CN

1. **What is the difference between Private IP and Public IP?**

- A **Private IP address** is used within a local area network (LAN) to identify devices internally. These IPs are not routable on the internet and are reserved for internal communication. Examples are ranges like `192.168.x.x`, `10.x.x.x`, and `172.16.x.x – 172.31.x.x`.
- A **Public IP address** is globally unique and assigned by ISPs. It allows devices to communicate directly over the internet. Public IPs are routable, meaning packets with these addresses can travel across different networks worldwide.
  For example:
  When I connect my laptop to Wi-Fi at home, my laptop is assigned a **Private IP** (e.g., `192.168.1.5`) by the router. However, when I access Google, the request goes out using my ISP-assigned **Public IP** (e.g., `103.25.210.15`). Google's servers see the public IP, not the private one.

This is typically managed through **NAT (Network Address Translation)** on the router, which maps internal private IPs to the single public IP for internet access.

- ***Q1: Why do we need private IP addresses if public IPs exist?***
  - Because IPv4 addresses are limited. If every device required a public IP, we would have exhausted them much earlier. Private IPs allow multiple devices to share a single public IP via NAT, which conserves address space.
- ***Q2: Can two devices in different private networks have the same IP?***
  - Yes. For example, my laptop at home and a server in a company LAN can both have `192.168.1.10` as a private IP. Since private IPs are not routed over the internet, they don't conflict globally. But within the same LAN, IPs must be unique.
- ***Q3: How does NAT come into play here?***
  - NAT (Network Address Translation) is used on routers to map private IPs to the router's public IP when accessing the internet. For instance, if three devices inside a home network (`192.168.0.2`, `.3`, `.4`) all open

Google, NAT rewrites their requests to use the single public IP. The router keeps a mapping table so responses are sent back to the correct device.

- ***Q4: What about IPv6? Do we still need private vs public IPs?***
  - In IPv6, the address space is so large that theoretically every device can have a unique public IP. But private addressing still exists (`fc00::/7` range) for internal communication and security reasons.

2. **What happens when you type `www.google.com` in the browser and press Enter?**

Below are the steps that are being followed:

- **1. Browser cache check (Content cache):**
  - The browser first checks if the content for `www.google.com` is already cached locally and still valid (based on HTTP cache headers like `Cache-Control` or `Expires`). If yes, it loads directly without going to the network.
- **2. DNS Lookup (Browser & OS cache first):**
  - Browser checks its own DNS cache.
  - If not found, it asks the OS. The OS may have it cached too.
  - If still not found, the OS requests the configured **DNS resolver** (usually ISP or public like 8.8.8.8).
  - DNS query is typically sent via **UDP port 53**. If response is too large, TCP may be used.
  - Resolver checks step by step (Root → `.com` TLD → Google's authoritative DNS) and finally returns the IP (e.g., `142.250.183.100`).
- **3. TCP Connection Setup (Three-way handshake):**
  - Browser initiates a **TCP connection** with the resolved IP on port **443** (for HTTPS).
  - TCP uses the **three-way handshake**:
    1. Client → SYN
    2. Server → SYN-ACK
    3. Client → ACK
- **4. TLS Handshake (for HTTPS):**
  - Since Google uses HTTPS, after TCP handshake, a **TLS/SSL handshake** happens:

    - Client and server agree on encryption algorithms.
    - Server sends its **SSL certificate** (issued by a trusted CA).
    - Client verifies certificate validity.
    - A session key is exchanged (via Diffie-Hellman or RSA).
    - Now a secure, encrypted channel is established.
- **5. HTTP Request:**
    - The browser sends an **HTTP request** (usually HTTP/2 or HTTP/3 for Google):
    - `GET / HTTP/2 Host: www.google.com User-Agent: Chrome/… Accept: text/html,application/xhtml+xml`
- **6. Server Processing:**
    - Google's load balancers and web servers receive the request.
        - A **reverse proxy** (like Google Frontend) routes it to the appropriate backend.
        - The server fetches data (HTML, CSS, JS, images) from Google's infrastructure.
        - An **HTTP response** is sent back. Example:
    - `HTTP/2 200 OK Content-Type: text/html Content-Length: 10500`
- **7. Browser Receives Response:**
    - Browser checks cache headers to decide if it should store the response.
    - It parses HTML, downloads referenced resources (CSS, JS, images) — often in parallel.
    - It may establish multiple TCP/TLS connections, or reuse the same one if **HTTP/2 multiplexing** is supported.
- **8. Rendering Process:**
    - **HTML Parsing** → builds the **DOM tree**.
    - **CSS Parsing** → builds the **CSSOM tree**.
    - **JS Execution** → modifies DOM/CSSOM if needed.
    - **Render Tree** is created (DOM + CSSOM).
    - **Layout** → calculates positions and sizes.
    - **Painting** → fills pixels on the screen.
    - GPU may assist in rendering.
- **9. Connection Handling:**
    - If `Connection: keep-alive` is set (default in HTTP/1.1+), the TCP connection may be reused for subsequent requests.
    - Otherwise, the connection is closed.

- ***Q1: Why does DNS use UDP mostly, not TCP?***
  - Because DNS queries are small, UDP is faster and less overhead. TCP is used only if responses are too large.
- ***Q2: What if DNS fails?***
  - If DNS is unavailable, the browser cannot resolve the hostname to an IP, and you'll see an error like *DNS_PROBE_FINISHED_NXDOMAIN*.

3. **Does DNS contain all records of IP addresses? If no, why?**

No, DNS does **not** contain the record of every IP address in the world. Instead, it's a **distributed, hierarchical system**. The entire DNS database is spread across millions of DNS servers globally, each responsible for specific domains.

If DNS tried to maintain a central record of all IP addresses, it would be impossible to manage due to the sheer scale and constant updates. Instead, DNS works like a directory tree:

- **Root servers** → know where Top-Level Domain (TLD) servers are (like `.com`, `.org`) present.
- **TLD servers** → know which authoritative servers handle specific domains (like `google.com`).
- **Authoritative servers** → contain the actual IP records for that domain.
  So when you look up `www.google.com`, your resolver queries step by step until it reaches the authoritative DNS server for Google, which returns the IP.

Real-world example:
If I type `www.bitmesra.ac.in` in the browser, my system's resolver will first check local DNS cache, then query DNS servers up the hierarchy. Eventually, it reaches the authoritative DNS server for `ac.in`, which gives the IP of BIT Mesra's server. This IP is then cached locally for faster future lookups.

- **What is DNS?**
  - **DNS (Domain Name System):** is the internet's **naming system** that translates human-readable domain names (like `www.google.com`) into IP addresses (like `142.250.182.14`) required by computers to communicate.
- ***Q1: Why not keep one big central DNS database?***

    - Because the internet has billions of devices and domains. A centralized system would be a single point of failure, very slow, and impossible to update in real time. The distributed model ensures scalability, reliability, and redundancy.
- ***Q2: How does DNS handle scalability then?***
    - Through caching and distribution:
        - **Caching**: Once your resolver gets an IP (say, Google's), it stores it temporarily (based on TTL). Next time, it doesn't need to query again.
        - **Distribution**: Each domain is managed by its own authoritative servers. For example, Amazon manages its own DNS records via Route 53.
- ***Q3: What happens if DNS server fails?***
    - DNS has multiple redundant servers. For example, root servers are replicated across the world via Anycast. Similarly, authoritative DNS providers (like Cloudflare, Google, AWS) deploy globally distributed DNS servers to ensure resilience.
- ***Q4: Can DNS records be updated in real-time?***
    - Yes, but updates depend on **TTL (Time To Live)**. If a record's TTL is set to 24 hours, cached resolvers won't see the change until that expires. That's why sometimes when companies migrate websites, DNS propagation can take a few hours worldwide.
- ***Q5: Do Authoritative Servers have all the IP addresses?***
    - No, even **authoritative DNS servers** do not have all IP addresses.
        - An authoritative server only stores records for **the domains it is responsible for**.
        - For example, Google's authoritative DNS servers will store the IPs for `google.com` and its subdomains (like `mail.google.com`), but not for `bitmesra.ac.in`.

4. **Explain the OSI Model in detail ?**

The **OSI (Open Systems Interconnection) model** is a conceptual framework that standardizes how data travels across a network. It has **7 layers**, each with specific responsibilities. Let me walk through them from top to bottom with real-world examples:

- **1. Application Layer**
  - Provides network services directly to the end-user applications.
  - Protocols: **HTTP, HTTPS, FTP, SMTP, DNS, Telnet**.
  - Real-world example: When I open `www.google.com`, the browser uses **HTTP/HTTPS** at the application layer to request the webpage.
- **2. Presentation Layer**
  - Responsible for **data translation, encryption, compression** so that different systems can understand each other.
  - Protocols: **SSL/TLS, JPEG, MPEG, ASCII, Unicode**.
  - Real-world example: When I connect to Gmail over HTTPS, my data is encrypted/decrypted using **TLS** (Transport Layer Security) at the presentation layer before reaching the application.
- **3. Session Layer**
  - Manages and controls **sessions** (establishment, maintenance, termination) between applications.
  - Protocols/APIs: **NetBIOS, RPC, PPTP, SQL session management**.
  - Real-world example: In a **video call (Zoom/Google Meet)**, the session layer keeps track of active sessions so both sides stay synchronized until the call ends.
- **4. Transport Layer**
  - Ensures **end-to-end delivery**, reliability, segmentation, and error recovery.
  - Protocols: **TCP, UDP**.
  - Real-world examples:
    - **TCP**: When downloading a file from Google Drive, TCP ensures reliability and retransmits lost packets.
    - **UDP**: In a **live YouTube stream**, UDP is used since speed is prioritized over reliability.
- **5. Network Layer**
  - Handles **logical addressing (IP addresses)**, routing, and forwarding packets across networks.
  - Protocols: **IPv4, IPv6, ICMP, ARP, OSPF, BGP**.
  - Real-world example: When I send a request to `www.microsoft.com`, the data is routed using the **IP address** of Microsoft's servers. Routers use **BGP** (Border Gateway Protocol) to determine the best path across the internet backbone.

- **6. Data Link Layer**
  - Provides **MAC addressing, error detection, and frame delivery** between two directly connected devices.
  - Protocols: **Ethernet, PPP, Switches, ARP**.
  - Real-world example: When I connect my laptop to Wi-Fi, the access point identifies my device using its **MAC address** (`34:12:98:AB:...`) and ensures frames are delivered correctly over the wireless medium.
- **7. Physical Layer**
  - Deals with **actual transmission of raw bits** over a physical medium (cables, Wi-Fi, fiber optics, radio waves).
  - Examples: **Ethernet cables, Wi-Fi signals, fiber optics, hubs**.
  - Real-world example: When I plug in an **Ethernet cable** or use **Wi-Fi radio waves**, the physical layer is transmitting electrical/optical signals that carry the bits of my request to Google's servers.
- **An example where multiple layers interact.**
  - When I watch a **YouTube video**:
    - Application Layer: Uses **HTTPS** to request video.
    - Presentation Layer: Video compressed using **MPEG**.
    - Session Layer: Keeps the streaming session alive.
    - Transport Layer: **TCP** ensures reliable delivery of chunks.
    - Network Layer: Uses **IP** to route packets.
    - Data Link Layer: Wi-Fi frames delivered via MAC address.
    - Physical Layer: Bits sent as **radio waves** to my router.

5. **Explain these protocols: HTTP, HTTPS, SSL/TLS, BGP**

---

1. HTTP (HyperText Transfer Protocol)

- HTTP is an **application layer protocol** used for communication between a web browser (client) and a web server. It's stateless, meaning each request-response is independent. It defines methods like `GET` , `POST` , `PUT` , `DELETE` for resource access.
- **Real-world usage example:** When I open `http://example.com` , my browser sends an HTTP `GET` request to the server hosting that website. The server responds with HTML/CSS/JS files, which the browser renders into the

webpage.

2. HTTPS (HyperText Transfer Protocol Secure)

- HTTPS is simply HTTP running over an **encrypted channel using SSL/TLS**. This ensures **confidentiality, integrity, and authentication**. Without HTTPS, data like login credentials or payment information would travel in plaintext.
- **Real-world usage example:** When I log into `https://gmail.com`, my username and password are encrypted before traveling across the network. Even if someone intercepts packets using Wireshark, they'll only see ciphertext, not the actual credentials.

3. SSL/TLS (Secure Sockets Layer / Transport Layer Security)

- SSL (now deprecated) and TLS are **cryptographic protocols** that secure communication over a network. TLS works between the **transport and application layers**.
- It uses **asymmetric encryption** (public/private key) for handshake and authentication.
- Then it switches to **symmetric encryption** (faster) for actual data transfer.
- **Real-world usage example:** When I visit `https://amazon.in`, the browser and server perform a **TLS handshake**:
  1. Browser asks for server certificate.
  2. Server sends its SSL/TLS certificate issued by a trusted Certificate Authority (CA).
  3. Browser verifies it and establishes a shared session key.
  4. Now all data (my shopping cart, payment details) is encrypted with that key.

4. BGP (Border Gateway Protocol)

- BGP is the **routing protocol of the internet**. It's a path-vector protocol that connects different **Autonomous Systems (AS)**, i.e., large networks managed by ISPs, cloud providers, or enterprises. BGP decides how data is routed between countries, ISPs, and organizations.
- **Real-world usage example:** When I connect from India to `www.google.com`, my packets may go through my ISP → Airtel AS → Tata Communications AS → Google AS. These routes are exchanged and optimized using BGP.

6. **Explain TCP vs UDP**

---

1. **TCP (Transmission Control Protocol)**

- **Connection-oriented** → Establishes a connection (3-way handshake) before sending data.
- **Reliable** → Guarantees delivery using acknowledgments, retransmissions, sequencing.
- **Ordered** → Packets arrive in the same order they were sent.
- **Slower but reliable** → Because of error checking and handshakes.
- **Use cases**: Web browsing (HTTP/HTTPS), email (SMTP, IMAP), file transfers (FTP).
  Example: When I load `https://google.com`, TCP ensures all HTML/CSS/JS packets arrive completely and in order. Otherwise, the page would break.

2. **UDP (User Datagram Protocol)**

- **Connectionless** → No handshake, just sends packets (datagrams).
- **Unreliable** → No guarantee of delivery or order.
- **Faster, lightweight** → Less overhead compared to TCP.
- **Use cases**: Real-time applications where speed matters more than reliability.
  Example: When I'm on a Zoom call, UDP is used. If one voice packet is lost, it's okay — retransmitting it would cause delay. Better to skip and keep the conversation flowing.

7. **Explain Switching vs Routing ?**

---

1. **Switching** (Layer 2 - Data Link Layer)

- A **switch** connects devices within the **same network/LAN**.
- Works with **MAC addresses** to forward frames.
- Creates a MAC table → learns which port is connected to which device.
- Doesn't care about IP addresses.
  Example: In my office LAN, a switch connects 20 PCs. If PC1 wants to send data to PC2, the switch looks at the MAC address and forwards directly — fast and efficient.

2. **Routing** (Layer 3 - Network Layer)

- A **router** connects **different networks (LAN to WAN or LAN to LAN)**.
- Works with **IP addresses** to forward packets between networks.

- Uses routing protocols (like OSPF, BGP) to decide best path.
  Example: If I send an email from India to someone in the US, my packet leaves my LAN via a router → goes across ISPs → finally reaches the recipient's LAN. Without routing, my LAN wouldn't talk to other networks.

8. **Tell me about ipv4 and ipv6 ?**

IPv4 and IPv6 are two versions of the Internet Protocol (IP), a set of rules for routing data packets across networks. They both operate at the network layer (Layer 3) of the OSI model. IPv4 is the older version, using 32-bit addresses, while IPv6 is the newer version, using 128-bit addresses to address the exhaustion of IPv4 addresses.

9. **Tell me about MAC addressing and IP addressing ?**

MAC addresses and IP addresses are both identifiers for devices on a network, but they serve different purposes. MAC addresses are unique hardware identifiers used for local network communication, while IP addresses are logical addresses used for routing data across networks, including the internet.

- **MAC Address (Media Access Control Address):**
  - **Purpose:** A MAC address is a unique identifier assigned to a network interface controller (NIC) for communication within a local network (like your home or office network).
  - **Format:** Typically expressed as a 12-digit hexadecimal number (e.g., 00-1B-44-11-3A-B7).
  - **Function:** Used at the data link layer to ensure data packets reach the correct device on the local network.
- **IP Address (Internet Protocol Address):**
  - **Purpose:** An IP address is a logical address used to identify a device on a larger network, like the internet, and to route data packets to the correct network and device.
  - **Format:** Usually represented as a series of numbers separated by periods (e.g., 192.168.1.1 for IPv4).
  - **Function:** Used at the network layer for routing data packets across networks, including the internet.

- **Flexibility:** IP addresses can be dynamically assigned and can change, unlike MAC addresses.

- **Scenario: You turn on hotspot on your mobile 📱 and connect your laptop 💻 to it.**
  - **Step 1: Laptop connects to your phone's Wi-Fi**
    - Your **laptop's Wi-Fi card** has a **MAC address** (say `AA:BB:CC:DD:EE:11`).
    - Your **phone's Wi-Fi card** (hotspot interface) has a **MAC address** (say `77:88:99:AA:BB:22`).
    - Inside the hotspot (like a mini router), the connection between laptop ↔ phone uses **MAC addresses** to identify "which device on Wi-Fi should get the packet."
    - Example: When laptop sends a request → it says:
      - "To phone's MAC address (`77:88:99:AA:BB:22`), from laptop's MAC address (`AA:BB:CC:DD:EE:11`)."
    - That's **local delivery** within the Wi-Fi network.
  - **Step 2: Phone gives your laptop an IP address (via DHCP)**
    - Phone assigns:
      - **Laptop IP (private)** → `192.168.43.101`
      - **Phone IP (gateway)** → `192.168.43.1`
    - So, in the **IP world**, your laptop is now identified by `192.168.43.101` on this small private network.
  - **Step 3: You open google.com from laptop**
    1. Laptop sends the request → **"Hey phone, I want to go to Google's IP (142.250.xx.xx)"**
       - Packet:
         - **MAC header**: Laptop MAC → Phone MAC (local delivery)
         - **IP header**: `192.168.43.101` → `142.250.xx.xx` (Google server)
    2. Phone receives it, strips MAC part (because Wi-Fi part is done), then forwards it through **mobile data**.
    3. On mobile data side, your **phone has its own public IP** (say `103.25.xx.xx`) provided by the telecom operator.
       - Google only sees: **Source IP = 103.25.xx.xx** (your phone's public IP).

4. Google replies back to `103.25.xx.xx` → phone receives it → phone knows which laptop (via MAC & private IP mapping) requested it → sends it back to your laptop.

10. **Difference between Hub, Switch, and Router?**

- A **Hub** is the most basic device. It operates at the **Physical Layer (Layer 1)**. It doesn't understand addresses; it simply repeats incoming signals to all connected devices. This causes collisions and makes the network inefficient.
- A **Switch** works at the **Data Link Layer (Layer 2)**. It understands **MAC addresses**. When a packet comes in, the switch checks the MAC address table and forwards it only to the specific port/device instead of broadcasting to all. This reduces collisions and improves efficiency.
- A **Router** works at the **Network Layer (Layer 3)**. It understands **IP addresses** and is responsible for connecting **different networks**. A router decides the best path for packets to travel across networks and can connect LAN to WAN, apply NAT, and enforce policies like firewall rules.
- **Real Networking Example (not analogy):**
  Let's say we have an **office network**:
  - **Hub case:** If you connect 4 PCs with a hub, and PC1 sends data to PC3, the hub broadcasts it to all 4 PCs. PC2 and PC4 receive unnecessary traffic. This leads to network congestion.
  - **Switch case:** If you replace that hub with a switch, when PC1 sends to PC3, the switch looks up PC3's MAC address in its table and sends it **only to PC3**. PC2 and PC4 won't even know about the transfer.
  - **Router case:** Now, suppose that office LAN (`192.168.1.x`) needs to access the internet. The switch can't help here because it only works inside the LAN. A router with a public IP connects this LAN to the ISP's network, performs **NAT** (translating private IPs to public), and routes traffic to external networks (like google.com).

11. **Explain about Subnetting ?**

Subnetting is the process of dividing a larger network into smaller, more manageable sub-networks (subnets). This is achieved by using a subnet mask, which is a set of bits that identify which part of an IP address represents the

network and which part represents the host. Subnetting helps improve network performance, security, and organization by creating smaller broadcast domains and allowing for more efficient use of IP addresses.

12. **What is Firewall ?**

- A **firewall** is a **network security system** that monitors and controls incoming and outgoing network traffic based on predefined rules.
- Its main purpose is to act as a **barrier** between a trusted internal network and an untrusted external network (like the internet), preventing unauthorized access while allowing legitimate communication.
- For example:
    - Suppose a company wants to block all traffic from outside except HTTP (port 80) and HTTPS (port 443).
    - The firewall will drop packets that don’t match these rules, effectively protecting the internal network.