1. IP Foundation & Addressing

Layers:
Application (URL,email), Transport (TCP/UDP process to process), Network (IP routing IPv4 32b/IPv6 128b), Link (MAC,Ethernet/Wi-Fi), Physical (bits).

When search google.com:
(1) DHCP assigns IP/DNS/gateway, (2) ARP resolves router MAC, (3), DNS resolves domain, (4) TCP 3-way handshake→HTTP request/response.

Subnetting: (1) First addr = network ID, last = broadcast, (2) Classful A(/8),B(/16),C(/24), (3) CIDR /x for custom prefix, (4) Subnet splits net, supernet merges nets (aggregation).

IPv4 Header: (1) 20 B min, (2) HLEN 4 bits (x4 bytes (e.g. 5 means 20B), (3) Total len = payload (bytes) + header length (bytes), (4) Flags DF/MF, (5) Frag offset in 8B units (e.g if offset = 232 bytes, then frag offset = 232 / 8 = 9), (6) Big-endian: Transmission is row by row, big endian byte order, (7) DS/ECN (1 B): 6-bit DS QoS + 2-bit ECN, (8) MTU limits frame size (≥46 B Ethernet), (9) Too large: fragment at sender or router; reassemble at destination.

2. ARP and DHCP

ARP: **(1)** Maps IP to MAC in LAN, **(2)** ARP Request: Broadcast, restricted to within a LAN, **(3)** ARP Reply: Unicast. Cache stores mappings for all hosts within a single LAN, **(4)** Proxy ARP: With Proxy ARP, a router (or another host) answers on behalf of the real host. The router then forwards the packet to the correct destination on the other subnet, **(5)** ARP request to non-existing host: Keep sending several ARP requests until give up, **(6)** Gratuitous ARP: Host sends ARP request for its own IP address, to detect if IP address already assigned and to update ARP caches about its IP to MAC mapping, **(7)** Why ARP may be unsafe: No auth, stateless, ARP poisoning (manipulate ARP cache)

DHCP:
- Server UDP 67, client UDP 68.
- DORA: Discover (broadcast) → Offer → Request (broadcast) → Ack. On Ack, client sends 3 ARP probes (gratuitous ARP) to ensure no one uses the IP, and announcement ARP for the IP to broadcast to LAN to update ARP caches. Server uses least recently used address alloc, performs conflict detection using ICMP ping. Lease renew at 50 %.
- Relay Agent forwards broadcasts across subnets.
- Fields: op(1=request/2=reply).
- Timers: T1=0.5 lease, T2=0.875 lease, expiry=1.0.

3. Framing, Switching & VLANs

VLAN = logical L2 segmentation of a physical LAN into multiple broadcast domains within the same or across connected switches. Devices in the same VLAN behave as if on same LAN even across switches. VLANs separate networks at Layer 2 (Data Link), while subnetting separates at Layer 3 (IP layer).

LAN vs VLAN:
LAN = one broadcast domain; VLAN = multiple virtual broadcast domains within one physical LAN. Each VLAN has its own VLAN ID (1-4094) and is configured in switch software. Broadcasts stay inside each VLAN, improving performance and isolation.

Single-Switch VLANs:
One switch divided into multiple VLANs. Each VLAN forms its own broadcast domain. Hosts in different VLANs need a router or Layer 3 switch for inter-VLAN communication.

Trunking (Inter-Switch VLANs):
Trunk links carry frames for multiple VLANs between switches. Implemented using IEEE 802.1Q tagging - adds a 4-byte VLAN tag between source MAC and EtherType field to identify VLAN membership. VLAN tag includes 12-bit VLAN ID, 3-bit Priority (PCP), and 1-bit Drop Eligible Indicator (DEI).

Frame Filtering vs Frame Tagging:
Access ports (end devices) are untagged- switch assigns VLAN internally. Trunk ports (switch to switch or switch to router) use tagging so intermediate devices know VLAN membership. Ingress switch adds tag, intermediate switches forward using tag, egress removes tag before delivery.

Inter-VLAN Routing:
Different VLANs cannot directly communicate (different broadcast domains). Routing is done via:
- **Router-on-a-Stick:** Single router interface split into subinterfaces, each mapped to one VLAN over a trunk link. Efficient, low-cost inter-VLAN routing.
- **Layer 3 Switch:** Performs routing in hardware (ASIC), faster for enterprise networks.

Static vs Dynamic VLANs:
Static VLAN: port-based, manually configured on switch (fixed mapping of ports→VLAN). Dynamic VLAN: membership decided by MAC address, protocol, or user identity (requires VLAN Management Policy Server, e.g., VMPS).

MPLS Overview:
MPLS = "Layer 2.5" forwarding based on labels, not IP lookups. **LER** (Label Edge Router) adds/removes labels; **LSR** (Label Switch Router) forwards by label. Enables faster switching, traffic engineering (explicit routing), QoS prioritization, and supports multiple protocols (IP, ATM, Frame Relay). Used in ISPs for scalability and VPNs.

4. Network Applications

Principles:
Server well-known port, client ephemeral. TCP (reliable stream, flow + congestion ctrl), UDP (unreliable message). SSL/TLS adds encryption. TCP stateful; HTTP stateless.

HTTP Versions: (1) 1.0: new TCP per req, no persistent conn, minimal caching, no host header, (2) 1.1: persistent conn, Host header, pipelining → HOL (HOL is due to FCFS), (3) 2: multiplexing on 1 TCP, binary, HPACK, server push, still TCP HOL, (4) 3: QUIC over UDP, no HOL, TLS 1.3 integrated, 0-RTT.
- Cookies ('Set-Cookie', 'Cookie') store state, Conditional GET ('If-Modified-Since', ETag) → 304 Not Modified. Caches use 'Cache-Control', 'Expires'.

Email:
- SMTP (push, TCP 25/587), POP3 (pull, TCP 110/995), IMAP (sync, TCP 143/993). MTA (sends), MAA (retrieves). POP deletes after download; IMAP keeps server copy.
- e.g: MTA use SMTP to push email, MAA use POP or IMAP to pull email

5. Network Application Development

Socket API:
1. TCP (SOCK_STREAM): reliable byte stream (5-tuple: src/dst IP, port, proto), 1 socket per connection
2. UDP (SOCK_DGRAM): message-based (no connection).
3. Raw sockets (SOCK_RAW): for custom IP/ICMP, need root.

Fields: family (AF_INET6), type, protocol, local + remote addr.

DNS Basics:
- Tree structure (root → TLD → SLD → subdomain). Example 'www.cs.nus.edu.sg'. Root knows TLDs, TLDs know authoritatives. Resolver (client) → Local DNS → Root → TLD → Authoritative.
- Recursive (query chain done by resolver) vs Iterative (referral chasing). Domain = entire subtree; Zone = portion managed by one server. Primary (master) stores zone file; Secondary (slave) copies via AXFR.
- Default UDP 53 (TCP 53 for large/zone). RR: A, AAAA, NS, CNAME, SOA, PTR, MX, TXT. Reverse lookup in 'in-addr.arpa' (PTR).
- Header (12 B) + sections: Question, Answer, Authority, Additional. Flags: QR, AA, TC, RD, RA, RCODE. Tools: 'dig', 'nslookup', 'host', 'whois'.

6. Wireless LAN (802.11)

CSMA/CD vs CSMA/CA:
- **CSMA/CD (Ethernet, 802.3):** Detects collisions after they occur → sender stops, sends jam signal, exponential backoff. Works only in wired medium (simultaneous transmit + detect possible).
- **CSMA/CA (WiFi, 802.11):** Collision *avoidance* - cannot detect while transmitting (half-duplex, hidden terminals). Steps: Sense channel → wait IFS → if busy → random backoff → transmit when idle. Receiver sends ACK → confirms success (since sender can't detect collision). **Hidden node:** A and C both send to B but can't sense each other → collision at B. Solution: RTS/CTS handshake reserves medium before data; nearby stations hearing RTS or CTS defer.

Exposed node: B sends to A; C overhears and defers even though it could send to D (no real conflict). No perfect fix

IFS (Inter-Frame Spacing) Priorities Different IFS values create transmission priority levels: SIFS < PIFS < DIFS **(1) SIFS (Short IFS):** Highest priority - used by ACK, RTS/CTS, and immediate responses, **(2) PIFS (Point IFS):** Used by AP during polling (PCF mode), **(3) DIFS (DCF IFS):** Used by normal stations before contention access.

Access Modes:
1. **DCF (Distributed Coordination Function):** Default mode - contention-based CSMA/CA + ACK. Each station contends independently using random backoff.
2. **PCF (Point Coordination Function):** Optional centralized mode - AP acts as coordinator, polls stations during *Contention-Free Period (CFP)*. CFP + CP (Contention Period) alternate for fairness. AP uses shorter IFS (PIFS) to gain priority access. **NAV (Network Allocation Vector)** announces reserved channel time (duration field in RTS/CTS).

7. IP Options & ICMP

IPv4 Options:
- 0-40 B extension in header (rare)
- Common: Record Route (RR) appends router IPs (max 9 entries), Strict/Loose Source Route (SSR/LSR) fix/guide path, Timestamp records router times.
- Increases latency; often blocked.

ICMP:
- Encapsulated in IP, not for user data.
- Used for errors + diagnostics.
- Error types: 3 (Destination Unreachable - network, host, port, proto), 4 (Source Quench obsolete), 5 (Redirect), 11 (TTL Expired), 12 (Bad Header). Query types: 8/0 (Echo Req/Reply, Ping), 13/14 (Timestamp), 17/18 (Address Mask).
- Traceroute: sends UDP packets w/ increasing TTL → routers reply Type 11 (TTL expired) until dest (Type 3).

Network Address Translation (NAT):
- Maps private IPs inside a LAN to a single public IP for Internet access, conserving address space.

- Why breaks end-to-end connectivity: NAT hides internal hosts behind one public address, so external hosts cannot directly initiate connections to private devices. NAT doesn't know which internal device should receive the incoming packet.

- Normal Operation (Client → Public Server): When internal client contacts an external server, the NAT creates a mapping between its internal (private IP, port) and a new public (NAT IP, port). This mapping is reactive.

Problem: Accessing a Server Behind NAT

External hosts can't start a connection because NAT blocks unsolicited inbound traffic.

Solutions:

1. Port Forwarding: Manually configure the NAT to map a public port to an internal host/port.

2. Connection Reversal: The internal server first connects outward, allowing the external client to communicate through that established mapping.

Problem: Both Hosts Behind NAT

Neither side can initiate directly because both are hidden.

Solutions:

1. Relay through a Proxy/Server: A public relay forwards packets between both private hosts (e.g., TURN server).

2. Hole Punching: Both hosts simultaneously send UDP packets to each other via a coordination server, creating NAT mappings that allow direct peer-to-peer communication (used in VoIP, WebRTC).

8. OSPF (Open Shortest Path First)

Type: Link-State, intra-domain routing (within AS). Protocol No. 89 -runs directly on IP (no TCP/UDP). Each router learns full area topology (link states) and runs Dijkstra's SPF to compute shortest paths. Faster convergence, no loops or count-to-infinity (vs DVR).

Hierarchy & Structure: OSPF divides an AS into **areas** to reduce LSA flooding. Area 0 = backbone interconnecting all other areas. Routers: - **Internal:** All interfaces in one area.

- **ABR:** Connects multiple areas, summarizes LSAs between them.

- **ASBR:** Redistributes external routes (e.g. RIP/BGP).

- **Backbone Router:** In Area 0.

Databases:

1. **Adjacency DB** - list of directly connected neighbours (from Hello).

2. **Link-State DB (LSDB)** - identical copy of area topology (synchronized across routers).

3. **Routing Table** - built by running Dijkstra on LSDB.

Hello Protocol: Hello packets (every 10s) sent to multicast 224.0.0.5. Contains Router ID, area ID, intervals, neighbour list. If no Hello within 40s (dead interval) → neighbour considered down. Used only to discover/maintain adjacencies (not for routing).

Link-State Advertisements (LSAs): Advertise router links and costs. Sent when topology changes or every 30min. Flooded reliably within an area using seqnum, age, ACKs.

Types:

1. **Router LSA:** Each router lists its links

2. **Network LSA:** Sent by DR/BDR on multi-access networks

3. **Summary LSA:** ABR advertises networks between areas

4. **External LSA:** ASBR advertises routes from other ASes.

DR/BDR Election: On broadcast networks (Ethernet). Highest priority wins; tie → highest Router ID. Routers form adjacencies only with DR/BDR → all LSAs sent to DR/BDR, which multicasts to others (reduces flooding).

Database Exchange (New Router Join): (1) DR → Database Description (list of LSAs), (2) New router → Link State Request (asks for missing LSAs), (3) DR → Link State Update (sends full LSAs), (4) Router → Link State Acknowledgment. Then floods its own LSAs → DR/BDR → area multicast.

Link Types: (1) Point-to-Point: two routers directly connected, (2) Broadcast: multi-access LANs (needs DR/BDR), (3) Stub: one-router network (no transit), (4) Virtual: logical link used to restore backbone connectivity.

Operation Flow: Hello → Build Adjacency → Exchange LSAs → Flood LSDB → Run Dijkstra → Update Routing Table.

Administrative Distance: Lower = more trusted. OSPF = 110, RIP = 120, EIGRP = 90, BGP (external) = 20.

9. Network Security

Core Services:

1. Confidentiality: keeps data secret using symmetric/asymmetric encryption $E_K(m)$.

2. Integrity: ensures message not altered (via hash or HMAC).

3. Authentication: verifies sender identity (MAC or digital signature).

4. Non-repudiation: sender cannot deny sending - achieved using digital signatures (public key distributed by trusted third party)

5. Entity Authentication: challenge-response using nonce R (A proves identity to B by signing R).

6. Availability: prevent DDoS (e.g., filter TCP SYN floods, block sniffing).

Message Authentication: **(1)** Message Authentication Code (MAC): hash of message + shared key for integrity + auth, **(2)** Digital Signature: sender encrypts message digest with private key, receiver verifies with sender's public key.

Key Distribution:

Symmetric Key: Key Distribution Centre (KDC) shares trusted session keys. Diffie-Hellman: both sides exchange public params (p, g) and compute same shared key.

Asymmetric Key: Certification Authority (CA) binds a public key to an entity, ensures authenticity of public keys.

TLS (Transport Layer Security):

Provides confidentiality (symmetric), integrity (hash), and authentication (public key). Successor to SSL. Handshake (TLS 1.3): ClientHello (supported algos, DH params) → ServerHello (choice + cert) → both derive shared session key via Diffie-Hellman. 0-RTT mode allows early data but vulnerable to replay. Forward secrecy: DH provides new key per session; RSA alone does not.

IPsec (Network Layer Security): Protects IP packets. Transport mode: encrypts only payload (not IP header). Tunnel mode: encrypts entire packet → used in VPNs. AH = integrity + auth; ESP = encryption + integrity.

PGP: A sends $K_S(K_A^-(H(m)) + m) + K_B^+(K_S)$, B decrypts with B's private key to get shared symmetric key, Use shared key to decrypt MAC, Use A's public key to get H(m) + m, Hash m to verify integrity

10. TCP Congestion Control

Overview: TCP = reliable, in-order byte stream, full-duplex, connection-oriented. Uses ACKs for bytes, rwnd for flow control, and cwnd to prevent network overload. Effective send window = min(rwnd, cwnd). MSS = max segment size; 3 DupACK → Fast Retransmit; Timeout → Slow Start.

Connection Control: 3-way handshake (SYN → SYN+ACK → ACK) establishes state; 4-way FIN handshake terminates. 2MSL timer prevents delayed duplicates. SYN/FIN consume seqno; pure ACK does not; Half-close: one side ends sending (FIN) but can still receive; Simultaneous open: both sides send SYN at same time; Deny connection: send RST (refuse); Abort connection: send RST+ACK (terminate abruptly).

Flow Control (Receiver-based): Receiver advertises **rwnd** = buffer space available. Sender sends ≤ min(rwnd, cwnd). Receiver increases rwnd as it consumes data → sender adapts. Throughput ≈ $\frac{\text{Window Size}}{\text{RTT}}$.

A. TCP Tahoe Phases: (1) Slow Start: *cwnd* starts at 1 MSS, doubles each RTT until *ssthresh*. ⇒ *cwnd* doubles per ACK (exponential growth). **(2) Congestion Avoidance:** when $cwnd \geq ssthresh$, linear growth. ⇒ *cwnd* += 1 MSS (linear) per RTT (additive increase), **(3)On Loss/Timeout:** $ssthresh = cwnd/2$, $cwnd$ = 1 MSS → restart Slow Start. (Additive Increase, Multiplicative Decrease) (AIMD).

Drawback: Full restart on loss → underutilization.

B. TCP Reno: Adds **Fast Retransmit** + **Fast Recovery**: **(1)** On 3 duplicate ACKs → assume segment lost, **(2) Fast Retransmit:** retransmit immediately, **(3) Fast Recovery:** $ssthresh = cwnd/2$, $cwnd = ssthresh + 3 \times MSS$. For each extra DUP ACK → $cwnd += MSS$. When ACK for new data arrives → $cwnd = ssthresh$. - Avoids full Slow Start after minor loss. Slow Start is more aggressive (faster cwnd growth).

C. TCP NewReno (RFC 2581): Improves Reno by handling multiple losses in a single window. **(1)** On 3 DUP ACKs: $ssthresh = \max(cwnd/2, 2 \times MSS)$, **(2)** Retransmit lost segment; $cwnd = ssthresh + 3 \times MSS$, **(3)** For each extra DUP ACK → $cwnd += MSS$, **(4)** Send new data if allowed, **(5)** On ACK for new data → $cwnd = ssthresh$

- During Fast Recovery, *cwnd* artificially inflated by in-flight segments. Ends recovery only when ACK covers all outstanding data.

D. TCP CUBIC (Default in Linux ≥ 2.6.8): **(1)** Loss-based algorithm; *cwnd* grows as a cubic function of time: $cwnd \propto (t-K)^3$, **(2)** Rapid growth when far from previous W_{max}, slower near W_{max}, **(3)** Provides better throughput and fairness on high BDP (bandwidth-delay product) links.

E. Delay-Based Algorithms (e.g., TCP Vegas, TCP BBR): Use RTT variations as early signs of congestion (before packet loss).

- **Vegas: (1)** If RTT increases → queues filling → decrease rate, **(2)** If RTT decreases → underutilized → increase rate, **(3)** Goal: keep pipe "just full, not overfull."

- **BBR (Model-based): (1)** Estimates bottleneck bandwidth (BtlBw) and minimum RTT, **(2)** Maintains *cwnd* ≈ BtlBw × RTT (pipe size), **(3)** Insensitive to loss; operates in phases: Startup, Drain, ProbeBW, ProbeRTT, **(4)** Achieves higher throughput than loss-based TCPs (CUBIC, Reno) on LFNs.

F. Network-Assisted Congestion Control (ECN): **(1)** Routers **mark** (not drop) packets when nearing congestion, **(2)** Uses 2 bits in IP ToS (Type of Service) field

Fairness: If K TCP flows share a bottleneck of rate R, each gets ≈ R/K (assuming equal RTTs). AIMD ensures proportional fairness; UDP (no congestion control) can starve TCP.

Command Example: `iperf3 -c 192.168.1.1 -C cubic`

11. Software-Defined Networking (SDN)

Motivation: Traditional networks = distributed control plane (each router runs OSPF/BGP, makes local decisions). SDN **separates control plane (logic)** from **data plane (forwarding)**. Controller = centralised "network brain" with global view; switches = simple forwarding devices. Enables programmability, dynamic policy, and easier management.

Architecture:

1. **Data Plane:** Switches/routers that forward packets based on flow tables.

2. **Control Plane:** SDN Controller computes routes, installs forwarding rules in switches.

3. **Application Plane:** Network apps (load balancing, firewalls, monitoring) use controller APIs. Communication uses southbound (controller→switch) and northbound (controller to apps) interfaces.

OpenFlow Protocol: Standard southbound API between controller and switches. Controller installs flow entries: (match fields, actions, priority, counters, timeout). Flow table = match (IP, MAC, port) → action (forward/drop/modify). If no match → packet-in → controller installs rule.