INTERVIEW PREPARATION GUIDE | 2026 EDITION

100 Computer Networks Interview Questions and Answers

For freshers, intermediate and experienced developers.
OSI, subnetting, TCP, DNS, HTTP, security and real scenarios.

40
Freshers questions

35
Intermediate questions

25
Experienced questions

Code and Debug

codeanddebug.in | YouTube: @codeanddebug | Free to share for learning

How to use this guide

The 100 questions are grouped by level: **Freshers** (1 to 40), **Intermediate** (41 to 75) and **Experienced** (76 to 100). Freshers should master the first section completely and read the second. Experienced candidates are expected to answer all three.

Do not memorise answers word for word. Interviewers follow up with "why?" and "what happens if...?". Practise explaining each answer out loud, and draw diagrams for the TCP handshake, the OSI layers and DNS resolution.

Examples marked PYTHON (TESTED) were run and the output shown is real. Examples marked COMMAND (REAL OUTPUT) were run against our own server on 15 November 2026; your values will differ. Examples marked COMMAND (SYNTAX) show the command only.

The interactive version of this guide has a packet journey through the layers, a TCP handshake and sliding window simulator, a subnetting calculator with practice questions and a DNS lookup walkthrough: codeanddebug.in/blog/computer-networks-interview-questions.

Contents

Freshers (1 to 40)

1. What is a computer network?
2. What are the types of networks based on size (PAN, LAN, MAN, WAN)?
3. What is network topology? Explain the common types.
4. What is the difference between a hub, a switch and a router?
5. What are a bridge, a gateway, a repeater and a modem?
6. What is the OSI model? Explain its seven layers.
7. What is the TCP/IP model?
8. What is the difference between the OSI and TCP/IP models?
9. What are encapsulation and decapsulation?
10. Which devices and protocols work at which OSI layer?
11. What is an IP address?
12. What is the difference between IPv4 and IPv6?
13. What are public and private IP addresses?
14. What are the classes of IPv4 addresses?
15. What is a subnet mask?
16. What is a MAC address?
17. What is the difference between a MAC address and an IP address?
18. What is ARP and how does it work?
19. What is DNS? Why do we need it?
20. What is DHCP? Explain the DORA process.
21. What is the difference between HTTP and HTTPS?
22. What is the difference between TCP and UDP?
23. What is a port number? Name some well-known ports.
24. What is a protocol? Name the common protocols at each layer.
25. What is the difference between bandwidth, throughput and latency?
26. What are unicast, broadcast, multicast and anycast?
27. What are simplex, half-duplex and full-duplex communication?
28. What is the difference between the client-server and peer-to-peer models?
29. What is NAT? Why is it used?
30. What is a default gateway?
31. What is the loopback address?
32. What is ICMP? How does ping work?
33. What is the difference between packet switching and circuit switching?
34. What are the types of transmission media?
35. What is a firewall?
36. What is a VPN and how does it work?
37. What is the difference between a URL, a URI and a URN?
38. What are collision domains and broadcast domains?
39. What is the difference between HTTP GET and POST?
40. Which protocols are used for email?

Intermediate (41 to 75)

41. What is subnetting and CIDR? How do you calculate a subnet?
42. How many subnets and hosts do you get when you subnet? Solve an example.
43. What is supernetting (route aggregation)?
44. Explain the TCP three-way handshake.
45. How is a TCP connection closed? Explain the four-way handshake.
46. How does TCP guarantee reliable delivery?
47. What is TCP flow control? Explain the sliding window.
48. What is TCP congestion control?
49. What is the difference between flow control and congestion control?
50. What are the main fields in the TCP and UDP headers?
51. What is a socket?
52. How does DNS resolution work, step by step?
53. What are the common DNS record types?
54. What is the difference between recursive and iterative DNS queries?
55. What are the main HTTP methods?
56. What do the HTTP status code classes mean? Name the important codes.
57. What does a raw HTTP request and response look like?
58. What is the difference between HTTP/1.1, HTTP/2 and HTTP/3?
59. What is the difference between cookies, sessions and tokens?
60. How does traceroute work?
61. What is the difference between static and dynamic routing?
62. What is the difference between distance vector and link state routing?
63. What is BGP?
64. What is the difference between CSMA/CD and CSMA/CA?
65. What is a VLAN?
66. How does a switch learn MAC addresses?
67. How does the TLS handshake work?
68. What is the difference between symmetric and asymmetric encryption?
69. What is the difference between a forward proxy and a reverse proxy?
70. How are transmission errors detected and corrected?
71. What are Stop-and-Wait, Go-Back-N and Selective Repeat?
72. What are MTU and fragmentation?
73. What are FTP, SFTP and SSH?
74. How would you check a device's IP configuration, routes and open connections?
75. What is port forwarding?

Experienced (76 to 100)

76. What happens when you type a URL into the browser and press Enter?
77. What is a load balancer? What is the difference between layer 4 and layer 7 load balancing?
78. What is a CDN and how does it work?
79. What is QUIC and why does HTTP/3 use it?
80. What is head-of-line blocking?
81. What is the TIME_WAIT state? Why can too many connections in TIME_WAIT cause problems?
82. What are Nagle's algorithm and delayed ACK? Why do they cause latency together?
83. What is the bandwidth-delay product? Why does it matter?
84. What is the difference between polling, long polling, Server-Sent Events and WebSockets?
85. What is a DDoS attack? How is it mitigated?
86. What is a SYN flood? How do SYN cookies prevent it?
87. What is a man-in-the-middle attack? How do HTTPS, certificates and HSTS prevent it?
88. How does the world move from IPv4 to IPv6?
89. What are DNS TTL and DNS propagation? How do you change DNS safely?
90. What is anycast and where is it used?
91. How do peer-to-peer apps connect through NAT? What are STUN and TURN?
92. A website is slow for users. How would you find out whether the network is the problem?
93. What is a VPC? How are public and private subnets designed in the cloud?
94. What are HTTP keep-alive and connection pooling? Why do they matter?
95. What is zero trust networking?
96. What is SDN (Software-Defined Networking)?
97. What is CORS and why do browsers enforce it?
98. What is rate limiting? Name common algorithms.
99. Design the network for a company's new office of 120 people.
100. Users can reach websites by IP address but not by name. How do you troubleshoot it?

Freshers level

Network types and devices, the OSI and TCP/IP models, IP and MAC addresses, ARP, DNS, DHCP, HTTP and HTTPS, TCP and UDP, ports and NAT.

Q1. What is a computer network?

FRESHERS · BASICS

A computer network is a group of devices (computers, phones, servers, printers) connected so they can share data and resources. The connection can be wired, such as Ethernet cables or optical fibre, or wireless, such as Wi-Fi or mobile data.

Every network needs three things: **devices** that send and receive data, a **medium** that carries the signal, and **protocols**, the agreed rules that tell devices how to format, send and understand the data. The internet is simply a very large network of networks that all follow the same core protocols, TCP/IP.

Q2. What are the types of networks based on size (PAN, LAN, MAN, WAN)?

FRESHERS · BASICS

Type	Full form	Covers	Example
PAN	Personal Area Network	A few metres around one person	Phone connected to earbuds over Bluetooth
LAN	Local Area Network	A home, office or building	Computers in a college lab on one switch
MAN	Metropolitan Area Network	A city	A cable or fibre network across a city
WAN	Wide Area Network	Countries and continents	The internet; a company linking offices in Surat and Bengaluru

As the area grows, speed usually drops, delay rises, and the network is more often owned by a service provider instead of you.

Q3. What is network topology? Explain the common types.

FRESHERS · BASICS

Topology is the layout of how devices are connected.

- **Bus:** every device is on one shared cable. Cheap, but one cable fault takes the whole network down.
- **Star:** every device connects to a central switch or hub. Easy to manage and the most common layout in LANs, but the central device is a single point of failure.
- **Ring:** each device connects to two neighbours in a loop, and data travels around it. A single break can stop the ring unless it is a dual ring.
- **Mesh:** devices have many links to each other. A full mesh is very reliable but expensive. The internet backbone is a partial mesh.
- **Tree:** star networks connected in a hierarchy, common in large buildings.
- **Hybrid:** a mix of two or more of the above.

Q4. What is the difference between a hub, a switch and a router?

FRESHERS · BASICS

Point	Hub	Switch	Router
OSI layer	Layer 1 (Physical)	Layer 2 (Data Link)	Layer 3 (Network)
Uses	Nothing; just repeats signals	MAC addresses	IP addresses
Sends data to	Every port	Only the port of the destination device	The best next hop towards another network
Connects	Devices in one LAN	Devices in one LAN	Different networks, such as your LAN and the internet
Collision domain	One for all ports	One per port	One per port

Hubs are now obsolete. Your home Wi-Fi box is actually a router, a switch and a wireless access point combined in one device.

Q5. What are a bridge, a gateway, a repeater and a modem?

FRESHERS · BASICS

- **Repeater:** regenerates a weak signal so it can travel further. Layer 1.
- **Bridge:** connects two LAN segments and filters traffic by MAC address. A switch is essentially a bridge with many ports. Layer 2.
- **Gateway:** connects two networks that use different protocols and translates between them. In everyday use, the "default gateway" is simply the router your device sends traffic to when the destination is outside its own network.
- **Modem (modulator-demodulator):** converts digital data into a signal that can travel over a phone line, cable or fibre, and back again. It connects your home to your internet service provider.

Q6. What is the OSI model? Explain its seven layers.

FRESHERS · OSI AND TCP/IP

The OSI (Open Systems Interconnection) model is a reference model that splits network communication into seven layers. Each layer has one job and only talks to the layers directly above and below it. The tool at the top of this page lets you send a message down these layers and watch each header being added.

#	Layer	Job	Data unit	Examples
7	Application	Network services for applications	Data	HTTP, DNS, SMTP, FTP
6	Presentation	Format, encryption, compression	Data	TLS, JPEG, UTF-8
5	Session	Open, manage and close sessions	Data	NetBIOS, RPC
4	Transport	End-to-end delivery between processes, ports	Segment (TCP), datagram (UDP)	TCP, UDP
3	Network	Logical addressing and routing between networks	Packet	IP, ICMP, routers

#	Layer	Job	Data unit	Examples
2	Data Link	Delivery between neighbours on one link, MAC addresses	Frame	Ethernet, Wi-Fi, switches
1	Physical	Bits as electrical, light or radio signals	Bits	Cables, hubs, repeaters

A common way to remember them from layer 7 down: **All People Seem To Need Data Processing**.

Q7. What is the TCP/IP model?

FRESHERS · OSI AND TCP/IP

The TCP/IP model is the model the internet actually runs on. It has four layers (some books split the bottom layer and show five):

- **Application:** combines OSI layers 5, 6 and 7. HTTP, DNS, SMTP, SSH.
- **Transport:** TCP and UDP, the same as OSI layer 4.
- **Internet:** IP, ICMP and routing, the same as OSI layer 3.
- **Network access (link):** combines OSI layers 1 and 2. Ethernet, Wi-Fi.

It came from real protocols developed for ARPANET, so it describes what is actually used. OSI came later as a general reference and is mainly used for teaching and troubleshooting ("is this a layer 2 or layer 3 problem?").

Q8. What is the difference between the OSI and TCP/IP models?

FRESHERS · OSI AND TCP/IP

Point	OSI model	TCP/IP model
Layers	7	4 (or 5)
Nature	Theoretical reference model	Practical model the internet uses
Developed	Model first, protocols later	Protocols first, model described from them
Session and presentation	Separate layers	Merged into the application layer
Physical and data link	Separate layers	Merged into one network access layer
Used for	Teaching and troubleshooting vocabulary	Real implementation

Q9. What are encapsulation and decapsulation?

FRESHERS · OSI AND TCP/IP

Encapsulation happens on the sender. As data moves down the layers, each layer wraps what it receives with its own header (and the data link layer also adds a trailer):

- The application data gets a TCP header and becomes a **segment**.
- The segment gets an IP header and becomes a **packet**.
- The packet gets an Ethernet header and trailer and becomes a **frame**.
- The frame is sent as **bits**.

Decapsulation is the reverse on the receiver: each layer reads and removes its own header and passes the rest up. A router in the middle only decapsulates up to layer 3, reads the destination IP, then builds a new frame for the next link. The packet journey tool on this page shows each

step.

Q10. Which devices and protocols work at which OSI layer?

FRESHERS · OSI AND TCP/IP

Layer	Devices	Protocols and standards
7 Application	Proxies, layer 7 load balancers, firewalls with deep inspection	HTTP, HTTPS, DNS, DHCP, SMTP, FTP, SSH
6 Presentation	None specific	TLS/SSL (often placed here), encoding and compression
5 Session	None specific	RPC, NetBIOS
4 Transport	Layer 4 load balancers, stateful firewalls	TCP, UDP, QUIC
3 Network	Routers, layer 3 switches	IPv4, IPv6, ICMP, OSPF, BGP
2 Data Link	Switches, bridges, network cards, access points	Ethernet, Wi-Fi (802.11), ARP, PPP
1 Physical	Hubs, repeaters, cables, modems	Signalling standards, connectors

Some protocols sit between layers. ARP, for example, links layer 3 addresses to layer 2 addresses and is often called a layer 2.5 protocol.

Q11. What is an IP address?

FRESHERS · IP ADDRESSING

An IP (Internet Protocol) address is a logical address that identifies a device's network interface on an IP network, so packets can be routed to it. It has two parts: a **network part**, which says which network the device is on, and a **host part**, which says which device on that network.

- **IPv4:** 32 bits, written as four decimal numbers from 0 to 255, such as `192.168.1.10`.
- **IPv6:** 128 bits, written as eight groups of hexadecimal, such as `2001:db8::1`.

An IP address can change when a device moves to another network. That is the key difference from a MAC address, which is tied to the hardware.

Q12. What is the difference between IPv4 and IPv6?

FRESHERS · IP ADDRESSING

Point	IPv4	IPv6
Address size	32 bits	128 bits
Number of addresses	About 4.3 billion	About 3.4×10^{38}
Format	Dotted decimal: 192.168.1.1	Hexadecimal groups: 2001:db8::1
Header	Variable, 20 to 60 bytes, with a checksum	Fixed 40 bytes, no checksum, simpler for routers
Address setup	Manual or DHCP	Can configure itself (SLAAC), or DHCPv6
Broadcast	Yes	No; uses multicast instead
NAT	Widely needed because addresses ran out	Not needed
Fragmentation	By routers or the sender	Only by the sender

Python's `ipaddress` module understands both versions. Note how IPv6 lets you shorten runs of zeros with `::`.

PYTHON (TESTED)

```
import ipaddress

v4 = ipaddress.ip_address("192.168.1.10")
v6 = ipaddress.ip_address("2001:0db8:0000:0000:0000:0000:0000:0001")

print(v4.version, v4.max_prefixlen, "bits")
print(v6.version, v6.max_prefixlen, "bits")
print("Short form:", v6.compressed)
print("Full form :", v6.exploded)
print("Total IPv4 addresses:", 2 ** 32)
```

OUTPUT

```
4 32 bits
6 128 bits
Short form: 2001:db8::1
Full form : 2001:0db8:0000:0000:0000:0000:0000:0001
Total IPv4 addresses: 4294967296
```

Q13. What are public and private IP addresses?

FRESHERS · IP ADDRESSING

A **public IP address** is unique on the whole internet and can be reached from anywhere. Your internet service provider assigns it to your router.

A **private IP address** is used only inside a local network and is not routed on the internet. The same private addresses are reused in millions of homes and offices. Three ranges are reserved for this (RFC 1918):

- `10.0.0.0` to `10.255.255.255` (10.0.0.0/8)
- `172.16.0.0` to `172.31.255.255` (172.16.0.0/12)
- `192.168.0.0` to `192.168.255.255` (192.168.0.0/16)

Devices with private addresses reach the internet through NAT on the router, which swaps the private address for the router's public one.

PYTHON (TESTED)

```
import ipaddress

for ip in ["192.168.1.10", "10.20.30.40", "172.20.5.1", "172.32.0.1", "64.227.174.41"]:
    kind = "private" if ipaddress.ip_address(ip).is_private else "public"
    print(f"{ip:15} -> {kind}")
```

OUTPUT

```
192.168.1.10    -> private
10.20.30.40     -> private
172.20.5.1      -> private
172.32.0.1      -> public
64.227.174.41   -> public
```

Q14. What are the classes of IPv4 addresses?

FRESHERS · IP ADDRESSING

Before CIDR, IPv4 addresses were divided into fixed classes based on the first bits of the address:

Class	First octet	Default mask	Networks and hosts	Use
A	1 to 126	255.0.0.0 (/8)	126 networks, about 16.7 million hosts each	Very large organisations

Class	First octet	Default mask	Networks and hosts	Use
B	128 to 191	255.255.0.0 (/16)	About 16,000 networks, 65,534 hosts each	Medium organisations
C	192 to 223	255.255.255.0 (/24)	About 2 million networks, 254 hosts each	Small networks
D	224 to 239	None	Not split into hosts	Multicast
E	240 to 255	None	Not split into hosts	Reserved, experimental

127 is missing from class A because `127.0.0.0/8` is reserved for loopback. Classful addressing wasted huge numbers of addresses, so it was replaced by **CIDR** in 1993. Classes are still asked in interviews, but real networks use CIDR prefixes.

Q15. What is a subnet mask?

FRESHERS · IP ADDRESSING

A subnet mask tells a device which part of an IP address is the network part and which part is the host part. In binary, the mask is a run of 1s (network bits) followed by 0s (host bits).

For `192.168.1.10` with mask `255.255.255.0`, the first 24 bits are the network, so the network is `192.168.1.0` and the host is `.10`. The same mask is written `/24` in CIDR notation.

A device uses the mask to decide where to send a packet. If the destination is in the same network, it delivers the packet directly. If not, it sends the packet to its default gateway. The subnetting calculator on this page shows the mask in binary for any prefix.

Q16. What is a MAC address?

FRESHERS · IP ADDRESSING

A MAC (Media Access Control) address is the physical address of a network interface card, used to deliver frames within one local network (layer 2). It is 48 bits long, written as six pairs of hexadecimal digits such as `3C:22:FB:9A:10:4E`.

- The first 24 bits are the **OUI** (Organisationally Unique Identifier), assigned to the manufacturer.
- The last 24 bits are assigned by the manufacturer to that card.
- `FF:FF:FF:FF:FF:FF` is the broadcast MAC address.

MAC addresses are burned into the hardware, but they can be changed in software. Phones now use a random MAC for each Wi-Fi network to protect privacy.

Q17. What is the difference between a MAC address and an IP address?

FRESHERS · IP ADDRESSING

Point	MAC address	IP address
Layer	Data Link (2)	Network (3)
Size	48 bits	32 bits (IPv4), 128 bits (IPv6)
Assigned by	Manufacturer	Network administrator or DHCP
Changes when you move networks	No	Yes
Scope	One local link	End to end across networks
Used by	Switches	Routers

An analogy: the IP address is like a postal address, which says where the parcel finally goes. The MAC address is like the name of the next courier hub, which changes at every hop. As a packet crosses routers, the IP addresses stay the same, but the MAC addresses in the frame are rewritten at each hop.

Q18. What is ARP and how does it work?

FRESHERS · PROTOCOLS

ARP (Address Resolution Protocol) finds the MAC address that belongs to an IP address on the local network. A device needs this because frames on a LAN are delivered by MAC address, while applications only know IP addresses.

1. The device checks its **ARP cache**. If the mapping is there, it uses it.
2. If not, it broadcasts an **ARP request** to everyone on the LAN: "Who has 192.168.1.1? Tell 192.168.1.10."
3. The device with that IP replies directly (unicast) with an **ARP reply** containing its MAC address.
4. The sender stores the answer in its cache for a few minutes and sends the frame.

If the destination is on another network, the device asks ARP for the MAC address of its **default gateway**, not the final destination. In IPv6, ARP is replaced by Neighbor Discovery (NDP).

Q19. What is DNS? Why do we need it?

FRESHERS · PROTOCOLS

DNS (Domain Name System) is the internet's phone book. It translates human-friendly names such as `codeanddebug.in` into IP addresses that computers use to connect.

Without DNS you would need to remember an IP address for every site, and a site could never move to a new server without telling everyone its new address. DNS is a **distributed, hierarchical** database: no single server knows everything; root servers point to TLD servers, which point to each domain's authoritative servers. It mostly runs over UDP port 53, and switches to TCP for large responses.

Here is a real lookup of our own domain. The DNS lookup tool on this page walks through every server involved.

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
nslookup codeanddebug.in
```

OUTPUT

```
Server:         172.16.10.1
Address:        172.16.10.1#53

Non-authoritative answer:
Name:   codeanddebug.in
Address: 64.227.174.41
```

Q20. What is DHCP? Explain the DORA process.

FRESHERS · PROTOCOLS

DHCP (Dynamic Host Configuration Protocol) automatically gives a device an IP address, subnet mask, default gateway and DNS servers when it joins a network. Without it, every device would need manual setup. It uses UDP ports 67 (server) and 68 (client).

The process has four messages, remembered as **DORA**:

1. **Discover:** the new device broadcasts "Is there a DHCP server?" It has no IP yet, so it uses source 0.0.0.0 and destination 255.255.255.255.
2. **Offer:** a DHCP server offers an available IP address and settings.
3. **Request:** the device broadcasts that it accepts that offer, so any other servers know their offers were declined.
4. **Acknowledge:** the server confirms, and the address is **leased** for a set time. The device renews the lease before it expires.

Q21. What is the difference between HTTP and HTTPS?

FRESHERS · PROTOCOLS

Point	HTTP	HTTPS
Full form	HyperText Transfer Protocol	HTTP Secure (HTTP over TLS)
Default port	80	443
Encryption	None; anyone on the path can read it	Encrypted with TLS
Server identity	Not verified	Verified with a certificate
Data integrity	Can be changed in transit	Tampering is detected

HTTPS protects passwords, payments and personal data, and browsers mark plain HTTP pages as "Not secure". Google also uses HTTPS as a ranking signal. Here is how our own server sends HTTP visitors to HTTPS with a permanent redirect:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
curl -I http://codeanddebug.in/blog/
```

OUTPUT

```
HTTP/1.1 301 Moved Permanently
Server: nginx
Date: Sun, 15 Nov 2026 07:22:31 GMT
Content-Type: text/html
Content-Length: 162
Connection: keep-alive
Location: https://codeanddebug.in/blog/
```

Q22. What is the difference between TCP and UDP?

FRESHERS · TRANSPORT LAYER

Point	TCP	UDP
Connection	Connection-oriented (3-way handshake first)	Connectionless
Reliability	Guaranteed delivery with ACKs and retransmission	No guarantee; lost packets stay lost
Order	Data arrives in order	May arrive out of order
Flow and congestion control	Yes	No
Header size	20 to 60 bytes	8 bytes
Speed	Slower, more overhead	Faster, very little overhead
Data model	Byte stream	Separate messages (datagrams)
Used by	Web, email, file transfer, SSH	DNS, video calls, online games, live streaming, DHCP

Choose TCP when every byte must arrive correctly. Choose UDP when speed matters more than perfection, for example a video call, where a late packet is useless anyway. The example below sends the same message over both, on one machine.

PYTHON (TESTED)

```
import socket, threading

def tcp_server(sock):
    conn, _ = sock.accept()                 # wait for a connection
    data = conn.recv(1024)
    conn.sendall(b"TCP echo: " + data)
    conn.close()

srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # TCP
srv.bind(("127.0.0.1", 0)); srv.listen(1)
threading.Thread(target=tcp_server, args=(srv,)).start()

cli = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
cli.connect(srv.getsockname())              # 3-way handshake happens here
cli.sendall(b"hello")
print(cli.recv(1024).decode())
cli.close()

usrv = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)   # UDP
usrv.bind(("127.0.0.1", 0))
ucli = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
ucli.sendto(b"hello", usrv.getsockname())   # no connection, just send
data, addr = usrv.recvfrom(1024)
print("UDP got:", data.decode())
```

OUTPUT

```
TCP echo: hello
UDP got: hello
```

Q23. What is a port number? Name some well-known ports.

FRESHERS · TRANSPORT LAYER

An IP address takes data to the right device. A **port number** takes it to the right application on that device. It is a 16-bit number, from 0 to 65535. The pair (IP address, port) is called a **socket address**.

- **Well-known ports:** 0 to 1023, for standard services.
- **Registered ports:** 1024 to 49151, for specific applications such as MySQL.
- **Dynamic or ephemeral ports:** 49152 to 65535, picked by the operating system for the client side of a connection.

Port	Protocol	Port	Protocol
20, 21	FTP	80	HTTP
22	SSH	110	POP3
23	Telnet	143	IMAP
25	SMTP	443	HTTPS
53	DNS	3306	MySQL
67, 68	DHCP	5432	PostgreSQL

Q24. What is a protocol? Name the common protocols at each layer.

FRESHERS · BASICS

A protocol is a set of rules that defines how data is formatted, sent, received and acknowledged, so that devices from different makers can understand each other. It specifies the message format, the order of messages, timing, and what to do when something goes wrong.

- **Application:** HTTP and HTTPS (web), DNS (names), SMTP, POP3 and IMAP (email), FTP and SFTP (files), SSH (remote login), DHCP (address setup).
- **Transport:** TCP, UDP, QUIC.
- **Network:** IP, ICMP (errors and ping), routing protocols such as OSPF and BGP.
- **Data link:** Ethernet, Wi-Fi (IEEE 802.11), ARP, PPP.

Q25. What is the difference between bandwidth, throughput and latency?

FRESHERS · BASICS

- **Bandwidth** is the maximum amount of data a link can carry per second, such as a 100 Mbps plan. It is the width of the pipe.
- **Throughput** is how much data actually gets through per second. It is always at or below bandwidth, because of congestion, packet loss and protocol overhead.
- **Latency** is the time for data to travel from source to destination, usually measured in milliseconds. Round-trip time (RTT) is the time to go there and come back.
- **Jitter** is the variation in latency, which hurts calls and games.

A link can have high bandwidth and still feel slow if latency is high. Satellite internet is the classic example. For loading a web page, which needs many small round trips, latency often matters more than bandwidth.

Q26. What are unicast, broadcast, multicast and anycast?

FRESHERS · BASICS

- **Unicast:** one sender to one receiver. Normal web browsing.
- **Broadcast:** one sender to every device on the local network, such as an ARP request or DHCP Discover. Routers do not forward broadcasts, so a broadcast stays inside its network.
- **Multicast:** one sender to a group of receivers who have joined that group, such as IPTV or stock price feeds. IPv4 multicast uses 224.0.0.0 to 239.255.255.255.
- **Anycast:** one address is announced from several locations, and each sender reaches the nearest one. DNS root servers and CDNs use anycast.

Q27. What are simplex, half-duplex and full-duplex communication?

FRESHERS · BASICS

- **Simplex:** data flows in one direction only. Example: a keyboard to a computer, or a TV broadcast.
- **Half-duplex:** data flows in both directions, but only one at a time. Example: a walkie-talkie, or an old Ethernet hub.
- **Full-duplex:** data flows in both directions at the same time. Example: a phone call, or a modern switched Ethernet link.

Full-duplex Ethernet removed collisions completely, which is one reason CSMA/CD no longer matters on modern wired networks.

Q28. What is the difference between the client-server and peer-to-peer models?

FRESHERS · BASICS

Point	Client-server	Peer-to-peer (P2P)
Roles	Clients request, servers provide	Every peer can do both
Control	Central	Distributed
Scaling	Server must grow with users	More peers add more capacity
Single point of failure	The server	None
Security and management	Easier	Harder
Examples	Websites, email, banking apps	BitTorrent, some blockchain networks

Q29. What is NAT? Why is it used?

FRESHERS · NETWORK LAYER

NAT (Network Address Translation) lets many devices with private IP addresses share one public IP address. The router rewrites the source address of outgoing packets to its own public address and keeps a **translation table** so it can send replies back to the right device.

- **Static NAT:** one private address maps to one public address permanently.
- **Dynamic NAT:** private addresses map to a pool of public addresses.
- **PAT (Port Address Translation, also called NAT overload):** many devices share one public IP, told apart by port number. This is what your home router does.

Main reasons: IPv4 addresses ran out, and NAT hides internal addresses. The downside is that devices behind NAT cannot easily accept incoming connections, which makes peer-to-peer calls and gaming harder.

Q30. What is a default gateway?

FRESHERS · NETWORK LAYER

The default gateway is the router a device sends packets to when the destination IP is **not** in its own network. It is usually the router's address on the LAN, such as `192.168.1.1`.

When a laptop at `192.168.1.10/24` wants to reach `64.227.174.41`, it sees that the destination is outside `192.168.1.0/24`, so it uses ARP to find the gateway's MAC address and sends the frame there. The gateway then routes the packet onwards. A device with a wrong or missing default gateway can talk to its own LAN but not the internet, which is a classic troubleshooting question.

Q31. What is the loopback address?

FRESHERS · IP ADDRESSING

The loopback address is a special address that always points back to the same machine. Packets sent to it never leave the device or reach the network card.

- IPv4 reserves the whole `127.0.0.0/8` block. `127.0.0.1` is the one everyone uses, and the name `localhost` points to it.
- IPv6 uses `::1`.

Developers use it to run and test servers locally, for example a Django app on `http://127.0.0.1:8000`. Pinging 127.0.0.1 is also a quick check that the TCP/IP stack on your machine is working.

Q32. What is ICMP? How does ping work?

FRESHERS · NETWORK LAYER

ICMP (Internet Control Message Protocol) is used by network devices to send error and diagnostic messages, such as "destination unreachable" or "time exceeded". It works alongside IP at the network layer and has no ports.

`ping` sends ICMP **Echo Request** messages and waits for **Echo Reply** messages. It tells you whether a host is reachable, the round-trip time for each packet, and how many were lost.

Here is a real ping to our own server. The `time` column is the round-trip time. The `ttl` value is how many more hops the reply could have made; its starting value depends on the operating system that sent it, and it can be changed by devices along the path. A host that does not reply is not always down: many servers and firewalls block ICMP.

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
ping -c 4 codeanddebug.in
```

OUTPUT

```
PING codeanddebug.in (64.227.174.41) 56(84) bytes of data.
64 bytes from 64.227.174.41 (64.227.174.41): icmp_seq=1 ttl=64 time=76.4 ms
64 bytes from 64.227.174.41 (64.227.174.41): icmp_seq=2 ttl=64 time=59.2 ms
64 bytes from 64.227.174.41 (64.227.174.41): icmp_seq=3 ttl=64 time=73.6 ms
64 bytes from 64.227.174.41 (64.227.174.41): icmp_seq=4 ttl=64 time=71.7 ms

--- codeanddebug.in ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 59.191/70.229/76.397/6.587 ms
```

Q33. What is the difference between packet switching and circuit switching?

FRESHERS · BASICS

Point	Circuit switching	Packet switching
Path	A dedicated path is reserved before talking	Data is split into packets that can take different paths
Resource use	Reserved even when nobody speaks	Shared; used only when sending
Delay	Fixed and low once connected	Varies with congestion
Setup time	Needed	None
Failure	A broken link drops the call	Packets are routed around the failure
Example	Traditional landline telephone	The internet

The internet chose packet switching because computer traffic comes in bursts, so sharing links is far more efficient than reserving them.

Q34. What are the types of transmission media?

FRESHERS · BASICS

Guided (wired) media:

- **Twisted pair:** pairs of copper wires twisted to cut interference. Cat5e and Cat6 Ethernet cables. Cheap; up to 100 metres per run.
- **Coaxial cable:** a copper core with shielding. Cable TV and older broadband.
- **Optical fibre:** carries data as light. Very high bandwidth, long distances and no electrical interference. Used in backbones, undersea cables and fibre-to-the-home.

Unguided (wireless) media:

- **Radio waves:** Wi-Fi, mobile networks, Bluetooth.
- **Microwaves:** point-to-point towers and satellite links.
- **Infrared:** very short range, such as TV remotes.

Q35. What is a firewall?

FRESHERS · SECURITY

A firewall monitors network traffic and allows or blocks it based on rules, forming a barrier between a trusted network and an untrusted one such as the internet.

- **Packet-filtering firewall:** checks each packet's IP addresses, ports and protocol. Fast, but has no memory of connections.
- **Stateful firewall:** tracks connections, so it allows replies to traffic that was started from inside and blocks unsolicited incoming traffic.
- **Application-level (proxy) firewall and next-generation firewall:** inspect content at layer 7, for example blocking certain websites or detecting malware.
- **Web application firewall (WAF):** protects web apps from attacks such as SQL injection and cross-site scripting.

Firewalls can be hardware appliances, software on a host (Windows Defender Firewall, ufw on Linux), or cloud rules such as AWS security groups.

Q36. What is a VPN and how does it work?

FRESHERS · SECURITY

A VPN (Virtual Private Network) creates an encrypted **tunnel** between your device and a VPN server. Your traffic is wrapped inside encrypted packets, so people on the local network (for example public Wi-Fi) and your internet provider cannot read it. Websites see the VPN server's IP address instead of yours.

- **Remote-access VPN:** an employee working from home connects to the office network.
- **Site-to-site VPN:** two office networks are joined over the internet.

Common protocols are IPsec, OpenVPN and WireGuard. A VPN protects data in transit, but it does not make you anonymous: the VPN provider can still see your traffic.

Q37. What is the difference between a URL, a URI and a URN?

FRESHERS · APPLICATION LAYER

- **URI (Uniform Resource Identifier):** any string that identifies a resource. It is the general term.
- **URL (Uniform Resource Locator):** a URI that also says **where** the resource is and **how** to get it, such as `https://codeanddebug.in/blog/`.
- **URN (Uniform Resource Name):** a URI that names a resource without saying where it is, such as `urn:isbn:9780131103627` for a book.

So every URL is a URI, but not every URI is a URL. The parts of a URL:

PYTHON (TESTED)

```
from urllib.parse import urlparse

u = urlparse("https://codeanddebug.in:443/blog/search?q=dns#results")
print("scheme  :", u.scheme)
print("host    :", u.hostname)
print("port    :", u.port)
print("path    :", u.path)
print("query   :", u.query)
print("fragment:", u.fragment)
```

OUTPUT

```
scheme  : https
host    : codeanddebug.in
port    : 443
path    : /blog/search
query   : q=dns
fragment: results
```

Q38. What are collision domains and broadcast domains?

FRESHERS · DATA LINK LAYER

A **collision domain** is a part of the network where two devices sending at the same time would collide. A **broadcast domain** is the set of devices that receive each other's broadcast messages.

Device	Collision domains	Broadcast domains
Hub	One for all ports	One
Switch	One per port	One (unless VLANs are used)
Router	One per port	One per port

A typical interview puzzle: 3 hubs connected to a switch, which is connected to a router with 2 LAN ports in use. Each switch port is its own collision domain, and each router interface is its own

broadcast domain. Switches split collision domains; routers split broadcast domains.

Q39. What is the difference between HTTP GET and POST?

FRESHERS · APPLICATION LAYER

Point	GET	POST
Purpose	Read data	Send data to create or process something
Data location	In the URL query string	In the request body
Visible in URL and history	Yes	No
Size limit	Limited by URL length (often about 2,000 to 8,000 characters)	No practical limit
Cached and bookmarked	Yes	Normally no
Idempotent	Yes: repeating it changes nothing	No: repeating may create duplicates
Safe for sensitive data	No	Better, but still needs HTTPS

POST is not encrypted by itself. Only HTTPS keeps the body private.

Q40. Which protocols are used for email?

FRESHERS · APPLICATION LAYER

- **SMTP (Simple Mail Transfer Protocol):** sends email, from your mail app to your mail server and between mail servers. Port 25 between servers; 587 (or 465) for clients submitting mail.
- **POP3 (Post Office Protocol 3):** downloads email to one device and usually deletes it from the server. Port 110, or 995 with TLS.
- **IMAP (Internet Message Access Protocol):** keeps email on the server and syncs folders and read status across all your devices. Port 143, or 993 with TLS. Almost everyone uses IMAP today.

DNS is also involved: a domain's **MX records** tell other servers where to deliver its email. Ours point to Google Workspace:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
dig +short codeanddebug.in MX
```

OUTPUT

```
1 aspmx.l.google.com.
5 alt2.aspmx.l.google.com.
5 alt1.aspmx.l.google.com.
10 alt3.aspmx.l.google.com.
10 alt4.aspmx.l.google.com.
```

Intermediate level

Subnetting and CIDR, the TCP handshake and teardown, flow and congestion control, DNS in depth, HTTP versions, routing protocols, VLANs, TLS and error control.

Q41. What is subnetting and CIDR? How do you calculate a subnet?

INTERMEDIATE · SUBNETTING

Subnetting splits one network into smaller networks (subnets) by borrowing bits from the host part. It reduces broadcast traffic, improves security by separating groups, and avoids wasting addresses.

CIDR (Classless Inter-Domain Routing) writes the network size as a prefix length after a slash. `/26` means the first 26 bits are the network and the remaining 6 bits are for hosts.

For `192.168.10.77/26`:

1. Host bits = 32 − 26 = 6, so each subnet has 2^6 = 64 addresses and 64 − 2 = **62 usable hosts** (the first is the network address and the last is the broadcast address).
2. The mask is 255.255.255.192, so subnets step by 256 − 192 = 64 in the last octet: .0, .64, .128, .192.
3. 77 falls in the .64 block, so the network is **192.168.10.64**, the broadcast is **192.168.10.127** and the usable range is **.65 to .126**.

Python's `ipaddress` module checks the working. The subnetting calculator on this page does the same for any address and gives you practice questions.

PYTHON (TESTED)

```
import ipaddress

net = ipaddress.ip_interface("192.168.10.77/26").network
hosts = list(net.hosts())
print("Network  :", net.network_address)
print("Broadcast:", net.broadcast_address)
print("Mask     :", net.netmask)
print("Usable   :", hosts[0], "to", hosts[-1], f"({len(hosts)} hosts)")

print("Split 192.168.10.0/24 into /26 subnets:")
for s in ipaddress.ip_network("192.168.10.0/24").subnets(new_prefix=26):
    print("  ", s)
```

OUTPUT

```
Network  : 192.168.10.64
Broadcast: 192.168.10.127
Mask     : 255.255.255.192
Usable   : 192.168.10.65 to 192.168.10.126 (62 hosts)
Split 192.168.10.0/24 into /26 subnets:
   192.168.10.0/26
   192.168.10.64/26
   192.168.10.128/26
   192.168.10.192/26
```

Q42. How many subnets and hosts do you get when you subnet? Solve an example.

INTERMEDIATE · SUBNETTING

Two formulas cover almost every exam question:

- **Number of subnets = 2^n**, where n is the number of bits borrowed from the host part.
- **Usable hosts per subnet = $2^h - 2$**, where h is the number of host bits left. Subtract 2 for the network and broadcast addresses.

Example: a company has `172.16.0.0/16` and needs at least 50 subnets.

1. $2^5 = 32$ is too few, $2^6 = 64$ is enough, so borrow 6 bits.
2. The new prefix is /16 + 6 = **/22**, mask 255.255.252.0.
3. Host bits left = 32 − 22 = 10, so each subnet has $2^{10} - 2$ = **1,022 usable hosts**.

The exception is `/31`, which is allowed for point-to-point links with exactly 2 usable addresses, and `/32`, which is a single host.

Q43. What is supernetting (route aggregation)?

INTERMEDIATE · SUBNETTING

Supernetting is the opposite of subnetting: several smaller, contiguous networks are combined into one larger prefix. Routers then advertise one route instead of many, which keeps routing tables small.

Example: `192.168.0.0/24`, `192.168.1.0/24`, `192.168.2.0/24` and `192.168.3.0/24` share their first 22 bits, so they can be summarised as `192.168.0.0/22`.

The networks must be contiguous and start on the right boundary. You cannot summarise 192.168.1.0/24 to 192.168.4.0/24 into a single /22.

PYTHON (TESTED)

```
import ipaddress

nets = [ipaddress.ip_network(f"192.168.{i}.0/24") for i in range(4)]
print(list(ipaddress.collapse_addresses(nets)))
```

OUTPUT

```
[IPv4Network('192.168.0.0/22')]
```

Q44. Explain the TCP three-way handshake.

INTERMEDIATE · TRANSPORT LAYER

Before sending data, TCP sets up a connection so both sides agree on starting **sequence numbers** and know the other side is ready.

1. **SYN:** the client sends a segment with the SYN flag and a random initial sequence number, say x.
2. **SYN-ACK:** the server replies with SYN and ACK flags, its own sequence number y, and acknowledgement number x + 1.
3. **ACK:** the client replies with acknowledgement number y + 1. The connection is now **ESTABLISHED**.

Three messages are needed because each side must send its own sequence number and have it acknowledged. Two would confirm only one direction, and an old, delayed SYN could open a connection by mistake. Random starting numbers make it hard for an attacker to inject fake segments. Step through it in the TCP tool on this page.

Q45. How is a TCP connection closed? Explain the four-way handshake.

INTERMEDIATE · TRANSPORT LAYER

TCP is full-duplex, so each direction is closed separately:

1. The side that is finished sends **FIN**.
2. The other side replies with **ACK**. It can keep sending its own remaining data; the connection is now half-closed.
3. When it is done, it sends its own **FIN**.
4. The first side replies with **ACK** and enters **TIME_WAIT**.

TIME_WAIT lasts twice the maximum segment lifetime (often 60 seconds in total on Linux). It makes sure the final ACK can be resent if it was lost, and that delayed packets from the old connection cannot be mistaken for a new one using the same ports. Steps 2 and 3 are often combined, so you may see three segments instead of four.

A connection can also be aborted at once with an **RST** (reset) segment.

Q46. How does TCP guarantee reliable delivery?

INTERMEDIATE · TRANSPORT LAYER

- **Sequence numbers:** every byte is numbered, so the receiver can put segments back in order and drop duplicates.
- **Acknowledgements:** the receiver tells the sender the next byte it expects. ACKs are cumulative.
- **Retransmission timeout:** if no ACK arrives in time, the sender resends. The timeout adapts to the measured round-trip time.
- **Fast retransmit:** three duplicate ACKs for the same byte signal a lost segment, so the sender resends it without waiting for the timeout.
- **Checksum:** detects corrupted segments, which are discarded and then retransmitted.
- **Flow control and congestion control:** prevent losses caused by overwhelming the receiver or the network.

Q47. What is TCP flow control? Explain the sliding window.

INTERMEDIATE · TRANSPORT LAYER

Flow control stops a fast sender from overwhelming a slow receiver. The receiver advertises a **receive window (rwnd)** in every ACK: how many more bytes it has buffer space for.

The **sliding window** lets the sender transmit several segments without waiting for an ACK after each one, up to the window size. As ACKs arrive, the left edge of the window moves forward and new data can be sent, so the window "slides" along the stream.

- A larger window means more data in flight and higher throughput on long-distance links.
- If the receiver's buffer fills, it advertises a window of 0 and the sender pauses, sending small probes until space opens.

Without a window, a sender on a 50 ms link that waits for every ACK could send only one segment every 50 ms, no matter how fast the link is. The TCP tool on this page shows segments moving through a window, including a loss.

Q48. What is TCP congestion control?

INTERMEDIATE · TRANSPORT LAYER

Flow control protects the receiver. **Congestion control** protects the network. The sender keeps a **congestion window (cwnd)** and sends no more than the smaller of cwnd and the receiver's window.

- **Slow start:** cwnd starts small (often 10 segments today) and doubles every round trip, growing exponentially until it hits a threshold (ssthresh) or a loss.
- **Congestion avoidance:** after the threshold, cwnd grows by about one segment per round trip (additive increase).
- **On loss detected by duplicate ACKs (fast recovery):** cwnd is roughly halved and growth continues (multiplicative decrease).
- **On a timeout:** this suggests heavy congestion, so cwnd drops back to the starting size and slow start begins again.

This pattern is called **AIMD** (additive increase, multiplicative decrease). Linux uses the CUBIC algorithm by default, and Google developed BBR, which models bandwidth and delay instead of reacting to loss.

Q49. What is the difference between flow control and congestion control?

INTERMEDIATE · TRANSPORT LAYER

Point	Flow control	Congestion control
Protects	The receiver	The network in between
Problem it solves	Receiver's buffer overflowing	Routers' queues overflowing and dropping packets
Controlled by	Receive window (rwnd) sent by the receiver	Congestion window (cwnd) kept by the sender
Signal used	Explicit window size in each ACK	Inferred from loss, delay or ECN marks
Scope	Between two hosts	Whole path, shared with other flows

The sender's effective window is **min(rwnd, cwnd)**.

Q50. What are the main fields in the TCP and UDP headers?

INTERMEDIATE · TRANSPORT LAYER

TCP header (20 bytes minimum, up to 60 with options):

- Source port and destination port (16 bits each)
- Sequence number and acknowledgement number (32 bits each)
- Header length
- Flags: SYN, ACK, FIN, RST, PSH, URG (plus ECE and CWR for congestion signalling)
- Window size (receive window)
- Checksum and urgent pointer
- Options, such as maximum segment size (MSS), window scaling, timestamps and selective ACK (SACK)

UDP header (always 8 bytes): source port, destination port, length and checksum. That is all, which is why UDP is so light.

Q51. What is a socket?

INTERMEDIATE · TRANSPORT LAYER

A socket is one endpoint of a network connection, identified by an IP address and a port. Programs use the socket API to send and receive data without dealing with packets directly.

A TCP connection is uniquely identified by a **4-tuple**: source IP, source port, destination IP and destination port. That is how one web server on port 443 can handle thousands of clients at once: each client has a different IP and port pair.

The usual server calls are `socket()`, `bind()`, `listen()` and `accept()`. The client calls `socket()` and `connect()`. Both then `send()` and `recv()`, and finally `close()`. The example opens three client connections to one server port and prints each 4-tuple.

PYTHON (TESTED)

```
import socket

srv = socket.socket()                        # TCP by default
srv.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)   # restart safely while old sockets sit in TIME_WAIT
srv.bind(("127.0.0.1", 5000)); srv.listen()

clients = [socket.create_connection(("127.0.0.1", 5000)) for _ in range(3)]
tuples = set()
for n, c in enumerate(clients, 1):
    conn, _ = srv.accept()
    (cip, cport), (sip, sport) = conn.getpeername(), conn.getsockname()
    tuples.add((cip, cport, sip, sport))
    kind = "ephemeral" if cport >= 1024 else "well-known"
    print(f"connection {n}: client port is {kind}, server port is {sport}")
    conn.close(); c.close()
print(len(tuples), "different 4-tuples share server port 5000")
```

OUTPUT

```
connection 1: client port is ephemeral, server port is 5000
connection 2: client port is ephemeral, server port is 5000
connection 3: client port is ephemeral, server port is 5000
3 different 4-tuples share server port 5000
```

Q52. How does DNS resolution work, step by step?

INTERMEDIATE · APPLICATION LAYER

When you type `codeanddebug.in` into a browser:

1. The browser checks its own cache, then the operating system's cache and the `hosts` file.
2. If not found, the OS asks the configured **recursive resolver** (your ISP's, or a public one such as 8.8.8.8 or 1.1.1.1).
3. If the resolver has no cached answer, it asks a **root server**, which replies with the servers for `.in`.
4. The resolver asks a **.in TLD server**, which replies with the authoritative name servers for `codeanddebug.in`.
5. The resolver asks the **authoritative server**, which returns the A record: `64.227.174.41`.
6. The resolver caches the answer for its TTL and returns it to your computer, which caches it too.

Every cache along the way means most lookups take a few milliseconds. These are the real authoritative servers for our domain, the ones step 4 points to:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
dig +short codeanddebug.in NS
```

OUTPUT

```
ns-cloud-d1.googledomains.com.
ns-cloud-d3.googledomains.com.
ns-cloud-d4.googledomains.com.
ns-cloud-d2.googledomains.com.
```

Q53. What are the common DNS record types?

INTERMEDIATE · APPLICATION LAYER

Record	Purpose	Example
A	Name to IPv4 address	codeanddebug.in → 64.227.174.41
AAAA	Name to IPv6 address	example.com → 2606:2800:...
CNAME	Alias of another name	www.example.com → example.com
MX	Mail servers for the domain, with priority	1 aspmx.l.google.com
NS	Authoritative name servers for the domain	ns-cloud-d1.googledomains.com
TXT	Free text: domain verification, SPF, DKIM, DMARC	"v=spf1 include:_spf.google.com ~all"
SOA	Start of authority: primary server, admin, serial number, timers	One per zone
PTR	Reverse lookup: IP to name	Used by mail servers to check senders
SRV	Host and port for a service	_sip._tcp.example.com

A CNAME cannot exist at the root of a domain alongside other records, which is why providers offer ALIAS or flattened records for root domains.

Q54. What is the difference between recursive and iterative DNS queries?

INTERMEDIATE · APPLICATION LAYER

- **Recursive query:** the client asks a resolver for the final answer, and the resolver must do all the work and return either the answer or an error. Your laptop sends recursive queries to its resolver.
- **Iterative query:** the server replies with the best information it has, usually a referral ("I don't know, ask these servers"), and the asker follows up itself. The resolver sends iterative queries to root, TLD and authoritative servers.

Root and TLD servers only answer iteratively, because doing full recursion for the whole world would overload them.

Q55. What are the main HTTP methods?

INTERMEDIATE · APPLICATION LAYER

Method	Purpose	Safe	Idempotent
GET	Read a resource	Yes	Yes
HEAD	Like GET but returns only headers	Yes	Yes
POST	Create a resource or trigger processing	No	No
PUT	Create or fully replace a resource at a known URL	No	Yes
PATCH	Partly update a resource	No	Not guaranteed
DELETE	Remove a resource	No	Yes
OPTIONS	Ask which methods are allowed; used in CORS preflight	Yes	Yes

Safe means it does not change server state. **Idempotent** means sending it once or ten times has the same effect, so clients can retry it safely after a timeout.

Q56. What do the HTTP status code classes mean? Name the important codes.

INTERMEDIATE · APPLICATION LAYER

Class	Meaning	Common codes
1xx	Informational	101 Switching Protocols (WebSocket upgrade)
2xx	Success	200 OK, 201 Created, 204 No Content
3xx	Redirection	301 Moved Permanently, 302 Found, 304 Not Modified, 307 and 308
4xx	Client error	400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 429 Too Many Requests
5xx	Server error	500 Internal Server Error, 502 Bad Gateway, 503 Service Unavailable, 504 Gateway Timeout

Two favourites in interviews: **401** means "you are not logged in", while **403** means "you are logged in but not allowed". **502** means a proxy got a bad answer from the server behind it, and **504** means it got no answer in time. `curl -I` shows the status line and headers of a response; this is our blog:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
curl -I https://codeanddebug.in/blog/
```

OUTPUT

```
HTTP/2 200
server: nginx
date: Sun, 15 Nov 2026 07:22:31 GMT
content-type: text/html; charset=UTF-8
vary: Accept-Encoding
link: <https://codeanddebug.in/blog/wp-json/>; rel="https://api.w.org/"
x-cache: HIT
strict-transport-security: max-age=31536000
x-content-type-options: nosniff
x-frame-options: SAMEORIGIN
referrer-policy: strict-origin-when-cross-origin
permissions-policy: camera=(), microphone=(), geolocation=()
```

Q57. What does a raw HTTP request and response look like?

INTERMEDIATE · APPLICATION LAYER

HTTP/1.1 is plain text. A request is a request line (method, path, version), headers, a blank line, then an optional body. A response is a status line, headers, a blank line, then the body. The `Host` header is required in HTTP/1.1, because one server IP can host many websites.

The example starts a tiny web server on this machine and talks to it through a raw TCP socket, so you can see exactly what goes over the wire.

PYTHON (TESTED)

```
import socket, threading
from http.server import HTTPServer, BaseHTTPRequestHandler

class Hello(BaseHTTPRequestHandler):
    protocol_version = "HTTP/1.1"
    def do_GET(self):
        body = b"Hello from the server"
        self.send_response(200)
        self.send_header("Content-Type", "text/plain")
        self.send_header("Content-Length", str(len(body)))
        self.end_headers()
        self.wfile.write(body)
    def log_message(self, *args): pass
    def version_string(self): return "DemoServer"
    def date_time_string(self, timestamp=None): return "Sun, 15 Nov 2026 10:00:00 GMT"

httpd = HTTPServer(("127.0.0.1", 0), Hello)
threading.Thread(target=httpd.handle_request).start()

s = socket.create_connection(httpd.server_address)
s.sendall(b"GET /hello HTTP/1.1\r\nHost: localhost\r\nConnection: close\r\n\r\n")
reply = b""
while chunk := s.recv(4096):
    reply += chunk
print(reply.decode())
```

OUTPUT

```
HTTP/1.1 200 OK
Server: DemoServer
Date: Sun, 15 Nov 2026 10:00:00 GMT
Content-Type: text/plain
Content-Length: 21

Hello from the server
```

Q58. What is the difference between HTTP/1.1, HTTP/2 and HTTP/3?

INTERMEDIATE · APPLICATION LAYER

Point	HTTP/1.1	HTTP/2	HTTP/3
Transport	TCP	TCP	QUIC over UDP
Format	Plain text	Binary frames	Binary frames
Requests per connection	One at a time (browsers open about 6 connections)	Many at once (multiplexing)	Many at once
Header compression	None	HPACK	QPACK
Head-of-line blocking	At HTTP level	Fixed at HTTP level, still at TCP level	Removed: each stream is independent
Connection setup	TCP + TLS handshakes	TCP + TLS handshakes	Combined; 1 round trip, 0 for repeat visits

The browser and server agree on HTTP/2 during the TLS handshake using **ALPN**. Here is that negotiation with our server:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
curl -sv -o /dev/null https://codeanddebug.in/blog/
```

OUTPUT

```
* Connected to codeanddebug.in (64.227.174.41) port 443 (#0)
* ALPN, offering h2
* ALPN, offering http/1.1
* SSL connection using TLSv1.3 / TLS_AES_256_GCM_SHA384
* ALPN, server accepted to use h2
* Server certificate:
> GET /blog/ HTTP/2
> Host: codeanddebug.in
```

Q59. What is the difference between cookies, sessions and tokens?

INTERMEDIATE · APPLICATION LAYER

HTTP is **stateless**: each request is independent. These three mechanisms add memory between requests.

- **Cookie:** a small piece of data the server asks the browser to store (`Set-Cookie`) and send back with every request to that site (`Cookie`). Flags such as `HttpOnly`, `Secure` and `SameSite` make it safer.
- **Session:** the user's data is stored on the server, and the browser holds only a random session ID, usually in a cookie. Easy to revoke, but the server must store and share session data across machines.
- **Token (such as a JWT):** a signed token containing the user's claims, sent in the `Authorization` header. The server can verify it without a lookup, which suits APIs and mobile apps, but it is hard to revoke before it expires.

Q60. How does traceroute work?

INTERMEDIATE · NETWORK LAYER

Traceroute shows the path packets take to a destination, router by router. It uses the **TTL (Time To Live)** field in the IP header, which every router decreases by 1. When TTL reaches 0, the router drops the packet and sends back an ICMP **Time Exceeded** message.

1. Send probes with TTL = 1. The first router replies, revealing its address and the round-trip time.
2. Send probes with TTL = 2, then 3, and so on, revealing one more router each time.
3. Stop when the destination itself replies.

Linux `traceroute` sends UDP probes by default, while Windows `tracert` uses ICMP echo. A line of `* * *` means that router did not reply, which is often just a firewall rule, not a failure.

COMMAND (SYNTAX)

```
traceroute codeanddebug.in      # Linux and macOS
tracert codeanddebug.in         # Windows
tracepath codeanddebug.in       # Linux, no root needed
```

Q61. What is the difference between static and dynamic routing?

INTERMEDIATE · NETWORK LAYER

Point	Static routing	Dynamic routing
Routes set by	An administrator, manually	Routers, by exchanging information with protocols
Reacts to failures	No, until someone changes it	Yes, automatically
Overhead	None	CPU, memory and bandwidth for updates
Security	Predictable	Needs protection against fake updates
Best for	Small networks, default routes, stub networks	Large or changing networks
Examples	ip route add ...	RIP, OSPF, EIGRP, BGP

Q62. What is the difference between distance vector and link state routing?

INTERMEDIATE · NETWORK LAYER

Point	Distance vector	Link state
Each router knows	Distance and direction to each network, from its neighbours	The full map of the network
Shares	Its whole routing table with neighbours, periodically	The state of its own links with every router, when they change
Algorithm	Bellman-Ford	Dijkstra's shortest path
Convergence	Slow; can suffer from the count-to-infinity problem	Fast
Resources	Light	More memory and CPU
Examples	RIP (metric: hop count, maximum 15)	OSPF, IS-IS (metric: cost based on bandwidth)

Distance vector is "routing by rumour": each router trusts what its neighbours say. Split horizon, route poisoning and hold-down timers reduce the count-to-infinity problem.

Q63. What is BGP?

INTERMEDIATE · NETWORK LAYER

BGP (Border Gateway Protocol) is the routing protocol that connects the internet's independent networks, called **autonomous systems (AS)**, such as ISPs, cloud providers and large companies. Each AS has a number, and BGP routers announce which IP prefixes they can reach.

- It is a **path vector** protocol: each route carries the list of AS numbers it passed through, which prevents loops.
- It runs over TCP port 179.
- Route choice is driven by **policy** (business relationships, cost) more than by the shortest path.
- **eBGP** runs between different autonomous systems; **iBGP** runs inside one.

Mistakes in BGP have taken large parts of the internet offline, such as the 2021 Facebook outage, when its routes were withdrawn and its DNS servers became unreachable.

Q64. What is the difference between CSMA/CD and CSMA/CA?

INTERMEDIATE · DATA LINK LAYER

Both are rules for sharing a medium. Both start with **carrier sense**: listen before sending.

Point	CSMA/CD	CSMA/CA
Full form	Collision Detection	Collision Avoidance
Used in	Classic half-duplex Ethernet	Wi-Fi
Approach	Send, detect a collision, send a jam signal, wait a random backoff, retry	Wait a random backoff before sending; optionally reserve the channel with RTS/CTS; expect an ACK
Why	A wire can detect collisions while sending	A radio cannot hear collisions while it transmits, and hidden stations may not hear each other

Modern full-duplex switched Ethernet has no collisions, so CSMA/CD is mostly of historical interest, but it is still a common exam question.

Q65. What is a VLAN?

INTERMEDIATE · DATA LINK LAYER

A VLAN (Virtual LAN) splits one physical switch, or a set of switches, into several separate logical networks. Devices in different VLANs cannot talk directly, even if they are plugged into the same switch; traffic between VLANs must go through a router or layer 3 switch.

- **Benefits:** smaller broadcast domains, better security (guests separated from staff, cameras separated from servers), and grouping by team instead of by physical location.
- **Access port:** belongs to one VLAN and connects an end device.
- **Trunk port:** carries many VLANs between switches, adding an **802.1Q tag** with the 12-bit VLAN ID (up to 4,094 VLANs).

Q66. How does a switch learn MAC addresses?

INTERMEDIATE · DATA LINK LAYER

A switch builds a **MAC address table** (also called the CAM table) that maps MAC addresses to ports.

1. **Learn:** when a frame arrives, the switch records its *source* MAC and the port it came in on.
2. **Forward:** if the *destination* MAC is in the table, it sends the frame only out of that port.
3. **Flood:** if the destination is unknown, or is the broadcast address, it sends the frame out of every port except the one it came from.
4. **Filter:** if the destination is on the same port it came from, it drops the frame.
5. **Age out:** entries that are not refreshed expire, typically after 300 seconds.

When switches are connected in a loop, frames could circulate forever, so switches run **STP (Spanning Tree Protocol)** to block redundant links until they are needed.

Q67. How does the TLS handshake work?

INTERMEDIATE · SECURITY

TLS (Transport Layer Security) encrypts HTTPS and many other protocols. In **TLS 1.3**, the handshake takes one round trip:

1. **Client Hello:** supported TLS versions and cipher suites, a random value, and a key share for key exchange (usually elliptic-curve Diffie-Hellman). It also includes the site name (SNI) and the supported application protocols (ALPN).
2. **Server Hello:** the chosen cipher suite and the server's key share. Both sides can now compute the same shared secret, and everything after this is encrypted.
3. **Server certificate and signature:** the server proves it owns the private key for a certificate issued to that domain by a trusted certificate authority.
4. **Finished:** both sides confirm the handshake was not tampered with, then send application data with fast symmetric encryption such as AES-GCM.

TLS 1.2 needed two round trips. TLS 1.3 also removed old, weak algorithms, and with resumption can send data with zero round trips (0-RTT). This is the certificate our server presents:

COMMAND (REAL OUTPUT, 15 NOV 2026)

```
openssl s_client -connect codeanddebug.in:443 -servername codeanddebug.in </dev/null 2>/dev/null \
  | openssl x509 -noout -subject -issuer -dates
```

OUTPUT

```
subject=CN = codeanddebug.in
issuer=C = US, O = Let's Encrypt, CN = YR1
notBefore=Oct 26 23:58:10 2026 GMT
notAfter=Jan 24 23:58:09 2027 GMT
```

Q68. What is the difference between symmetric and asymmetric encryption?

INTERMEDIATE · SECURITY

Point	Symmetric	Asymmetric
Keys	One shared secret key	A public key and a private key
Speed	Very fast	Much slower
Main problem	How to share the key safely	Speed, and trusting that a public key is genuine
Used for	Encrypting the actual data	Key exchange and digital signatures
Examples	AES, ChaCha20	RSA, elliptic-curve (ECDH, ECDSA, Ed25519)

TLS uses both: asymmetric cryptography to agree on a key and prove the server's identity, then symmetric encryption for the data. **Hashing** (SHA-256) is different again: it is one-way and has no key to decrypt with.

Q69. What is the difference between a forward proxy and a reverse proxy?

INTERMEDIATE · APPLICATION LAYER

Point	Forward proxy	Reverse proxy
Sits in front of	Clients	Servers
Acts on behalf of	The user	The website
Hides	The client's identity from servers	The servers' identity and structure from clients
Typical uses	Office content filtering, caching, bypassing restrictions	Load balancing, TLS termination, caching, compression, security
Examples	Squid, corporate proxies	Nginx, HAProxy, Cloudflare

Our own site uses Nginx as a reverse proxy: it terminates HTTPS, serves the React app at `/`, passes `/blog` requests to WordPress and caches pages.

Q70. How are transmission errors detected and corrected?

INTERMEDIATE · DATA LINK LAYER

Detection:

- **Parity bit:** one extra bit makes the count of 1s even (or odd). It catches any single-bit error but misses two flipped bits.
- **Checksum:** add up the data in 16-bit words and send the complement. Used by IP, TCP and UDP. Simple, but weak against some error patterns.
- **CRC (Cyclic Redundancy Check):** treats the data as a polynomial and sends the remainder of a division. Very good at catching burst errors. Ethernet and Wi-Fi frames carry a CRC-32.

Correction: Hamming code adds several parity bits at power-of-two positions so the receiver can locate and fix a single-bit error. Other forward error correction codes (Reed-Solomon, LDPC) are used in Wi-Fi, 5G, QR codes and storage. The example below computes a CRC-32 and the 16-bit internet checksum, and shows a single flipped bit being caught.

PYTHON (TESTED)

```
import zlib

def internet_checksum(data: bytes) -> int:
    if len(data) % 2: data += b"\x00"
    total = sum(int.from_bytes(data[i:i+2], "big") for i in range(0, len(data), 2))
    while total >> 16:
        total = (total & 0xFFFF) + (total >> 16)
    return ~total & 0xFFFF

msg = b"Code and Debug"
bad = bytearray(msg); bad[0] ^= 0b00000001          # flip one bit: 'C' becomes 'B'

print("CRC-32  ok :", hex(zlib.crc32(msg)), "  corrupted:", hex(zlib.crc32(bytes(bad))))
print("Checksum ok:", hex(internet_checksum(msg)), "     corrupted:", hex(internet_checksum(bytes(bad))))
```

OUTPUT

```
CRC-32  ok : 0x1d001f4b   corrupted: 0x800ffe3d
Checksum ok: 0xce57      corrupted: 0xcf57
```

Q71. What are Stop-and-Wait, Go-Back-N and Selective Repeat?

INTERMEDIATE · DATA LINK LAYER

These are **ARQ (Automatic Repeat reQuest)** protocols for reliable delivery over a link that can lose frames.

Point	Stop-and-Wait	Go-Back-N	Selective Repeat
Frames in flight	1	Up to N	Up to N
Receiver accepts	Next frame only	Only in-order frames; discards the rest	Out-of-order frames, buffered
On a loss	Resend that frame	Resend the lost frame and everything after it	Resend only the lost frame
Efficiency	Low on long links	Good, wastes bandwidth on losses	Best
Complexity	Simplest	Medium	Highest; receiver needs buffers
Window size limit	1	$2^m - 1$	2^{m-1}

Here m is the number of bits in the sequence number. Modern TCP behaves like a mix of the last two: cumulative ACKs as in Go-Back-N, plus selective acknowledgements (SACK) so only missing data is resent.

Q72. What are MTU and fragmentation?

INTERMEDIATE · NETWORK LAYER

The **MTU (Maximum Transmission Unit)** is the largest packet a link can carry. Standard Ethernet MTU is 1,500 bytes. Subtracting 20 bytes of IP header and 20 bytes of TCP header leaves an **MSS (Maximum Segment Size)** of 1,460 bytes of data.

When a packet is larger than the next link's MTU:

- **IPv4** routers can split it into **fragments**, which are reassembled only at the destination. The Identification, Flags and Fragment Offset fields in the IP header make this possible.
- If the **Don't Fragment** flag is set, or in **IPv6**, the router drops the packet and sends an ICMP "packet too big" message instead.

Fragmentation hurts performance: losing one fragment means resending the whole packet. So hosts use **Path MTU Discovery** to find the smallest MTU on the path and send packets that fit.

Q73. What are FTP, SFTP and SSH?

INTERMEDIATE · APPLICATION LAYER

- **FTP (File Transfer Protocol):** transfers files using two TCP connections: a control connection on port 21 and a separate data connection (port 20 in active mode). Usernames, passwords and files are all sent in plain text, so it should not be used over the internet.
- **SSH (Secure Shell):** an encrypted protocol on port 22 for logging in to remote machines and running commands. It supports password and public-key login.
- **SFTP (SSH File Transfer Protocol):** file transfer that runs inside an SSH connection on port 22, so everything is encrypted. It is not FTP with encryption added.
- **FTPS:** the original FTP wrapped in TLS.

COMMAND (SYNTAX)

```
ssh deploy@203.0.113.10                    # log in to a server
scp app.zip deploy@203.0.113.10:/var/www/  # copy a file over SSH
sftp deploy@203.0.113.10                   # interactive SFTP session
```

Q74. How would you check a device's IP configuration, routes and open connections?

INTERMEDIATE · NETWORK LAYER

These are the commands interviewers expect you to know for first-level troubleshooting:

Task	Linux	Windows
IP addresses	`ip addr`	`ipconfig /all`
Routing table and default gateway	`ip route`	`route print`
ARP cache	`ip neigh`	`arp -a`
Open ports and connections	`ss -tulpn`	`netstat -ano`
DNS lookup	`dig`, `nslookup`	`nslookup`
Flush DNS cache	`resolvectl flush-caches`	`ipconfig /flushdns`
Reachability and path	`ping`, `traceroute`	`ping`, `tracert`

A good order for "the internet is not working": check the device has an IP and gateway, ping the gateway, ping a public IP such as 8.8.8.8, then ping a name. If the IP works but the name fails, the problem is DNS.

Q75. What is port forwarding?

INTERMEDIATE · NETWORK LAYER

Port forwarding is a NAT rule on a router that sends incoming traffic on a chosen public port to a specific device and port inside the private network. It is how you make a device behind NAT reachable from the internet.

Example: a rule forwards public port 8080 on the router to `192.168.1.50:80`, so a visitor opening `http://<public-ip>:8080` reaches the web server on that machine.

- It needs a stable private IP for the target, so use a DHCP reservation or a static address.
- Each forwarded port is exposed to the whole internet, so keep the service patched and forward only what you need.
- It does not work if your ISP uses **carrier-grade NAT (CGNAT)**, which is common on Indian mobile and many broadband connections, because your router does not have a real public IP.

Experienced level

What happens when you type a URL, load balancing, CDNs, QUIC, TCP performance, attacks and defences, cloud networking and troubleshooting scenarios.

Q76. What happens when you type a URL into the browser and press Enter?

EXPERIENCED · SCENARIOS

This is the most famous networking interview question because it touches every layer. For `https://codeanddebug.in/blog/`:

1. **URL parsing:** the browser reads the scheme (https), host and path. If you typed words instead of a URL, it searches instead. If the site is on the HSTS list, it upgrades to HTTPS before sending anything.
2. **DNS lookup:** browser cache, OS cache, hosts file, then the recursive resolver, which may ask root, TLD and authoritative servers. Result: 64.227.174.41.
3. **Routing to the server:** the destination is outside the local network, so the OS uses ARP to find the default gateway's MAC address and sends the packet there. Routers forward it hop by hop, and NAT rewrites the source address at your home router.
4. **TCP connection:** three-way handshake to port 443. With HTTP/3, a QUIC handshake over UDP replaces this step and the next.
5. **TLS handshake:** agree on keys, check the certificate, and pick HTTP/2 through ALPN.
6. **HTTP request:** `GET /blog/` with headers such as Host, cookies and Accept-Encoding.
7. **Server side:** a load balancer or reverse proxy (Nginx for us) receives it, may serve it from cache, or passes it to the application, which may query a database. The response comes back, often compressed.
8. **Rendering:** the browser parses HTML, builds the DOM, fetches CSS, JavaScript and images (reusing the connection), runs scripts and paints the page.

In an interview, start with this outline and then go deeper into whichever part the interviewer asks about.

Q77. What is a load balancer? What is the difference between layer 4 and layer 7 load balancing?

EXPERIENCED · SCALING

A load balancer spreads incoming traffic across several servers so no single server is overloaded, and it stops sending traffic to servers that fail their health checks.

Point	Layer 4 (transport)	Layer 7 (application)
Decides using	IP addresses and ports	HTTP data: URL path, host, headers, cookies
Sees content	No	Yes; usually terminates TLS
Speed	Faster, lower overhead	Slower, but smarter
Can do	Spread TCP or UDP connections	Path routing (/api to one pool, /images to another), header rewrites, caching, rate limiting
Examples	AWS NLB, LVS	AWS ALB, Nginx, HAProxy, Envoy

Common algorithms: **round robin**, **weighted round robin**, **least connections**, **least response time** and **IP hash** (keeps a client on the same server). If the app keeps state on a server, you need **sticky sessions**, though storing sessions in a shared cache such as Redis is the

better fix.

Q78. What is a CDN and how does it work?

EXPERIENCED · SCALING

A CDN (Content Delivery Network) is a large set of servers spread around the world, called **edge servers** or points of presence. They cache copies of a site's content so users download it from a nearby server instead of the origin.

1. A user's DNS lookup (or anycast routing) sends them to the nearest edge.
2. If the edge has the file cached and it is still fresh (a cache hit), it serves it immediately.
3. If not (a miss), the edge fetches it from the origin, stores it according to the `Cache-Control` headers, and serves it.

Benefits: lower latency, less load and bandwidth on the origin, protection against traffic spikes and DDoS attacks, and TLS handled close to the user. Examples: Cloudflare, Akamai, Amazon CloudFront. Purging or versioning file names (such as `app.3f9a.js`) handles content that changes.

Q79. What is QUIC and why does HTTP/3 use it?

EXPERIENCED · TRANSPORT LAYER

QUIC is a transport protocol that Google designed and the IETF standardised. It runs over **UDP** and builds reliability, congestion control and encryption (TLS 1.3) into one layer.

- **Faster setup:** transport and encryption handshakes are combined into 1 round trip, and 0 round trips when reconnecting to a known server.
- **No head-of-line blocking across streams:** in HTTP/2 over TCP, one lost packet stalls every stream, because TCP must deliver bytes in order. QUIC streams are independent, so a loss only delays its own stream.
- **Connection migration:** connections are identified by a connection ID instead of the IP and port 4-tuple, so a phone moving from Wi-Fi to mobile data keeps its connection.
- **Always encrypted:** even most of the header is protected, which also stops middleboxes from interfering.

It runs over UDP because new transport protocols cannot get through the many firewalls and NAT devices on the internet that only understand TCP and UDP. Improving TCP itself would need operating system updates everywhere.

Q80. What is head-of-line blocking?

EXPERIENCED · TRANSPORT LAYER

Head-of-line (HOL) blocking happens when the first item in a queue holds up everything behind it.

- **HTTP/1.1:** a connection handles one request at a time, so a slow response blocks the requests behind it. Browsers work around this by opening about six connections per site.
- **HTTP/2:** many streams share one TCP connection, which fixes the HTTP-level problem. But TCP delivers bytes strictly in order, so one lost packet makes every stream wait for the retransmission. On lossy mobile networks HTTP/2 can be slower than HTTP/1.1 for this reason.
- **HTTP/3 over QUIC:** each stream is ordered independently, so a loss only blocks its own stream.

Q81. What is the TIME_WAIT state? Why can too many connections in TIME_WAIT cause problems?

EXPERIENCED · TRANSPORT LAYER

The side that closes a TCP connection first stays in **TIME_WAIT** for twice the maximum segment lifetime (60 seconds on Linux). It does this to resend the final ACK if it was lost, and to make sure stray packets from the old connection cannot be confused with a new connection on the same 4-tuple.

The problem appears on busy clients, such as an application server that opens a new connection for every request to a database or API. Each closed connection holds its local port in TIME_WAIT, and with only about 28,000 ephemeral ports by default, the server can run out and fail with "cannot assign requested address".

Fixes: reuse connections with keep-alive and **connection pooling** (the real solution), widen the ephemeral port range, enable `net.ipv4.tcp_tw_reuse` for outgoing connections, or spread connections across more destination IPs. Do not try to remove TIME_WAIT itself; it exists for correctness.

COMMAND (SYNTAX)

```
ss -tan state time-wait | wc -l                   # count sockets in TIME_WAIT
sysctl net.ipv4.ip_local_port_range              # ephemeral port range
sysctl -w net.ipv4.tcp_tw_reuse=1                # reuse TIME_WAIT sockets for new outgoing connections
```

Q82. What are Nagle's algorithm and delayed ACK? Why do they cause latency together?

EXPERIENCED · TRANSPORT LAYER

- **Nagle's algorithm** reduces many tiny packets: while earlier data is still unacknowledged, the sender buffers small writes and sends them together.
- **Delayed ACK** reduces ACK traffic: the receiver waits up to about 40 ms (Linux) to 200 ms, hoping to piggyback the ACK on a reply or acknowledge two segments at once.

Together they can deadlock briefly: the sender waits for an ACK before sending its small final piece, and the receiver waits for more data before sending the ACK. Each request then gets an extra delay of tens of milliseconds. This often shows up in request-response protocols that write a header and a body in two separate small writes.

The fix for latency-sensitive applications is to disable Nagle with `TCP_NODELAY` (most HTTP clients, databases and games do), and to write each message in one call.

PYTHON (TESTED)

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
print("Nagle disabled by default?", bool(s.getsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY)))
s.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
print("Nagle disabled now?       ", bool(s.getsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY)))
```

OUTPUT

```
Nagle disabled by default? False
Nagle disabled now?        True
```

Q83. What is the bandwidth-delay product? Why does it matter?

EXPERIENCED · TRANSPORT LAYER

The bandwidth-delay product (BDP) is the amount of data that can be "in flight" on a link: bandwidth multiplied by round-trip time. To use the full bandwidth, TCP's window must be at least this big.

Example: a 100 Mbps link from Mumbai to a server in the US with a 250 ms RTT has a BDP of about 3 MB. If the window is only 64 KB, the sender sends 64 KB, then waits a whole round trip, so the maximum throughput is 64 KB ÷ 0.25 s, about 2 Mbps, on a 100 Mbps link.

That is why TCP has the **window scaling** option, which allows windows up to 1 GB, and why long-distance transfers benefit from tuned buffers or from moving servers closer to users.

PYTHON (TESTED)

```
bandwidth_bps = 100e6          # 100 Mbps
rtt_s = 0.250                  # 250 ms round trip

bdp_bytes = bandwidth_bps * rtt_s / 8
print(f"BDP: {bdp_bytes/1e6:.2f} MB must be in flight to fill the link")

window = 64 * 1024            # 64 KB, the maximum without window scaling
print(f"Max throughput with 64 KB window: {window * 8 / rtt_s / 1e6:.1f} Mbps")
```

OUTPUT

```
BDP: 3.12 MB must be in flight to fill the link
Max throughput with 64 KB window: 2.1 Mbps
```

Q84. What is the difference between polling, long polling, Server-Sent Events and WebSockets?

EXPERIENCED · APPLICATION LAYER

Technique	How it works	Direction	Good for
Short polling	Client asks every few seconds	Client pulls	Simple, infrequent updates; wastes requests
Long polling	Server holds the request open until there is news, then client asks again	Client pulls, near real time	Fallback when nothing else works
Server-Sent Events (SSE)	One long HTTP response that the server keeps writing events to	Server to client only	Live scores, notifications, streaming AI responses
WebSocket	HTTP request upgraded (status 101) to a persistent full-duplex connection	Both ways	Chat, multiplayer games, collaborative editing, trading screens

SSE is simpler, works through normal HTTP infrastructure and reconnects automatically. WebSockets are needed when the client also sends frequent messages. Both need care with load balancers and idle timeouts.

Q85. What is a DDoS attack? How is it mitigated?

EXPERIENCED · SECURITY

A DDoS (Distributed Denial of Service) attack floods a target with traffic from many machines, often a botnet of infected devices, so real users cannot get through.

- **Volumetric attacks** fill the bandwidth, often using **amplification**: small spoofed requests to open DNS, NTP or memcached servers trigger large replies aimed at the victim.
- **Protocol attacks** exhaust connection tables in servers and firewalls, such as SYN floods.
- **Application-layer attacks** send expensive but valid-looking requests, such as repeated searches or login attempts.

Mitigation: absorb traffic with a CDN or scrubbing service using anycast (Cloudflare, AWS Shield), rate limiting, SYN cookies, a WAF and bot detection for layer 7, caching to reduce origin load, autoscaling, and ISP-level blackholing for extreme cases.

Q86. What is a SYN flood? How do SYN cookies prevent it?

EXPERIENCED · SECURITY

In a **SYN flood**, an attacker sends a huge number of SYN segments, usually from spoofed IP addresses, and never completes the handshake. The server allocates memory for each half-open connection in its SYN queue, the queue fills up, and real users cannot connect.

SYN cookies remove the need to store anything for half-open connections. Instead of saving state, the server encodes it into the initial sequence number of its SYN-ACK: a cryptographic hash of the client's address and port, a timestamp and the MSS. A real client's final ACK carries that number + 1, so the server can check it and rebuild the connection. Spoofed SYNs never send an ACK, so they cost the server nothing.

Linux enables them automatically when the SYN queue overflows (`net.ipv4.tcp_syncookies = 1`).

Q87. What is a man-in-the-middle attack? How do HTTPS, certificates and HSTS prevent it?

EXPERIENCED · SECURITY

In a man-in-the-middle (MITM) attack, someone secretly sits between two parties, reading or changing their traffic. Common methods are a fake Wi-Fi hotspot, ARP spoofing on a LAN, or DNS spoofing.

- **Encryption (TLS)** stops the attacker reading or changing the data.
- **Certificates** stop the attacker pretending to be the server. A certificate binds a domain name to a public key and is signed by a **certificate authority** that the browser trusts. The browser checks the signature chain, the domain name, the validity dates and revocation. Without the private key, the attacker cannot present a valid certificate.
- **HSTS (HTTP Strict Transport Security)** stops **SSL stripping**, where the attacker keeps the victim on plain HTTP. The header tells the browser to use only HTTPS for that site for a set time. Our server sends `strict-transport-security: max-age=31536000`, which is one year.
- **Certificate Transparency** logs make wrongly issued certificates publicly visible.

Q88. How does the world move from IPv4 to IPv6?

EXPERIENCED · NETWORK LAYER

IPv4 and IPv6 cannot talk to each other directly, so several approaches run side by side:

- **Dual stack:** devices and networks run both IPv4 and IPv6 at the same time and prefer IPv6 when both ends support it. This is the most common approach. Browsers use "Happy Eyeballs" to try both and use whichever connects first.
- **Tunnelling:** IPv6 packets are wrapped inside IPv4 packets to cross IPv4-only networks (6in4, 6rd, Teredo), or the other way round.
- **Translation:** NAT64 with DNS64 lets IPv6-only devices reach IPv4-only servers. Many mobile networks, including large ones in India, run IPv6-only internally this way.

India has one of the highest IPv6 adoption rates in the world, mainly because of its mobile networks.

Q89. What are DNS TTL and DNS propagation? How do you change DNS safely?

EXPERIENCED · APPLICATION LAYER

Every DNS record has a **TTL (time to live)** in seconds. Resolvers and browsers cache the answer for that long before asking again.

"DNS propagation" is not really something spreading out. It is the time until all the cached copies of the old record expire. If a record has a TTL of 86,400 (one day), some users can keep getting the old IP address for up to a day after you change it.

Safe migration to a new server:

1. A day or more in advance, lower the TTL to something short, such as 300 seconds.
2. Wait for the old, longer TTL to expire everywhere.
3. Change the record. Keep the old server running, because some clients will still reach it for a few minutes.
4. Once traffic has moved, raise the TTL again to reduce lookups.

Q90. What is anycast and where is it used?

EXPERIENCED · NETWORK LAYER

With anycast, the **same IP address** is announced through BGP from many locations around the world. Internet routing naturally delivers each user's packets to the nearest location in network terms.

- **DNS:** the 13 root server names are served by well over a thousand anycast instances. Public resolvers such as 8.8.8.8 and 1.1.1.1 work the same way.
- **CDNs and DDoS protection:** attack traffic is spread across every location instead of hitting one.

Because routes can change mid-connection, anycast suits short UDP exchanges like DNS best. CDNs make it work for TCP too, since route changes are rare and connections are short.

Q91. How do peer-to-peer apps connect through NAT? What are STUN and TURN?

EXPERIENCED · NETWORK LAYER

Two devices behind NAT cannot normally connect to each other directly, because neither has a public address that accepts incoming connections. Video calls (WebRTC), online games and some file-sharing apps solve this with **NAT traversal**:

- **STUN:** each device asks a public STUN server "what public IP and port do you see me as?". The devices swap these addresses through a signalling server and then both send packets to each other at the same time, opening holes in their NATs (**hole punching**). This works for most home routers.
- **TURN:** when hole punching fails, for example with symmetric NAT or strict corporate firewalls, traffic is relayed through a TURN server. It always works but costs server bandwidth.
- **ICE:** the framework that tries direct, STUN and TURN candidates and picks the best one that works.

Q92. A website is slow for users. How would you find out whether the network is the problem?

EXPERIENCED · SCENARIOS

Break the request into stages and measure each one. `curl` can print the timing of every phase:

- **DNS lookup slow:** resolver problem or very low TTL; fix the resolver or raise the TTL.
- **TCP connect slow:** long distance or a congested path; check with `ping` and `mtr` (traceroute with loss per hop), consider a CDN or a closer region.
- **TLS handshake slow:** missing session resumption, long certificate chain, or no TLS 1.3.
- **Time to first byte slow, but connect fast:** the server or database is the bottleneck, not the network.
- **Download slow:** page too large, no compression, or packet loss limiting throughput.

Also compare users: if only one region, ISP or mobile network is slow, it is a routing issue. Then check server-side metrics, and use packet captures (`tcpdump`, Wireshark) to look for retransmissions.

COMMAND (SYNTAX)

```
curl -o /dev/null -s -w "dns: %{time_namelookup}s\nconnect: %{time_connect}s\ntls: %{time_appconnect}s\nfirst by
mtr -rwc 20 codeanddebug.in          # loss and latency at every hop
sudo tcpdump -i any host 64.227.174.41 -w capture.pcap   # open in Wireshark
```

Q93. What is a VPC? How are public and private subnets designed in the cloud?

EXPERIENCED · SCENARIOS

A VPC (Virtual Private Cloud) is your own isolated network inside a cloud provider such as AWS, Azure or Google Cloud. You choose its IP range (for example `10.0.0.0/16`) and divide it into subnets across availability zones.

- **Public subnet:** its route table sends `0.0.0.0/0` to an **internet gateway**. Load balancers and bastion hosts live here.
- **Private subnet:** no direct route from the internet. Application servers and databases live here. They reach the internet for updates through a **NAT gateway** in a public subnet, which allows outgoing connections only.

- **Security groups:** stateful firewalls attached to each instance ("allow 5432 only from the app servers").
- **Network ACLs:** stateless rules at the subnet level.
- **Peering, VPN or Direct Connect:** link the VPC to other VPCs or to an office network.

A typical three-tier layout: load balancer in public subnets, application in private subnets, database in separate private subnets that accept traffic only from the application.

Q94. What are HTTP keep-alive and connection pooling? Why do they matter?

EXPERIENCED · SCALING

Opening a new connection is expensive: one round trip for TCP plus one or two for TLS before any data moves. On a 100 ms link that is 200 to 300 ms of waiting per new connection.

- **HTTP keep-alive (persistent connections):** the same TCP connection is reused for many requests. It is the default in HTTP/1.1, and HTTP/2 goes further by multiplexing many requests over one connection at once.
- **Connection pooling:** an application keeps a set of open connections to a database or another service and lends them out per request, instead of connecting and disconnecting every time.

Benefits: lower latency, less CPU spent on handshakes, fewer sockets stuck in TIME_WAIT, and protection for databases that cannot handle thousands of new connections a second. Tools: PgBouncer for PostgreSQL, the pools built into HTTP client libraries, and keep-alive to the upstream servers in Nginx.

Q95. What is zero trust networking?

EXPERIENCED · SECURITY

The old model was "castle and moat": everything inside the company network was trusted, and the firewall at the edge was the main defence. Once an attacker got inside, for example through one phishing email, they could move freely.

Zero trust assumes no user or device is trusted just because of its network location. Every request is checked:

- Strong identity for users and services, with multi-factor authentication.
- Device health checks before access is granted.
- Least-privilege access to individual applications instead of the whole network.
- **Micro-segmentation**, so workloads can only talk to what they need.
- Encryption everywhere, including inside the data centre (for example mutual TLS between services).
- Continuous logging and monitoring.

Google's BeyondCorp is the best-known example, and many companies now use zero-trust access tools in place of a traditional VPN.

Q96. What is SDN (Software-Defined Networking)?

EXPERIENCED · SCALING

In traditional networking, every router and switch makes its own forwarding decisions and is configured one by one. SDN separates the two jobs of a network device:

- **Control plane:** decides where traffic should go. In SDN this moves to a central software **controller** with a view of the whole network.
- **Data plane:** actually forwards packets. It stays on the switches, which follow rules pushed by the controller (originally through the OpenFlow protocol).

Benefits: the network is programmable through APIs, changes are applied consistently everywhere, and traffic can be steered based on live conditions. Cloud providers build their virtual networks this way, and SD-WAN applies the same idea to links between company offices.

Q97. What is CORS and why do browsers enforce it?

EXPERIENCED · SECURITY

Browsers follow the **same-origin policy**: JavaScript on one origin (scheme + host + port) cannot read responses from another origin. Without it, any website you visit could call your bank's API using your logged-in cookies and read the results.

CORS (Cross-Origin Resource Sharing) lets a server relax this rule for origins it trusts by sending headers:

- `Access-Control-Allow-Origin: https://app.example.com` says which origin may read the response.
- For requests that are not "simple" (methods such as PUT or DELETE, a JSON content type, custom headers), the browser first sends a **preflight** `OPTIONS` request, and the server replies with the allowed methods and headers.
- `Access-Control-Allow-Credentials: true` is needed to send cookies, and then the origin cannot be the wildcard `*`.

CORS is enforced by the browser only. It does not protect an API from `curl` or other servers, so it is not a replacement for authentication.

Q98. What is rate limiting? Name common algorithms.

EXPERIENCED · SCALING

Rate limiting caps how many requests a client can make in a period. It protects services from abuse, brute-force logins, scraping and accidental overload, and keeps usage fair. Servers usually reply with **429 Too Many Requests** and a `Retry-After` header.

- **Fixed window:** count requests per minute. Simple, but allows a double burst at window edges.
- **Sliding window:** counts over a rolling period, which removes the edge burst.
- **Token bucket:** tokens refill at a steady rate up to a maximum; each request takes one. Allows short bursts while keeping the average rate. Widely used.
- **Leaky bucket:** requests drain from a queue at a fixed rate, smoothing traffic.

In the example, the first three requests use up the burst, the next two are refused, and by 1.0 seconds two tokens have refilled. In a system with many servers, counters are kept in a shared store such as Redis, or applied at the API gateway or load balancer.

PYTHON (TESTED)

```
import time

class TokenBucket:
    def __init__(self, rate, capacity):
        self.rate, self.capacity = rate, capacity
        self.tokens, self.last = capacity, 0.0
    def allow(self, now):
        self.tokens = min(self.capacity, self.tokens + (now - self.last) * self.rate)
        self.last = now
        if self.tokens >= 1:
            self.tokens -= 1
            return True
        return False

bucket = TokenBucket(rate=2, capacity=3)          # 2 requests per second, bursts of 3
times = [0.0, 0.1, 0.2, 0.3, 0.4, 1.0, 1.1, 1.2]
print([("ok" if bucket.allow(t) else "429") for t in times])
```

OUTPUT

```
['ok', 'ok', 'ok', '429', '429', 'ok', 'ok', '429']
```

Q99. Design the network for a company's new office of 120 people.

EXPERIENCED · SCENARIOS

Interviewers want to see that you think about requirements, separation, addressing and resilience. A sensible answer:

1. **Requirements:** 120 staff with laptops and phones, guest Wi-Fi, printers, CCTV, a few on-site servers, and access to cloud apps and other offices.
2. **Segmentation with VLANs:** Staff, Guest, Voice, Printers and IoT, CCTV, Servers, Management. Guests get internet only.
3. **Addressing:** use `10.20.0.0/16` for the site, with a subnet per VLAN sized with growth in mind, for example a `/23` (510 hosts) for staff devices, since each person may have two or three.
4. **Hardware:** access switches with PoE for access points and phones, uplinked to a core layer 3 switch that routes between VLANs. Enterprise Wi-Fi access points with a controller.
5. **Edge:** a firewall with two internet links from different ISPs for failover, NAT for outbound traffic, and a site-to-site VPN to head office.
6. **Services:** DHCP per VLAN, internal DNS, NTP, and WPA3-Enterprise (802.1X) for staff Wi-Fi.
7. **Operations:** monitoring, logging, configuration backups, and documented addressing.

Q100. Users can reach websites by IP address but not by name. How do you troubleshoot it?

EXPERIENCED · SCENARIOS

If the IP works but the name does not, routing and the internet connection are fine, and the problem is almost certainly **DNS**.

1. **Confirm:** `ping 8.8.8.8` works, but `ping example.com` fails with "could not resolve host".
2. **Check which DNS servers are configured:** `ipconfig /all` on Windows, or `resolvectl status` on Linux. A wrong address often comes from a bad DHCP setting.
3. **Query directly:** `nslookup example.com`, then compare with a public resolver: `nslookup example.com 8.8.8.8`. If the public one works, the local DNS server is the problem.
4. **Check the firewall:** UDP and TCP port 53 to the resolver must be allowed.
5. **Check the cache and hosts file:** flush the DNS cache and look for wrong entries in the hosts file.

6. **If only one domain fails:** check its records with `dig`, its name servers and DNSSEC.

COMMAND (SYNTAX)

```
ping -c 2 8.8.8.8                # internet reachable by IP?
nslookup example.com             # using the configured resolver
nslookup example.com 8.8.8.8     # using a public resolver
ipconfig /flushdns               # Windows: clear the DNS cache
```

Keep practising

Revise one level at a time, and explain each answer out loud as if you were in the interview.

Use the packet journey, TCP simulator, subnetting calculator and DNS walkthrough at codeanddebug.in/blog/computer-networks-interview-questions to see the hardest topics in action.

Cover the other core subjects with 100 DBMS Interview Questions and 100 SQL Interview Questions at codeanddebug.in/blog.

Watch the video explanations on YouTube: @codeanddebug.