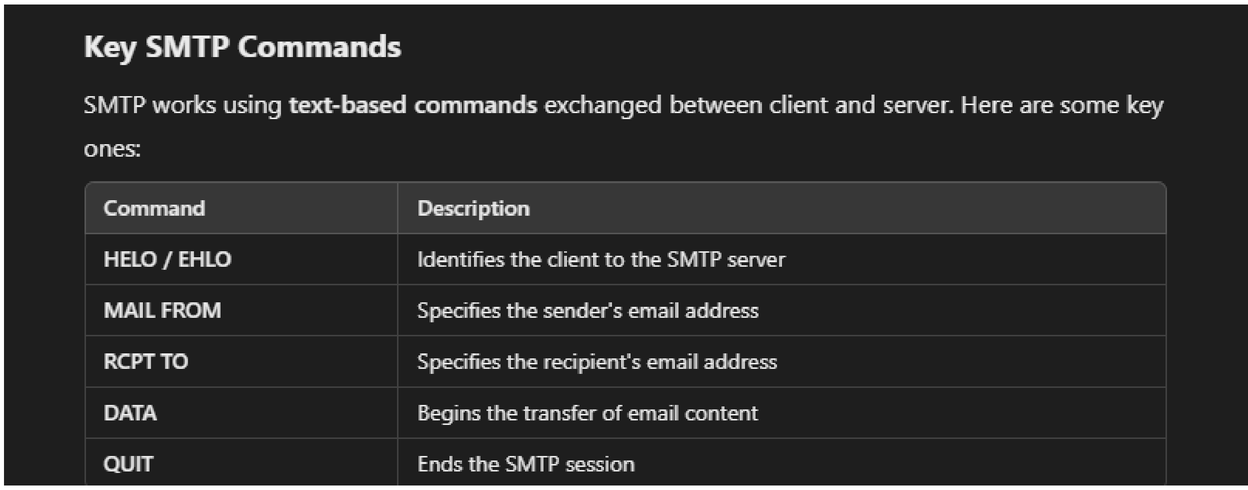


Type	Description
PAN (Personal Area Network)	Let devices connect and communicate over the range of a person. E.g. connecting Bluetooth devices.
LAN (Local Area Network)	It is a privately owned network that operates within and nearby a single building like a home, office, or factory
MAN (Metropolitan Area Network)	It connects and covers the whole city. E.g. TV Cable connection over the city
WAN (Wide Area Network)	It spans a large geographical area, often a country or continent. The Internet is the largest WAN
GAN (Global Area Network)	It is also known as the Internet which connects the globe using satellites. The Internet is also called the Network of WANs.

Explain LAN (Local Area Network)

- Vpn
    - Encryption: A VPN encrypts your internet traffic, turning it into unreadable code.
    - Tunneling: Your traffic is routed through a secure "tunnel" (due to encryption) to a VPN server. This tunnel hides your IP address and location. Vpn server receives. (is located wherever ur region is set at)
    - Decryption: The VPN server decrypts your traffic and sends it on to the internet.
- Ldap - quick directory access of people (like tree with lots of metadata)
    - User management, authorization
    - Directory Information Tree (DIT): The LDAP directory is organized in a hierarchical tree-like structure, called the DIT. This makes it easy to organize and navigate the information.
    - Distinguished Name (DN): Each object in the directory has a unique identifier called a Distinguished Name (DN). This is like the full path to the object in the DIT.

    - Attributes: Each object has attributes, which are pieces of information about the object (e.g., username, email, department).
    - In short: LDAP is a powerful tool for managing and accessing directory information. It provides a centralized, standardized way to store and retrieve information about users, computers, and other resources, simplifying administration and enabling application integration

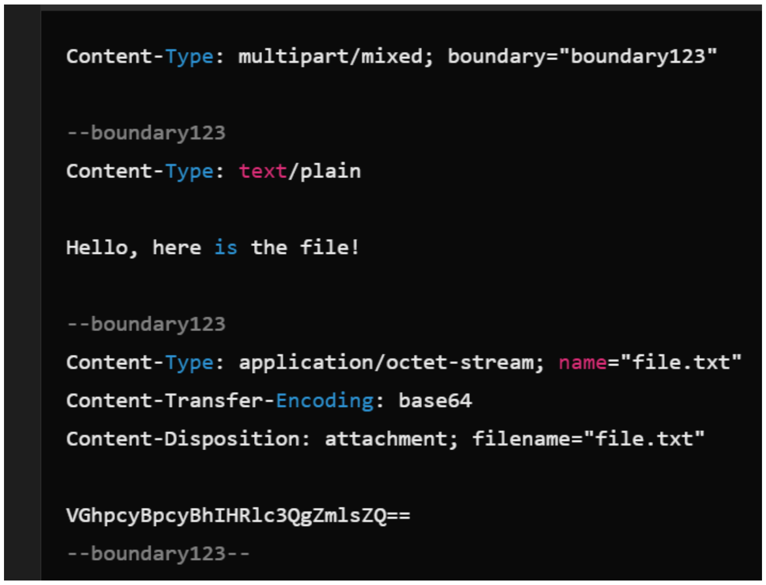
- Email
- Smtp
    - Simple mail transport protocol
    - Send mail as authenticated user and server relays this to recipient
    - Recipient receives via imap (retrieves and syncs across servers) or pop3 (retrieves and deletes across servers)
    - Ports
        - 25 (default and blocked by many isps)
        - 465 (over ssl, deprecated)
        - 587 (startls, recommended/modern)
    - 

Key SMTP Commands

SMTP works using **text-based commands** exchanged between client and server. Here are some key ones:

Command	Description
HELO / EHLO	Identifies the client to the SMTP server
MAIL FROM	Specifies the sender's email address
RCPT TO	Specifies the recipient's email address
DATA	Begins the transfer of email content
QUIT	Ends the SMTP session

- Sending email with smtp - uses tcp
    - Client opens connection with smtp server with tcp connection
    - Either port 587 (starttls) or port 465 (ssl)
    - Send commands to server from client:
        - helo/ehlo - identifies client
        - Mail from - sender’s email address
        - Rcpt to - recipient email address
        - Data - email content
        - Quit - smtp session end
    - Attachments are in mime encoding (base64 usually)
        - Is long string of text with boundaries to separate metadata / parts of email

        - 

```
Content-Type: multipart/mixed; boundary="boundary123"

--boundary123
Content-Type: text/plain

Hello, here is the file!

--boundary123
Content-Type: application/octet-stream; name="file.txt"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="file.txt"

VGhpcyBpcyBhIHRlc3QgZmlsZQ==
--boundary123--
```

    - Smtp relays it to recipients mailbox
- Receiving email
    - Imap - uses tcp
        - Port 993, ssl/tls - keep email on server so can sync across all other client's devices
    - Pop3 - uses tcp
        - Port 993, ssl/tls - downloads email to client and deletes on server
    - IMAP Example:
        - LOGIN: Authenticates the user.
        - SELECT INBOX: Chooses the mailbox.
        - FETCH: Retrieves the email data.
        - LOGOUT: Ends the session.
    - POP3 Example:
        - USER: Specifies the username.
        - PASS: Provides the password.
        - LIST: Lists messages in the mailbox.
        - RETR: Retrieves a specific message.
        - DELE: Deletes a message (optional).
        - QUIT: Ends the session.

GP Route Table:

- **Purpose**: Stores routes learned through BGP (Border Gateway Protocol).
- **Entries**: Contains routes from other networks (outside your local network) that your router has learned via BGP.
- **Content**: Each entry includes the destination network, the next hop, AS (Autonomous System) path, and other BGP-specific info like route preferences.

IP Route Table:

- **Purpose**: Stores all routes your router uses to forward packets, including local network routes, static routes, and routes learned via protocols like RIP, OSPF, or BGP.
- **Entries**: Contains routes to networks (both local and remote) and how to reach them (via the next hop or directly connected interfaces).
- **Content**: Each entry includes destination network, next hop, and how the router should forward the packet (e.g., directly or via a gateway).
- ## Long polling vs sse vs websockets
- Long polling - client makes http request; server keeps this open until expires or data is sent
  - Is like requesting for book from library whenever it is available
  - Used for: monitoring systems, data synch
- Server sent events (sse) - server pushes events to client over http. Unidirectional. Close once unsubscribe
- Websocket - full duplex, real time comms (tcp)
- ## Waf how it filters
- Here's a table summarizing the different attacks (including SQL Injection, XSS, DDoS, and others), along with how a **Web Application Firewall (WAF)** protects against them:

Attack Type	Description	How WAF Protects
SQL Injection	Attackers insert malicious SQL code into an input field to interact with a database.	- **Input sanitization**: Filters and escapes special characters. - **Query parameterization**: Ensures that input is treated as data, not executable code.
Cross-Site Scripting (XSS)	Malicious scripts are injected into web pages, executed by the victim's browser.	- **Input validation**: Ensures that user input is properly sanitized and encoded. - **Output encoding**: Prevents script execution in the browser.
Distributed Denial of Service (DDoS)	Attackers flood the network or application with excessive traffic to exhaust resources.	- **Rate limiting**: Controls the volume of requests from a single source. - **Traffic analysis**: Identifies and blocks high-traffic patterns.
Cross-Site Request Forgery (CSRF)	Malicious scripts trick users into performing unintended actions on a site they are logged into.	- **Anti-CSRF tokens**: Requires valid tokens for form submissions. - **Origin checks**: Verifies that requests come from trusted sources.

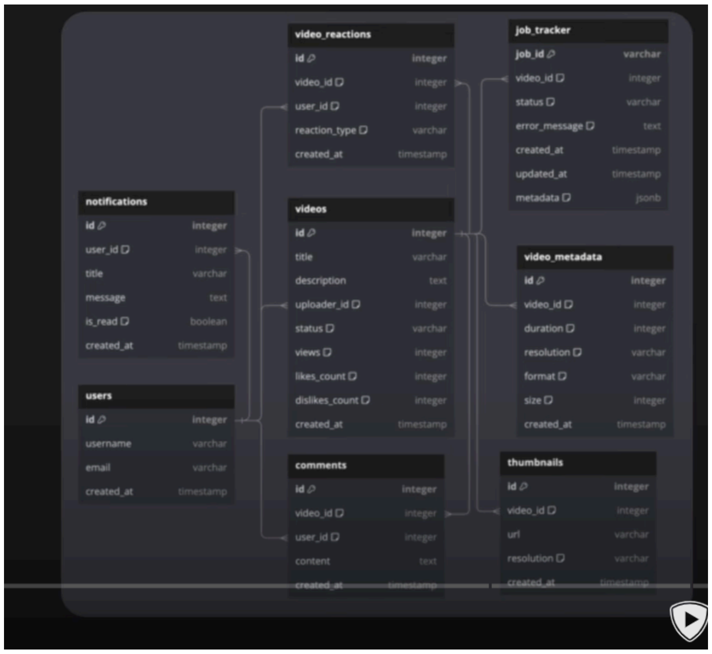
Remote File Inclusion (RFI)	Attackers include remote files from external servers, potentially executing malicious code.	- **File path validation**: Blocks unsafe or external file paths. - **Pattern matching**: Identifies and filters RFI attempts.
Local File Inclusion (LFI)	Attackers exploit the server to include local files, often leading to code execution.	- **File path sanitization**: Filters and normalizes file paths. - **Pattern detection**: Blocks directory traversal attempts (e.g., `../../`).
Command Injection	Malicious input is executed as a system command on the server.	- **Command filtering**: Blocks shell commands by detecting special characters like `;`, `&&`. - **Input sanitization**: Escapes dangerous characters.
Path Traversal	Attackers manipulate file paths to access unauthorized files.	- **Input validation**: Prevents directory traversal (`../../`). - **Normalization**: Strips out unwanted characters or sequences.
HTTP Response Splitting	Attackers manipulate HTTP headers to inject additional responses, leading to hijacking or poisoning.	- **Header validation**: Ensures proper header formatting. - **Response filtering**: Blocks suspicious response manipulations.
Session Fixation	Attacker forces a user to use a specific session ID to hijack their session.	- **Session ID validation**: Blocks manipulation of session IDs. - **Session management**: Ensures session IDs aren't passed via URLs.
HTTP Flooding (DoS)	A form of DoS where an attacker floods the application with HTTP requests to exhaust resources.	- **Rate limiting**: Limits requests from a single IP address. - **CAPTCHA challenges**: Ensures requests are made by legitimate users.

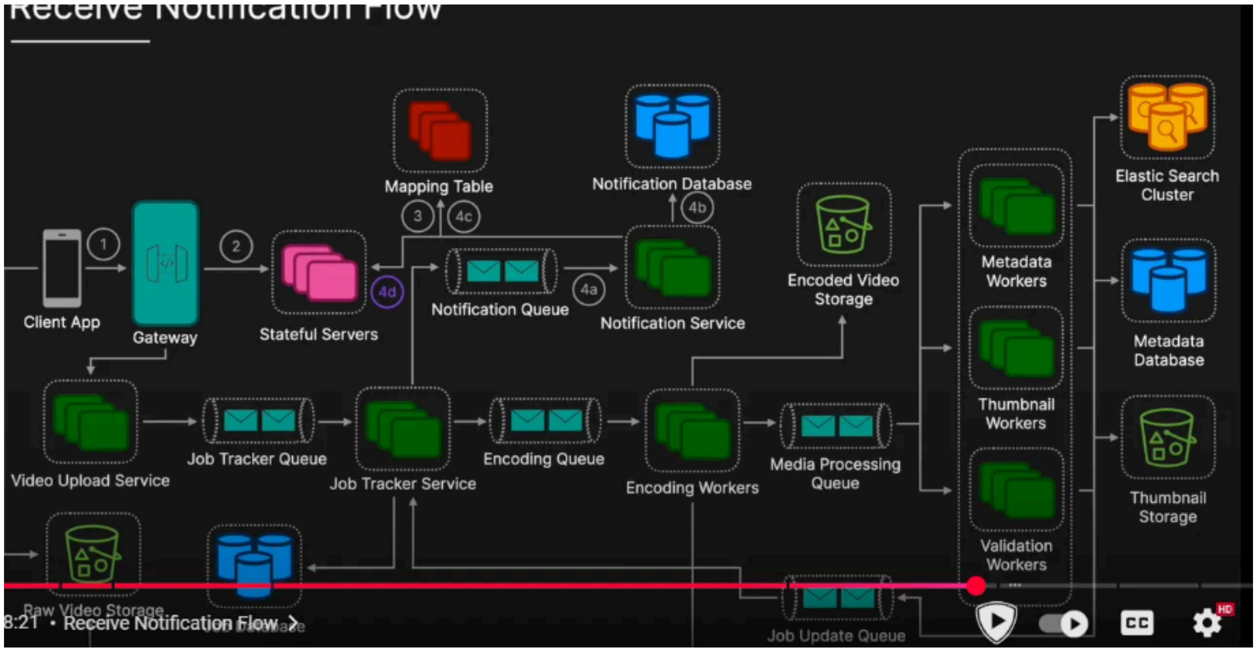
Bot and Web Scraping	Automated bots scrape data or perform unauthorized actions.	- **Bot detection**: Uses CAPTCHAs and behavioral analysis to detect bots. - **Rate limiting**: Restricts excessive request frequencies.
HTTP Verb Tampering	Attackers send unauthorized HTTP methods (e.g., `DELETE`, `TRACE`) to exploit vulnerabilities.	- **Method validation**: Restricts HTTP methods to only supported ones (e.g., `GET`, `POST`). - **Strict policies**: Blocks risky HTTP methods.
Denial of Service (DoS)	An attack overloads a server or its resources, preventing legitimate users from accessing it.	- **Traffic rate limiting**: Prevents excessive traffic. - **CAPTCHA**: Blocks automated requests from scripts or bots.
Insecure Direct Object References (IDOR)	Attackers manipulate input to access unauthorized resources (e.g., changing URL parameters).	- **Input validation**: Ensures that requests for objects are legitimate and properly authorized.
Malicious User-Agent Strings	Attackers use forged user-agent strings to disguise themselves or exploit vulnerabilities.	- **User-Agent filtering**: Blocks known malicious user-agent strings associated with bot traffic or attacks.
HTTP Request Smuggling	Attackers exploit discrepancies in how requests are parsed by different components (e.g., proxies).	- **Request validation**: Ensures standard HTTP formatting and blocks malformed requests. - **Header consistency checks**: Validates headers across layers.

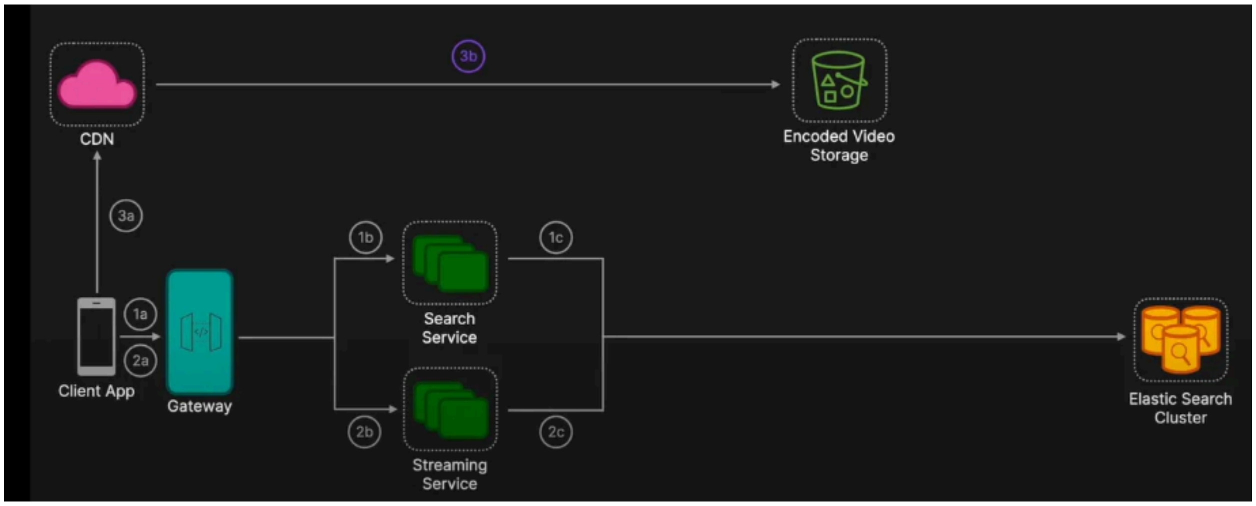
- This table covers how WAFs protect against various attacks, including some commonly known threats like SQL injection, XSS, and DDoS, as well as less common but still critical ones such as CSRF, path traversal, and HTTP request smuggling.
-
- ## System design simple
-

- ## System design tips
- immutable system design is much easier than mutable, so CONSIDER THE IMMUTABLE CASE
    - tackle the dumber case first, then add complexity later
    - avoid concurrency from edits, consistency after edits, reconciliation after failure with immutable case
- ask if there are any requirements you missed
- storage calculation formula
    - storage = daily data by 1 user * daily active user count * length of time to store data
- bandwidth estimation
    - bandwidth per second = daily data used by 1 user * daily active user / total seconds in a day
- there are around 100k seconds per day
- Interviewer will interrupt you if you are going off track
- Ask clarifying questions; prompt will be vague. Try to figure out how many users, if is read/write heavy, scale needed for distributed? (yes probably)
-
- ## Important parts
- Api gateway - routes to proper microservice; combines
- Reverse proxy - level 7 load balancing, security (firewalls), ssl offloading, caching/auth
- Load balancer - more fine grained control over distributing traffic. Level 3, 4, 7
- Cdns - push/pull; ssl offloading/auth
- Forward proxy - protect client from server, like blocking x domains or geolocations. Blocks server from telling ip of client.
- ## Kafka
- Distributed message queue - parallel processing with replication
- Brokers are instances of queue; have each topic, and each topic has a partition aka an instance in a broker
    - Broker -> holds partitions of topics, aka shard of a topic
- Producers put into queue; consumers read from them. Sequential number in queue (offset) represents message so know when read
- Each partition has one leader and 0 or 1 more followers. Replicate and take over if leader fails
- Useful for:
    - Data tracking, recs, location, microservice comms, ops monitoring
- ## Zookeeper (leader follower replication)
- Slower than redis for idempotency tasks; stronger consistency
- Used for single source of truth and decent consistency with replication (kafka, etc.)
    - Used for leader election (in kafka, first who creates ephemeral node becomes leader), config management (in distributed system), distributed locking (like for

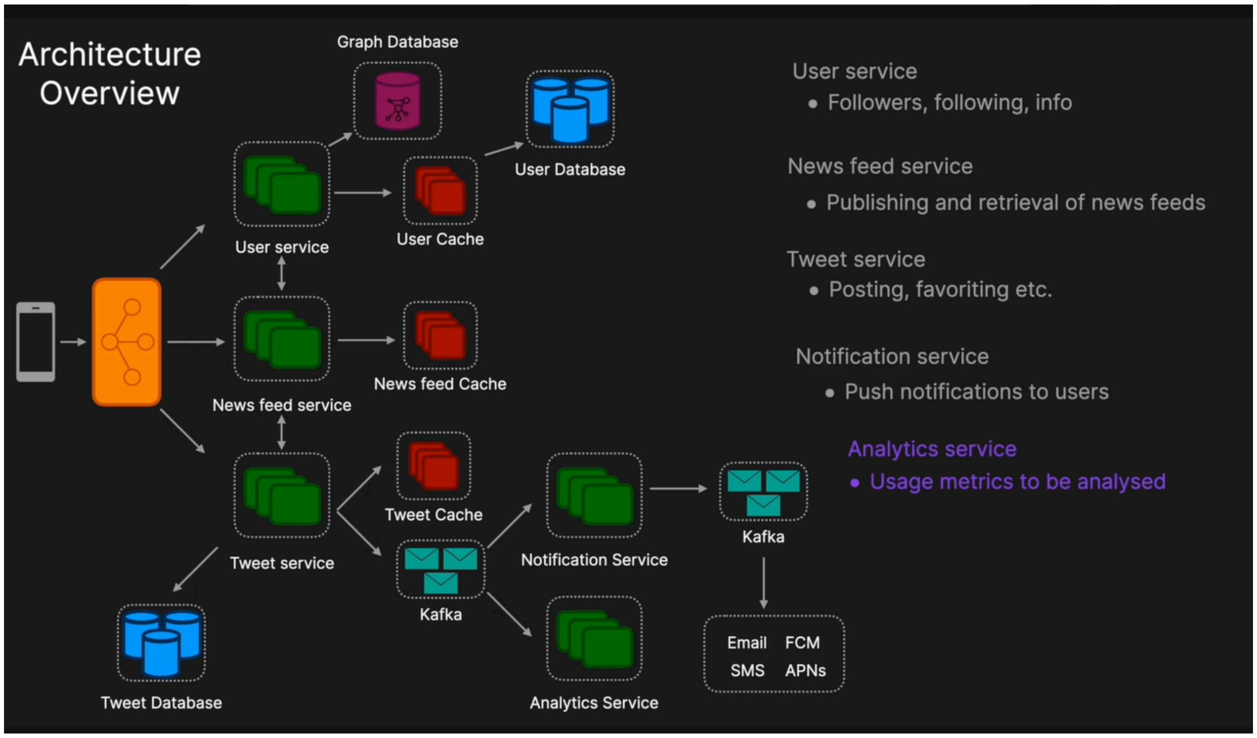
        ticket booking), service discovery (services are ephemeral nodes so only exist when up)
- Watches on znodes - when data changes happen, notify client
- Znodes - nodes in z-tree
    - Persistent - exists until deleted
        - Used for permanent, such as ticket purchases
    - Ephemeral - only exists when client connected. Used for locking
        - Or temporary ticket locking
- Zookeeper has one leader and follower z-node trees
    - Leader writes
    - Followers allow for replicated reads
- Good consistency, high performance, fault tolerance, simplicity
-
- ## Youtube
- Functional - upload, watch, recs/likes, search
- Nonfunctional - low latency, high availability
    - Analytics, securitty, rate limit
- 1 billion users -> each user watches 5/day. 100 watchers : 1 uploader. 1% of 5 billion is uploaded -> 50 million per day. Most are probably not watched
- Profile pic - nosql, eventually consistent
- Database: notifications, users
    - Connect those to video, | comment | video reactions
        - Connect video to thumbnails, job tracker, metadata
            - Job tracker to verify/notify when video done processing
- Load balancer -> App server -> object store -> message queue to encode -> encoded object store
    - # of encoding workers needs to be * how long takes to encode > number of videos uploaded in a second so can get through backlog, depending on encoding time
    - Add cache for app server
    - Nosql for metadata for search
- Cdn (push/pull) for caching based on location for videos (faster)
    - Load video via chunks
    - If livestream, then do in udp. Otherwise, tcp + http for chunks
- Recs - content-based, collaborative
    - Content - based on movie detail attributes and what you like. Need both of those data points
    - Collaborative - recommend what other users have watched with similar like histories. Not always accurate, as people’s tastes are not one for one
- Cache for search?
- Search - elasticsearch; uses inverted index to search; or vector space model and use bag of words association (associate movie keywords + title as bag of words, then search based on that)
- Users, movie metadata - sql

- Keep track of history - nosql
- Keep track of last watched - key-value cache and then eventually nosql, browser cache
- Adaptive bitrate streaming - encode and split up video in different quality. Create manifests (playlists) of these
    - Abr algorithm - decides buffering and encoding based on bandwidth
        - Calculates bandwidth and determines how much it should buffer to ensure low latency but high quality with given bandwidth. May down/upgrade encoding based on this + desired buffer
    - Manifest parser - reads manifest sent to understand options
- 

- 

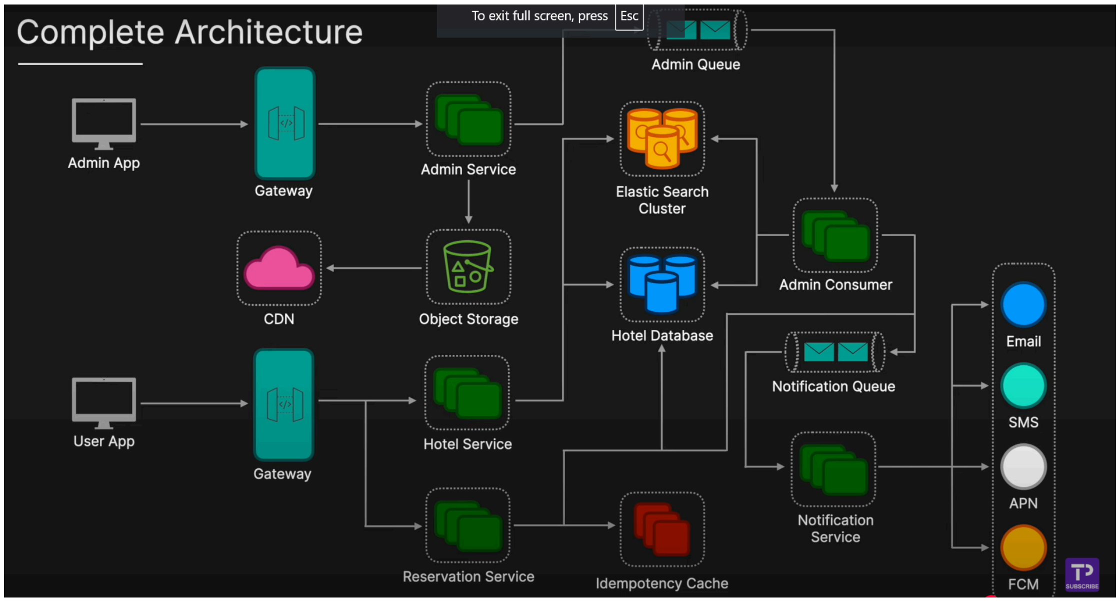
- 

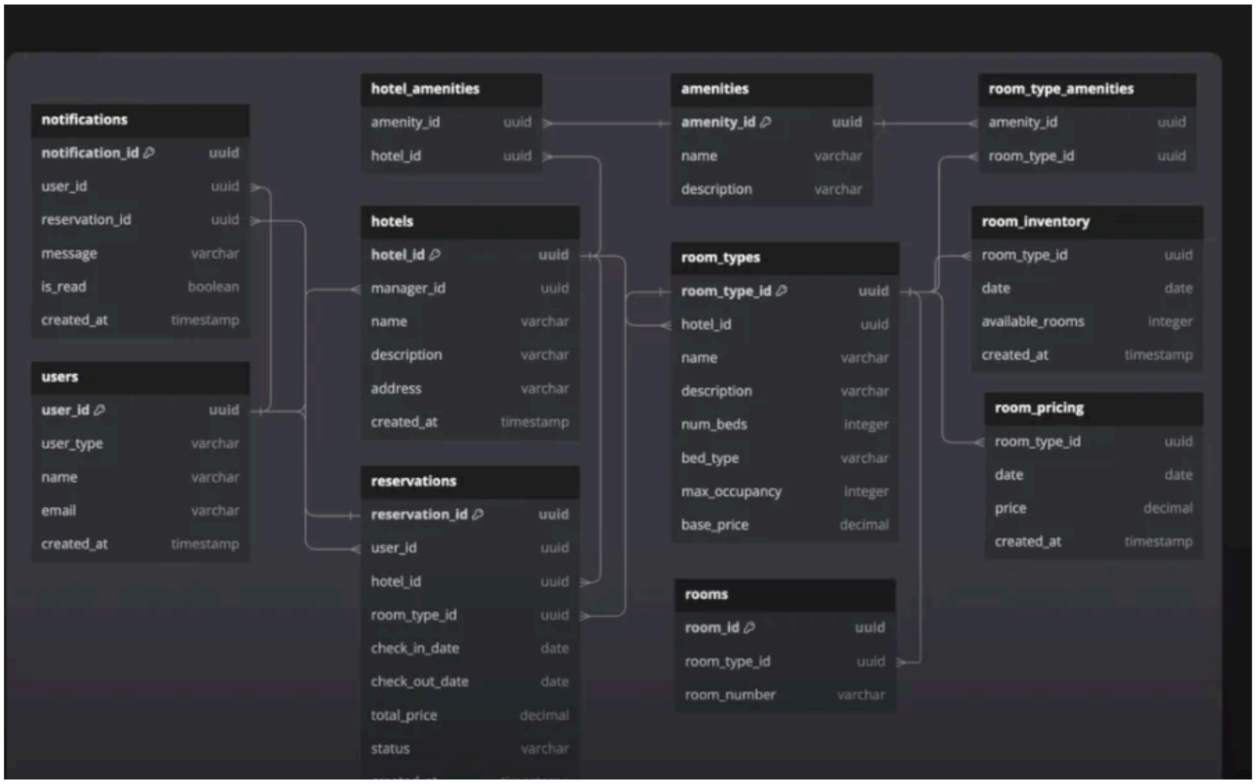
-
- ## Twitter/threads
- Functional: create, follow, view tweets in feed, reply (thread)
- Nonfunctional: high availability (what about ppl with lots of followers)
- How much data read per day? Not much write, but let's say each tweet is 1 mb. And have 1 billion users who read 20 tweets per day. 20 * 1 * 1000 -> 20 PB
    - If reada lot, eventual consistency is good
- graphDB is good for follower
- Read replica for tweets + master-master + sharding due to massive amounts of reads
    - (500 million erads per day / 100000 seconds in a day = 5000 reads/sec, a lot
    - Shard based on user id; users only care about x users (users only follow specific people). If did based on tweet id, more recent tweets have higher traffic. Not spread out
    - Add caching for feed

- Generate feeds async
    - When upload, put into message queue (pub/sub) that feeds into spark kluster, that processes and puts into feed cache
- Client -> load balancer -> app servers -> cache | relational db for users/tweets
    - Images, profile pics, vidoes stored in cdn (tied to object storage)
- 

- ## Booking
- Functional: make entry, book, search, cancel, notifications
- Nonfunctional: consistency, low latency
- Database: user, hotel, room type, reservations, room inventory -> amenities, prices
    - user -> reservation | hotel | room types | room amenities
    - Room inventory (num) connected to room types + price table
        - Rooms connected to room types
- Locking mechanism
    - Sql locking for small scale
    - Queue based booking if no timeout (has some delay)
    - Time based in redis, but temporarily may lock when no purchase and then just expire (wiat)
    - Distributed locking - extra infrastructure needed, but prevents race conditions
- Add hotel flow: gateway/load balaancer -> app server -> admin queue -> admin consumer -> sql db && elasticsearch -> notification queue, notification service
    - Imgs -> cdn
    - Search -> eventual consistency. Could do double commit but is hard to implement
- Reservation flow: gateway, hotel service->database && cluster,
    - Shard by id

- Idempotency cache to prevent duplicate bookings
  - Notification queue, notification service

- 

- 

notifications	
notification_id	uuid
user_id	uuid
reservation_id	uuid
message	varchar
is_read	boolean
created_at	timestamp

users	
user_id	uuid
user_type	varchar
name	varchar
email	varchar
created_at	timestamp

hotel_amenities	
amenity_id	uuid
hotel_id	uuid

hotels	
hotel_id	uuid
manager_id	uuid
name	varchar
description	varchar
address	varchar
created_at	timestamp

reservations	
reservation_id	uuid
user_id	uuid
hotel_id	uuid
room_type_id	uuid
check_in_date	date
check_out_date	date
total_price	decimal
status	varchar

amenities	
amenity_id	uuid
name	varchar
description	varchar

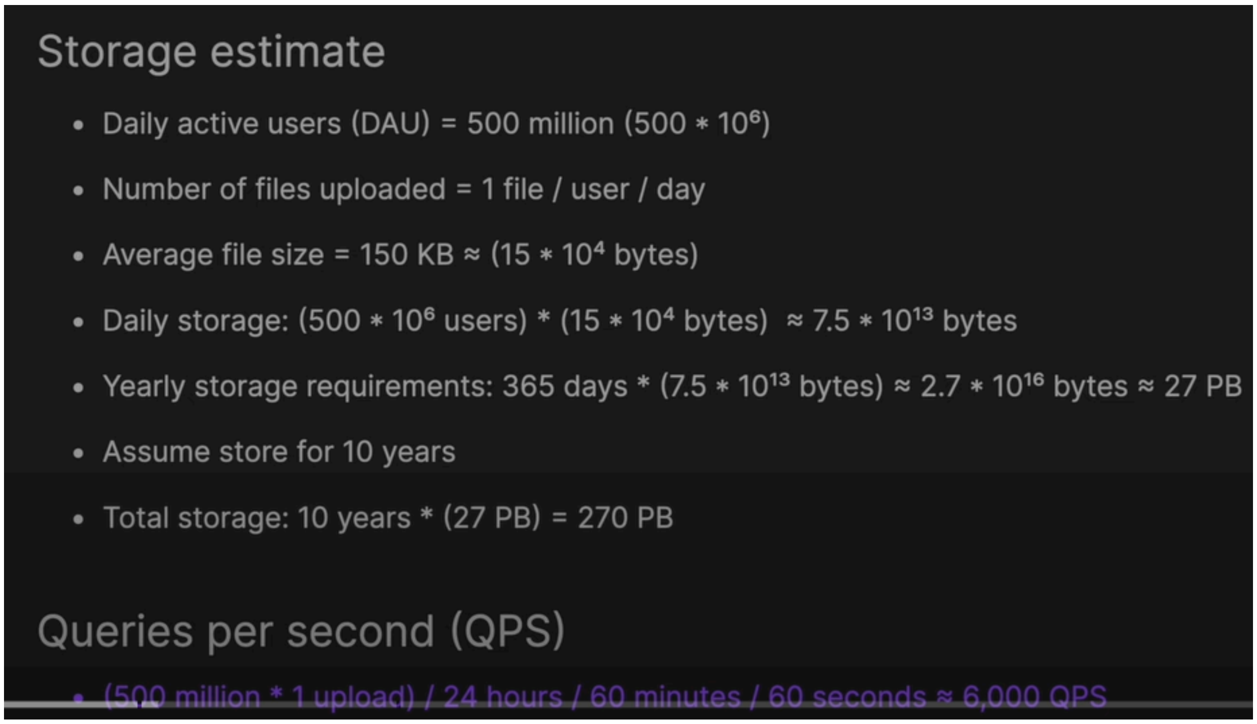
room_types	
room_type_id	uuid
hotel_id	uuid
name	varchar
description	varchar
num_beds	integer
bed_type	varchar
max_occupancy	integer
base_price	decimal

rooms	
room_id	uuid
room_type_id	uuid
room_number	varchar

room_type_amenities	
amenity_id	uuid
room_type_id	uuid

room_inventory	
room_type_id	uuid
date	date
available_rooms	integer
created_at	timestamp

room_pricing	
room_type_id	uuid
date	date
price	decimal
created_at	timestamp

- File sharing
- 

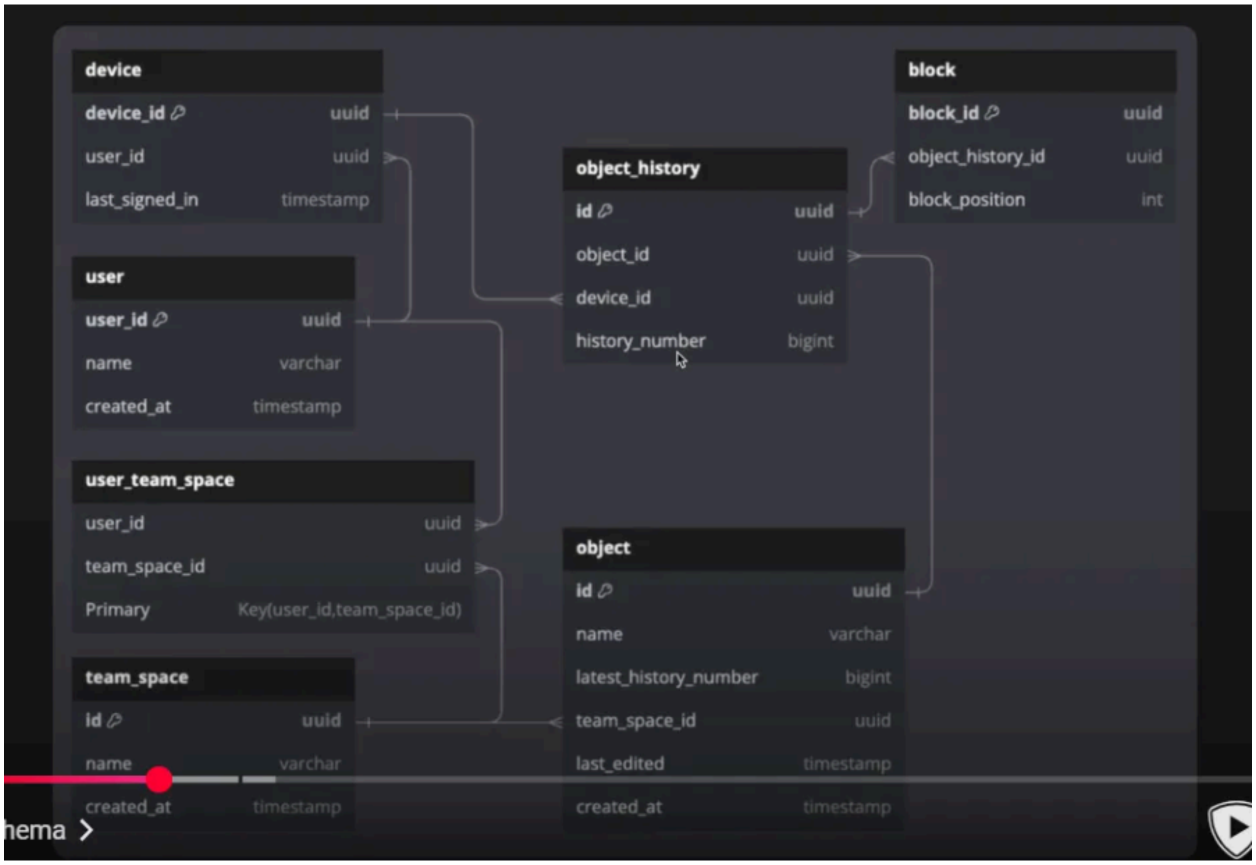
Storage estimate

- Daily active users (DAU) = 500 million ($500 * 10^6$)
- Number of files uploaded = 1 file / user / day
- Average file size = 150 KB ≈ ($15 * 10^4$ bytes)
- Daily storage: ($500 * 10^6$ users) * ($15 * 10^4$ bytes) ≈ $7.5 * 10^{13}$ bytes
- Yearly storage requirements: 365 days * ($7.5 * 10^{13}$ bytes) ≈ $2.7 * 10^{16}$ bytes ≈ 27 PB
- Assume store for 10 years
- Total storage: 10 years * (27 PB) = 270 PB

Queries per second (QPS)

- (500 million * 1 upload) / 24 hours / 60 minutes / 60 seconds ≈ 6,000 QPS

- Storage needed = users * average user upload rate * average file size
  - Then for 1 yr, 10 yr?
- Functional: store + share files
- Nonfunctional: availability, security
- Database:
  - Device and user
    - Connect device to object history (for tracking history of object; one history per diff version of object; only keep track of objects that changed), which is associated with blocks (part of a file)
      - Associate device so we know who made what change
    - Connect user with user team space, which is connected with team space, which objects are associated with



device	
device_id	uuid
user_id	uuid
last_signed_in	timestamp

user	
user_id	uuid
name	varchar
created_at	timestamp

user_team_space	
user_id	uuid
team_space_id	uuid
Primary	Key(user_id,team_space_id)

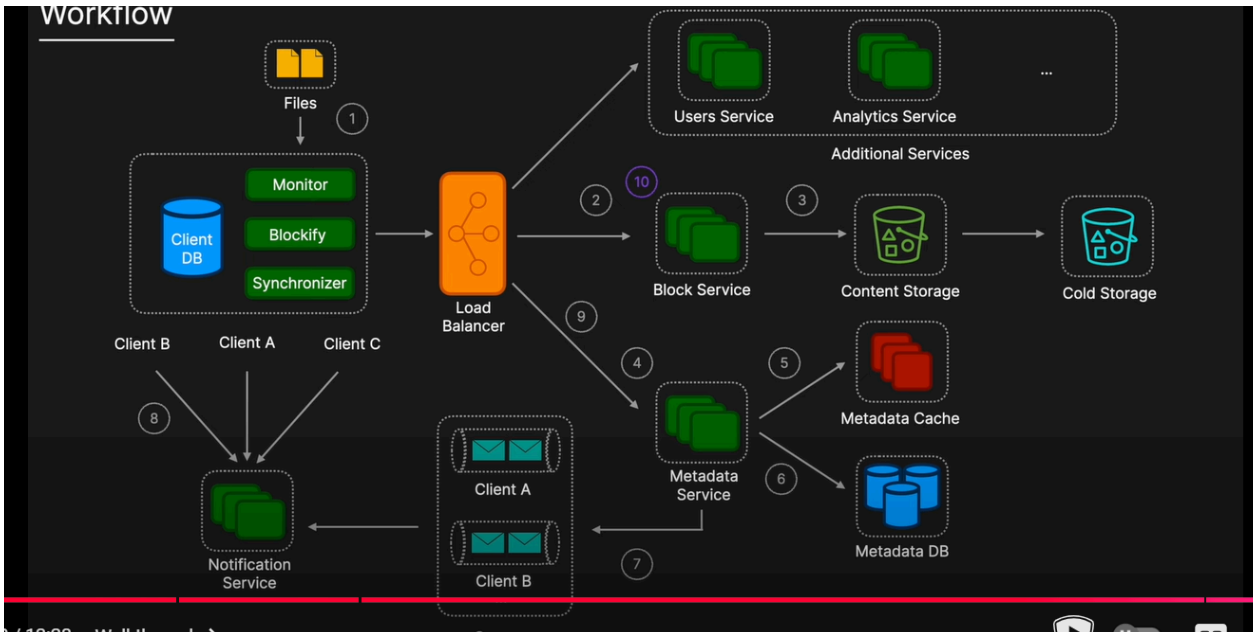
team_space	
id	uuid
name	varchar
created_at	timestamp

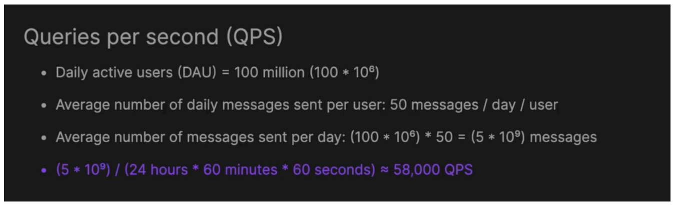
object_history	
id	uuid
object_id	uuid
device_id	uuid
history_number	bigint

object	
id	uuid
name	varchar
latest_history_number	bigint
team_space_id	uuid
last_edited	timestamp
created_at	timestamp

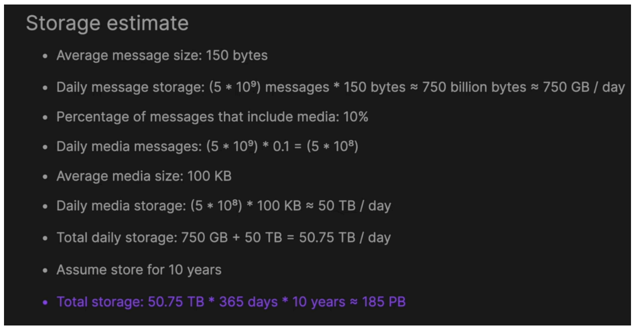
block	
block_id	uuid
object_history_id	uuid
block_position	int

- Files have monitor service, which checks for changes. Notifies blockify (splits file into smaller blocks) and then synchronizer (relay to db)
    - Goes to load balancer, then user services/analytics services
    - Then to block service, which stores in content storage, and compress in cold storage
    - Also adds to metadata service about blocks -> caches -> metadabase db
        - Push onto queue and then notification service
- Add'l features
    - Remove dupe items, encryption of blocks, multi datacenter (cdn)

- 

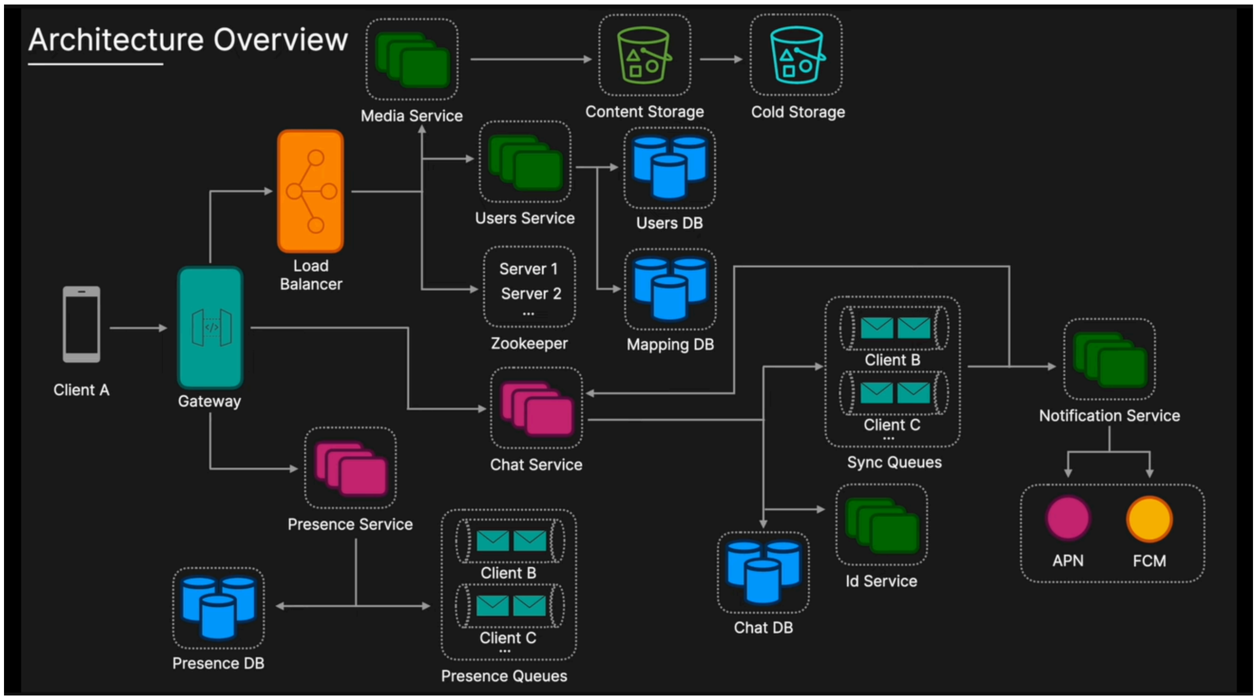
- Messaging
- Functional - 1 on 1, group chats (max 150), media sharing, notifs, online/offline indicator
- Nonfunctional - low latency, availability, scalability
- QPS, storage
    - 

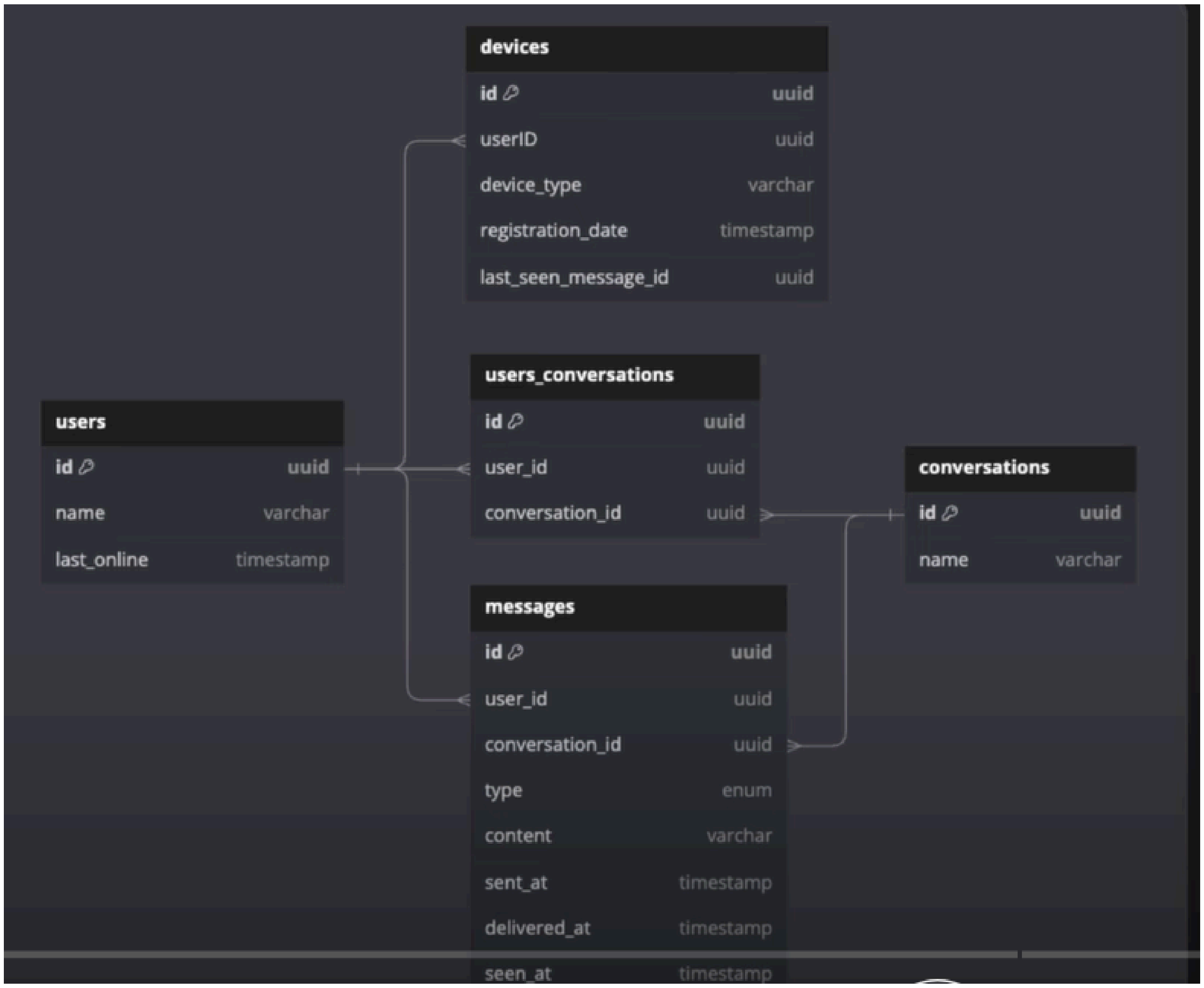
      Queries per second (QPS)

      - Daily active users (DAU) = 100 million (100 * 10^6)
      - Average number of daily messages sent per user: 50 messages / day / user
      - Average number of messages sent per day: (100 * 10^6) * 50 = (5 * 10^9) messages
      - (5 * 10^9) / (24 hours * 60 minutes * 60 seconds) ≈ 58,000 QPS
    - 

      Storage estimate

      - Average message size: 150 bytes
      - Daily message storage: (5 * 10^9) messages * 150 bytes ≈ 750 billion bytes ≈ 750 GB / day
      - Percentage of messages that include media: 10%
      - Daily media messages: (5 * 10^9) * 0.1 = (5 * 10^8)
      - Average media size: 100 KB
      - Daily media storage: (5 * 10^8) * 100 KB ≈ 50 TB / day
      - Total daily storage: 750 GB + 50 TB = 50.75 TB / day
      - Assume store for 10 years
      - Total storage: 50.75 TB * 365 days * 10 years ≈ 185 PB
- Client -> gateway -> load balancer -> user service -> users db | media service -> content storage && cold storage
- Chat service -> websockets (no long polling since may open too long + http connectoin) -> chat db (nosql; need to write quickly)
- Need message id to be unique so that way any new messages > last seen id for chat = unseen
- Database
    - Users, devices, conversation + name, group chat participants, messages (associated with user, convo)

- Notif queue for each user (max 150 users)
- Heartbeat for presence; service handles this + store in db
- 

- 

devices	
id	uuid
userID	uuid
device_type	varchar
registration_date	timestamp
last_seen_message_id	uuid

users_conversations	
id	uuid
user_id	uuid
conversation_id	uuid

users	
id	uuid
name	varchar
last_online	timestamp

conversations	
id	uuid
name	varchar

messages	
id	uuid
user_id	uuid
conversation_id	uuid
type	enum
content	varchar
sent_at	timestamp
delivered_at	timestamp
seen_at	timestamp

- ## pacelc
- cap theorem is replication but then if we have network failure, aka partition, cutting off servers from others, then we either have availability or consistency
- Then pac elc: if we don't have a network partition, then we either have low latency but no consistency or high consistency high latency
- ## Consistent hashing
- Is hash buckets in a ring. Bucket is section in ring
- Each bucket has subset of servers assigned to it; server is assigned multiple times (called a virtual node)
    - Thus, server A can be at point 1, 3, 5, 7 for example
- Data is assigned to its bucket. In bucket, it is assigned clockwise from its hashed position, handling conflicts well
- If have to resize, only resize bucket not whole thing, leading to better resizing times
    - Don’t have to move items much or far
- Good because:
    - Resize only section (when have to add nodes/remove them to change hashmap size)

    - Even distribution due to spreading servers across many ring sections, circular hash space
    - Failover - easy to reasign and not repartition much when nodes fail (similra to resize point)

- ## Monetization
- Who are users?
- Would they want it for free (meaning ads, take from vendors, etc.) or are they buying something?
    - Or freemium (free but with paid features), or $$ for faster?
    - Premium shoutout for vendors?
    - Vendor transaction fee
    - Transaction fees from user
    - Sell data
    - ads
- Continuous updates -> subscriptions
- Little updates -> Licesnes?
- ### Technical
- idempotency cache - prevents dupe purchases
- If fail, add to queue w/next avail time (time queue)
    - Exponential backoff to try again
- 3rd party service for payment