Networking System Foundations – Notes

October 22, 2025

Contents

1 TCP/IP Stack at a Glance

These notes follow the pragmatic four-layer TCP/IP view: *Link* (local delivery on a medium), *Network/Internet* (best-effort routing across networks via IP), *Transport* (end-to-end delivery, multiplexing, reliability/congestion control), and *Application* (protocols and data models used by programs). The same encapsulation/decapsulation steps repeat on every transmission: the application's bytes are wrapped by a transport header (e.g., TCP/UDP), then by an IP header, then by a link-layer frame; at the receiver, each layer strips its header and passes the payload up.

What each layer is responsible for

- **Application:** Naming, message formats, semantics (HTTP, DNS, SMTP, SSH, …).
- **Transport:** Ports, connections/sessions, reliability, flow & congestion control (TCP, UDP, QUIC*).
- **Network/Internet:** Logical addressing and routing across networks (IPv4/IPv6, ICMP; ARP/NDP at boundary).
- **Link:** Local delivery over a specific medium; framing, MAC addressing, access control (Ethernet, 802.11, PPP).

**QUIC lives in user space over UDP but provides transport semantics at the application layer.*

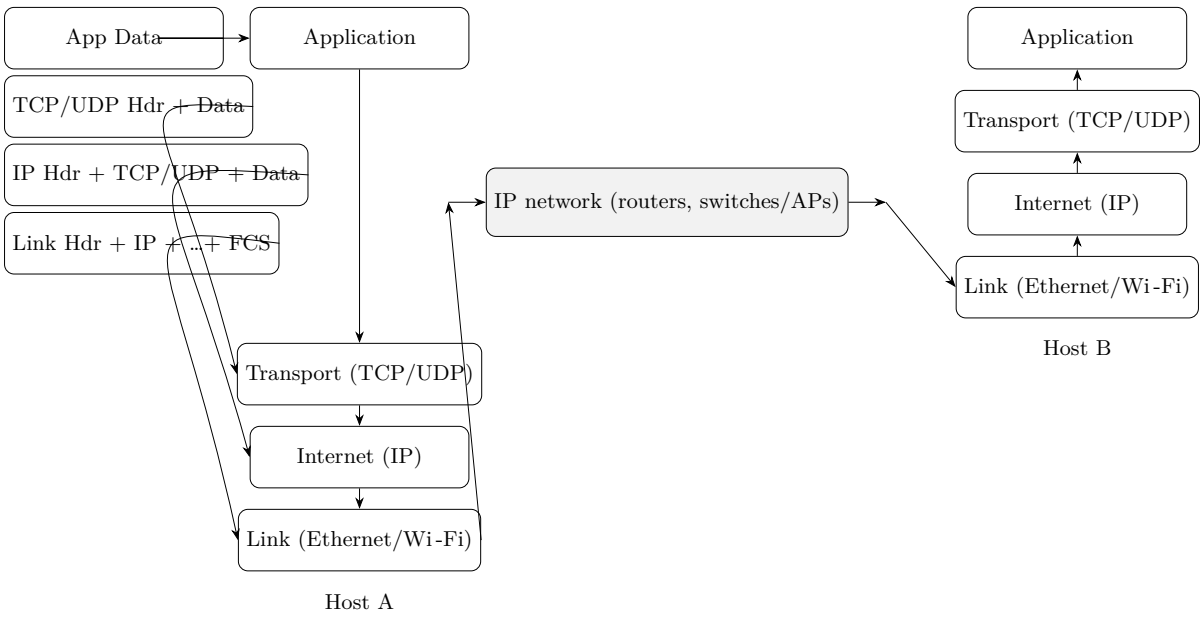


Figure 1: How the four TCP/IP layers interact: encapsulation at the sender, best-effort routing across the IP network, and decapsulation at the receiver.

2 Link Layer

2.1 Introduction to Links

The **link layer** is responsible for enabling direct communication between nodes that share a common physical medium (wired or wireless). A *link* is the physical and logical channel that connects two adjacent devices in a network.

Links and Nodes

Networks consist of *nodes* (hosts, routers, switches) interconnected by *links*. These links may be:

- **Wired** – copper cables (Ethernet, coaxial) or fiber optics
- **Wireless** – radio (Wi-Fi), infrared, or satellite communication

Each link provides the raw capability to transfer bits (1s and 0s).

Where the Link Layer Lives

The link layer is typically implemented in the **Network Interface Card (NIC)**:

- Provides the hardware logic to format data into frames
- Implements low-level protocols (e.g., Ethernet, Wi-Fi)
- Interacts with the host system over a bus (e.g., PCI)

Thus, the link layer bridges the computer's memory/CPU with the physical medium.

Problems Addressed by the Link Layer

The link layer solves key challenges in direct device-to-device communication:

1. **Encoding and addressing** – how to mark a transmission so the receiver knows who it is for and how to interpret it.
2. **Error detection (and correction)** – ensuring that corrupted bits can be detected, and in some cases corrected.
3. **Medium access control (MAC)** – managing access when multiple nodes share the same medium.
4. **Extending connectivity** – enabling local area networks (LANs) that scale beyond the limitations of a single physical medium.

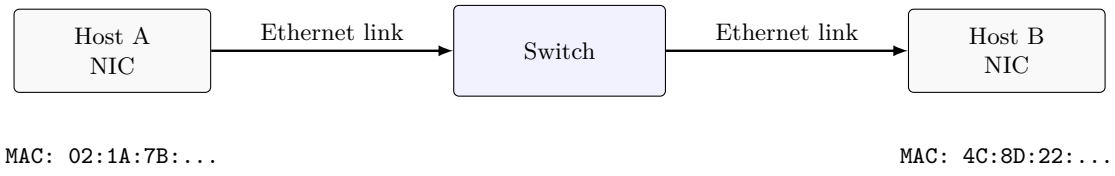


Figure 2: Basic node–link setup with hosts connected via a switch at the link layer.

Framing

Since raw streams of bits are meaningless, the link layer introduces *frames*:

- A **header** with metadata such as source and destination addresses
- A **payload** containing the actual data from higher layers
- A **trailer** (e.g., Frame Check Sequence) to assist with error detection

This structure allows receivers to recognize where a transmission starts and ends, and to verify its correctness.

Example: Ethernet Frames

An Ethernet frame typically contains:

- **Destination Address** (48-bit MAC)
- **Source Address** (48-bit MAC)
- **EtherType** (16-bit field indicating higher-layer protocol, e.g., 0x0800 = IPv4)
- **Payload** (up to 1500 bytes for standard Ethernet)
- **Frame Check Sequence (CRC-32)** for error detection

Ethernet Frame (IEEE 802.3-like layout)

FCS uses CRC-32 for error detection.
Minimum frame size helps collision detection in classic shared-media Ethernet.

Figure 3: Ethernet frame fields (visual proportions; not byte-accurate scale).

MAC Addressing

Each NIC has a *Media Access Control (MAC) address*, a 48-bit identifier. Traditionally, MAC addresses were globally unique (assigned by manufacturers), though virtualization and software-defined networking have made reuse more common.

In summary, the introduction to links establishes the foundation of networking: how raw bits are reliably sent across physical media and how nodes can identify and validate each other's transmissions.

2.2 Problems Solved by the Link Layer

The link layer provides critical services that make direct communication between adjacent nodes practical and reliable. Without these, higher layers would be unable to function over real-world physical media. The main problems addressed are:

Data Encoding & Addressing

Raw electrical, optical, or radio signals need to be mapped into a consistent bit representation. The link layer ensures that:

- **Encoding schemes** (e.g., NRZ, Manchester coding, 8b/10b) provide clock recovery and robustness against noise.
- **Framing** encapsulates packets into bounded units, with headers and trailers to mark boundaries.
- **Addressing** is handled via link-layer identifiers such as *Media Access Control (MAC) addresses* in Ethernet or Wi-Fi, allowing receivers to determine if a frame is meant for them, for a broadcast, or for a multicast group.

Error Detection & Correction

Physical channels are imperfect and bit flips occur due to noise or interference. The link layer therefore provides:

- **Error detection codes**, such as parity bits and Cyclic Redundancy Check (CRC), which allow receivers to identify corrupted frames.

- **Error correction codes** (less common at this layer), which enable recovery without retransmission by using redundancy (e.g., Hamming codes in wireless contexts).
- A policy for discarding or retransmitting bad frames, depending on the protocol.

While the Internet as a whole offers only "best-effort" service, these mechanisms prevent the majority of bit-level errors from propagating.

Shared Medium Access Coordination

When multiple nodes share the same physical medium (e.g., Wi-Fi, coaxial Ethernet), they must avoid collisions:

- **Channel partitioning** schemes such as TDMA (time slots) and FDMA (frequency bands).
- **Random access protocols**, e.g. ALOHA or CSMA (Carrier Sense Multiple Access), where nodes listen before transmitting.
- Collision handling: Ethernet's CSMA/CD (Collision Detection) aborts on collisions, while Wi-Fi's CSMA/CA (Collision Avoidance) uses acknowledgments and backoff timers.

This prevents chaos on a shared channel and ensures fair use of bandwidth.

Extending to LANs

Links are not limited to point-to-point connections:

- **Switches and bridges** interconnect multiple links, creating *Local Area Networks (LANs)* that scale to hundreds or thousands of hosts.
- The link layer supports **broadcast domains** (e.g., Ethernet broadcast address) and **multicast** delivery.
- Wireless Access Points act as link-layer relays, integrating mobile nodes into LANs.

These extensions allow a collection of physical links to behave as a coherent local network.

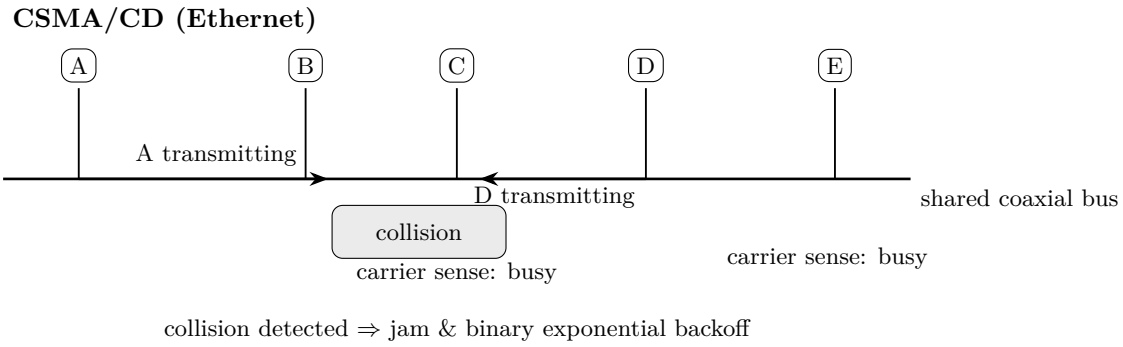


Figure 4: Ethernet on a shared bus using CSMA/CD: simultaneous transmissions lead to a collision, then jam & binary exponential backoff.

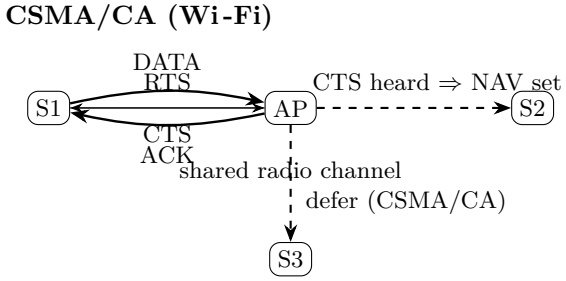


Figure 5: Wi-Fi using CSMA/CA: RTS→CTS reserves the channel; other stations set NAV and defer while DATA→ACK completes.

In summary, the link layer solves the fundamental challenges of *encoding, reliability, coordination, and scaling*, making it possible for higher layers (network, transport, and applications) to operate independently of the messy realities of the physical medium.

2.3 Framing and Ethernet

A fundamental task of the link layer is to transform a stream of raw bits into structured units called **frames**. Framing allows receivers to recognize where a transmission starts and ends, and provides the metadata needed for delivery.

Why Framing is Needed

- Without boundaries, a receiver cannot tell where one packet ends and another begins.
- Framing introduces **headers** and **trailers** around the data, allowing synchronization and error detection.
- Frames support **multiplexing**, since addresses in the header indicate which higher-level protocol should process the payload.

Ethernet as the Dominant Link-Layer Technology

Ethernet is the most widely used link-layer protocol in wired LANs. It defines both the physical signaling (e.g., twisted pair, fiber) and the frame format used on the link.

Ethernet Frame Structure An Ethernet frame has the following fields:

Destination MAC	Source MAC	EtherType/Length	Payload	FCS (CRC)
6 bytes	6 bytes	2 bytes	46–1500 bytes	4 bytes

- **Destination and Source MAC addresses** are 48-bit identifiers assigned to network interface cards (NICs).
- The **EtherType** field identifies the higher-level protocol encapsulated (e.g., IPv4, IPv6, ARP).
- The **Payload** contains the network-layer packet (commonly an IP datagram).
- The **Frame Check Sequence (FCS)** is a 32-bit CRC used for error detection.

Additional Fields Before the frame, Ethernet also uses:

- **Preamble** (7 bytes): a pattern of alternating 1s and 0s for clock synchronization.
- **Start Frame Delimiter (SFD)** (1 byte): marks the beginning of the frame.

Broadcast and Multicast

Ethernet supports special addressing:

- **Broadcast**: all bits set to 1 (FF:FF:FF:FF:FF:FF), delivered to all NICs.
- **Multicast**: frames delivered to a subset of hosts that join a multicast group.

Summary

Framing provides structure to data transmission, while Ethernet establishes a robust and simple format that has become the backbone of wired LANs. By defining clear headers, trailers, and addressing, Ethernet allows interoperability across vendors and higher-layer protocols.

2.4 Error Handling

Physical links are noisy: interference, attenuation, crosstalk, and sampling jitter all cause bit flips. The link layer therefore provides mechanisms to *detect* (and sometimes *correct*) errors so that corrupted frames don't propagate up the stack.

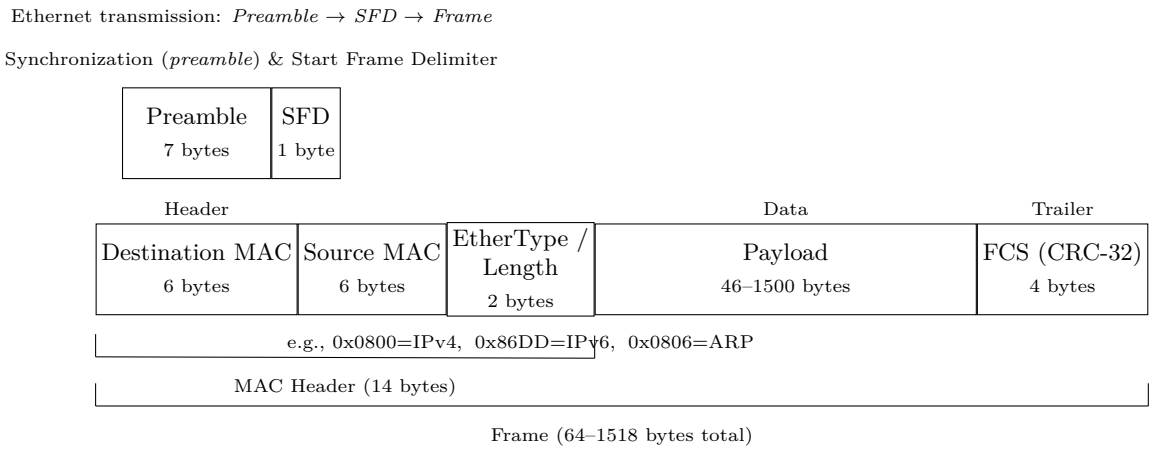


Figure 6: Ethernet framing: preamble & SFD for synchronization, then MAC header, payload, and CRC-32 trailer.

Goals and Design Choices

- **Detect corruption reliably** with minimal overhead and fast hardware.
- **Decide policy on bad frames**: drop vs. correct vs. retransmit.
- **Keep it stateless** on the wire where possible (simple, fast NIC logic).

Lightweight Detection: Parity

Parity adds a single bit so that the total number of 1s is even (even parity) or odd (odd parity). A single flipped bit flips the parity and is detected.

- Pros: tiny overhead, trivial to implement in hardware (chaining XORs).
- Cons: *only* guarantees detection of any *odd* number of bit errors; cannot localize or correct errors; weak against bursts.

Checksums vs. CRC

A **checksum** (e.g., ones'-complement Internet checksum) is simple and good at catching random noise, but is mathematically weaker than cyclic codes. Ethernet and many link protocols therefore use a **cyclic redundancy check (CRC)**.

- Treat the bitstring as coefficients of a polynomial over $\mathbb{F}_2$.
- Transmitter appends an r-bit *Frame Check Sequence (FCS)* so that the resulting polynomial is divisible by a public *generator* $G(x)$.
- Receiver redivides; nonzero remainder $\Rightarrow$ corruption.

Why CRCs are strong Choosing $G(x)$ with particular factors yields guarantees such as: detection of *all* single-bit errors, all double-bit errors at certain separations, all odd-weight errors, and all burst errors up to a bounded length (the bound depends on r). Practical links use $r \in \{16, 32\}$ for fast, strong detection.

Ethernet's CRC-32 (FCS)

Ethernet appends a 32-bit FCS (often described by generator polynomial $G(x)$ associated with 0x04C11DB7). In practice this achieves:

- Detection of **any** 1-bit error.
- Detection of **any** two adjacent 1-bit errors.

- Detection of **any** odd number of 1-bit errors.
- Detection of **any** burst error up to 32 bits.

Implementation is streaming and cheap in hardware via an LFSR (linear-feedback shift register), so the NIC can compute/verify the FCS at line rate as bytes arrive "off the wire."

Error *Correction* (FEC) at the Link

While many wired LANs detect-and-drop, some links use **forward error correction (FEC)** to *repair* certain errors without retransmission (helpful when RTTs are large or losses are frequent). Examples:

- **Block codes** (e.g., Hamming, Reed–Solomon) correct a bounded number of symbol errors per block.
- **Convolutional/LDPC codes** (often in wireless/optical) trade bandwidth for resilience; interleaving combats burst errors.

FEC improves effective BER at the cost of extra bits and decoding latency.

Retransmission Policies (ARQ) at the Link

Some link protocols pair detection with **ARQ**:

- Receiver **ACK**s good frames; lack of ACK $\Rightarrow$ sender **retransmits**.
- Exponential or randomized **backoff** reduces repeated collisions on shared media.
- Example: 802.11 (Wi-Fi) uses CSMA/CA with ACKs and backoff; classic shared-bus Ethernet used CSMA/CD with jam signals and backoff.

What Happens on a Bad Frame?

- **Drop**: most Ethernet NICs discard frames failing FCS.
- **Retry at link**: protocols with ARQ trigger a retransmission.
- **Retry end-to-end**: even if the link passes a frame, upper layers (e.g., TCP) may still detect loss/corruption and recover independently.

Rule of Thumb

Use parity or simple checksums for pedagogical purposes or ultra-low-cost embedded links; prefer CRCs (FCS) for serious LANs; add FEC where loss/RTT make retransmission expensive; layer ARQ when the link technology supports it and timing permits.

2.5 Medium Access Control (MAC)

When multiple nodes share the same physical medium, they must avoid talking over one another. The **Medium Access Control (MAC)** sublayer coordinates access to the link so that frames are delivered with minimal collisions, fair sharing, and efficient use of bandwidth.

Design Goals

- **Efficiency**: high channel utilization even under load.
- **Fairness**: no single host monopolizes access.
- **Low latency**: minimize waiting to transmit.
- **Robustness**: handle collisions and errors gracefully.

Channel Partitioning Approaches

Divide the channel into slices allocated to users.

- **Time Division Multiple Access (TDMA)**: nodes take turns in time slots.
- **Frequency Division Multiple Access (FDMA)**: nodes occupy distinct frequency bands.
- **Code Division Multiple Access (CDMA)**: nodes use orthogonal codes to transmit simultaneously.

Pros: predictable, collision-free. Cons: wasted slots if some users are idle; less flexible in bursty traffic.

Random Access Protocols

Nodes contend for the channel; collisions may occur.

- **ALOHA**: pure random transmission; simple but inefficient.
- **Slotted ALOHA**: reduces collisions by aligning transmissions to slots.
- **Carrier Sense Multiple Access (CSMA)**: node first listens; transmits only if idle.
- **CSMA/CD** (Collision Detection): Ethernet's original approach. If collision is sensed, abort, send jam signal, and back off.
- **CSMA/CA** (Collision Avoidance): used in Wi-Fi. Nodes wait random backoff, use ACKs, and sometimes RTS/CTS handshakes.

Pros: flexible and adaptive. Cons: wasted bandwidth in collisions and backoff.

Taking Turns: Polling and Tokens

An intermediate between partitioning and random access:

- **Polling**: master node invites each in turn (overhead, single point of failure).
- **Token passing**: a special frame (token) circulates; only holder may transmit. Example: Token Ring, FDDI.

Ethernet Evolution

- **Classic Ethernet (bus topology)** used CSMA/CD.
- **Modern switched Ethernet** removes collisions: each link is point-to-point to a switch, so MAC becomes mostly about addressing, not arbitration.

Wi-Fi MAC

IEEE 802.11 uses CSMA/CA with additional mechanisms:

- **ACKs**: ensure reliable delivery over noisy wireless channels.
- **RTS/CTS handshakes**: optional mechanism to combat the hidden-terminal problem.
- **Inter-frame spacing & exponential backoff**: coordinate fairness.

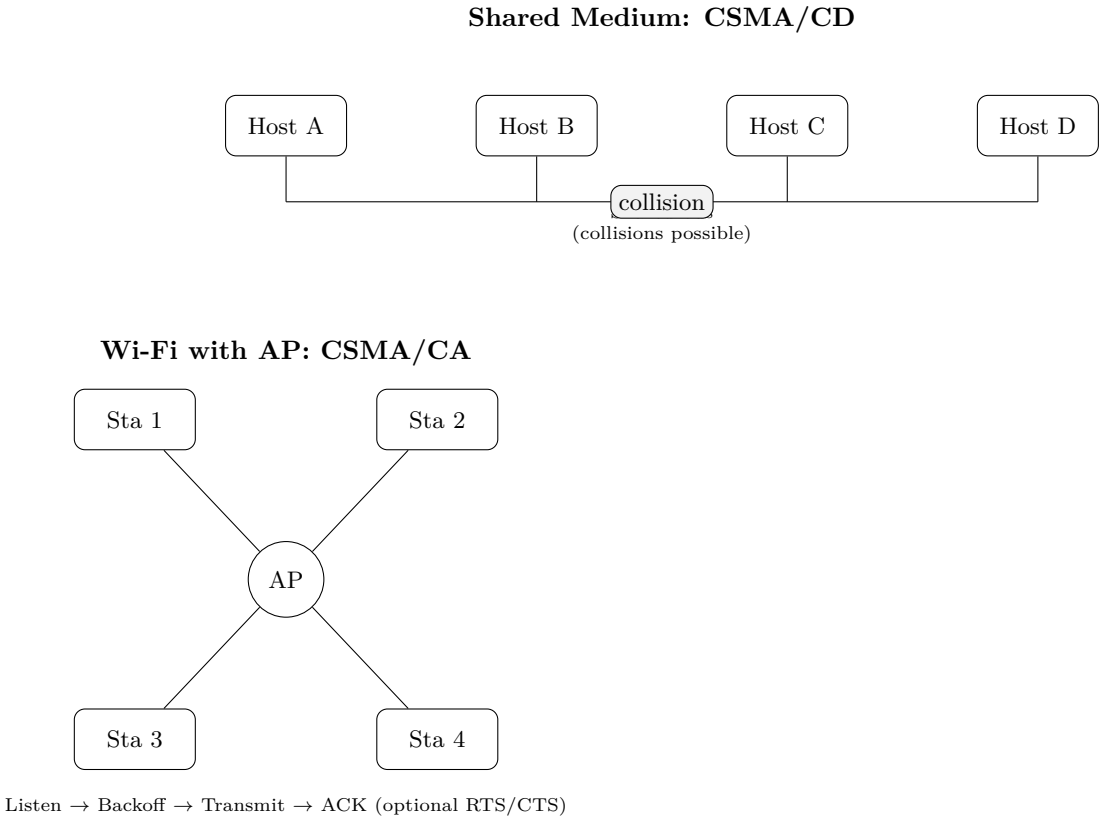


Figure 7: Comparison of wired shared bus (CSMA/CD) and Wi-Fi with an access point (CSMA/CA).

Summary

Medium Access Control protocols strike a balance between efficiency, fairness, and complexity. Ethernet historically relied on CSMA/CD, while modern wired networks eliminate collisions via switching. Wireless LANs still require careful coordination, so CSMA/CA and ACK-based recovery remain central to Wi-Fi MAC.

2.6 Local Area Networks

A **Local Area Network (LAN)** interconnects hosts over a limited geographic area (room, building, campus) with high bandwidth and low latency. In practice, two dominant link-layer realizations are (i) *switched Ethernet* for wired LANs and (ii) *802.11 Wi-Fi* with access points for wireless LANs.

Switched Ethernet (Wired LANs)

Modern Ethernet LANs use **switches** to interconnect point-to-point, full-duplex links. Each switch keeps a **MAC address table** that maps destination MAC → output port; on receipt of a frame, the switch looks up the destination and forwards out the indicated port.

Learning & Flooding Switches *learn* the source address of arriving frames and record the ingress port (self-populating the table). If the destination is *unknown*, the switch *floods* the frame out all ports (except the ingress); only the true destination accepts it. When the destination replies, the switch learns its location too, reducing future floods.

Broadcast & Loops LANs support broadcast and multicast; combined with flooding, this creates a *loop hazard* when multiple switches form a cycle (broadcast storms + duplicated frames). Mitigations include: track messages, bound hop counts, or disable loop-forming links via **Spanning Tree Protocol (STP)**; in Ethernet practice, STP (or designing loop-free topologies) is standard.

Collision vs Broadcast Domains With switching, each host–switch link is its own *collision domain* (full-duplex, no CSMA/CD), but the entire bridged LAN is a common *broadcast domain* unless segmented (e.g., via VLANs or routers). (Your slides emphasize the switch-based scaling beyond a single shared cable.)

Wireless LANs (802.11 Wi-Fi)

Wi-Fi LANs center on **access points (APs)** that connect wireless stations to the wired distribution network; all traffic transits the AP even when stations are in mutual radio range. A key control task is **association** of a station to a specific AP.

MAC Behavior (CSMA/CA) Because stations cannot detect collisions while transmitting and may suffer hidden terminals, Wi-Fi relies on *carrier sense multiple access with collision avoidance* (listen → randomized backoff → transmit → ACK; optional RTS/CTS).

Putting It Together

- **Switch learning** + selective forwarding gives wired LANs high throughput, limiting floods to cold misses and broadcasts.
- **STP or loop-free design** prevents L2 broadcast storms in multi-switch topologies.
- **Wi-Fi APs** extend the LAN wirelessly; association and CSMA/CA manage who speaks when over the shared radio medium.

2.7 Quiz: Link Layer Review

Link Layer Quiz

1. Which of these is a valid Ethernet MAC address (select all that apply)?

- 10.10.2.10 — *No – IPv4 address*
- 2001:db8:85a3:8d3:1319:8a2e:370:7348 — *No – IPv6 address*
- **20:f1:9e:bd:3b:18** — *Yes – valid unicast MAC*
- **FF:FF:FF:FF:FF:FF** — *Yes – Ethernet broadcast*

2. Which of these is *not* a field in the Ethernet header?

- Destination (dst) — *Valid*
- Type (EtherType) — *Valid*
- Source (src) — *Valid*
- **Quality of Service (QoS)** — *Not in the basic Ethernet header*

3. For even parity, what is the parity bit for this data: `0101 0001`?
The data has three ones (odd). To make the total count even, the parity bit must be **1**.

4. Indicate which of the following is a characteristic of the Ethernet FCS field (select all that apply).

- **Can detect burst errors** — *CRC-32 detects up to 32-bit bursts*
- **Can detect a single bit error** — *Always*
- **Can be calculated as data arrives off the wire** — *Streaming LFSR*
- Is used to determine vendor of the network card — *No – that's from MAC OUI*
- Can fix any single bit error — *No – detection, not correction*

5. Which is a negative property of channel partitioning (e.g., TDMA/FDMA)?

- A sender can become blocked — *Not inherent*
- **The channel isn't efficiently used unless all nodes have data to send**

- Collisions occur frequently — *No – partitioning avoids collisions*
- It requires complex coordination with an arbiter — *Not necessarily*

6. In CSMA/CD, how does a node detect whether a collision occurred?
It monitors (reads) the channel while transmitting and compares with what it sent. A mismatch ⇒ collision.

7. In CSMA/CA, how does a node detect whether a collision occurred?
The receiver sends an ACK if successful. Lack of an ACK implies a collision (or error).

8. Given a switch with MAC table (20:30:40:50:60:70 → port 0), if it receives a frame with destination FF:FF:FF:FF:FF:FF, what will it do?
Broadcast: forward out all ports (typically all except the ingress).

9. Same switch; destination 20:30:40:50:60:70 — what will it do?
Known unicast: forward out port 0.

10. Same switch; destination 20:30:40:22:AA:22 — what will it do?
Unknown unicast: flood out all ports (typically all except the ingress).

3 The TCP/IP Model and Architecture

3.1 Historical Context and Design Philosophy

The origins of the TCP/IP protocol suite are deeply rooted in the evolution of digital communication and packet-switching theory during the 1960s and 1970s. Early computer networks, such as the ARPANET, were conceived as experiments in interconnecting heterogeneous systems for research collaboration. The United States Department of Defense's Advanced Research Projects Agency (ARPA) sought to design a communication system that could remain functional even in the event of partial network failure, motivating a distributed and resilient design philosophy. The result was a radical departure from circuit-switched communication systems, such as the telephone network, toward a packet-switched paradigm that emphasized decentralization and flexibility.

From ARPANET to the Internet. By the early 1970s, the ARPANET had demonstrated the viability of packet switching. Researchers such as Vinton Cerf and Robert Kahn formalized the concept of *internetworking*, the idea that distinct networks could interoperate through a shared protocol layer rather than a shared physical infrastructure. Their 1974 paper, *A Protocol for Packet Network Intercommunication*, introduced the fundamental principles that became TCP/IP: layering, reliability through end-to-end control, and the use of gateways (now routers) to interconnect networks without requiring them to share a common internal structure.

The term "Internet" itself derives from this notion of an "internetwork," a network of networks. Each network could operate independently, with TCP/IP providing the universal glue binding them together. By 1983, the TCP/IP protocols were adopted as the standard for ARPANET, marking the birth of the modern Internet.

Philosophy of Design. As noted by Clark (1988) and summarized in Stevens' text, the Internet's architecture was guided by several explicit goals:

- Resilience: communication should continue despite the loss of networks or routers.
- Heterogeneity: the system must support diverse network technologies.
- Scalability: the architecture should permit distributed, decentralized management.
- Simplicity and low cost: host attachment and protocol implementation should be easy.
- Accountability: resources must be traceable and measurable across the network.

These principles collectively emphasized robustness over efficiency and simplicity over central control. The network's intelligence was placed at the edges—within the hosts—rather than within the intermediary routers. This design approach, known as the **end-to-end principle**, remains one of the foundational philosophies of the Internet's architecture.

Evolution of the TCP/IP Model. While other protocol architectures existed—most notably the ISO's seven-layer OSI model—the TCP/IP suite distinguished itself by its pragmatism and early deployment. TCP/IP's layering (Link, Internet, Transport, and Application) was designed to be implementable, not idealized. As Stevens notes, TCP/IP "evolved through implementation rather than design," meaning that protocol behavior was often shaped by empirical results from live networks rather than by theoretical abstraction. This evolutionary approach led to a highly adaptive system capable of integrating new technologies such as Ethernet, Wi-Fi, and optical fiber with minimal redesign.

Open and Cooperative Development. A final defining feature of TCP/IP's design philosophy was its openness. Specifications were published as freely accessible *Requests for Comments* (RFCs), enabling a global community of engineers to contribute to and critique protocol designs. This open model of standardization—supported by institutions such as the Internet Engineering Task Force (IETF)—allowed TCP/IP to evolve faster than proprietary alternatives and to become the dominant global networking standard.

Legacy. The design decisions of TCP/IP—particularly its simplicity, layering, and resilience—have allowed it to scale from its academic origins to a planetary-scale infrastructure. As Vinton Cerf observed in his foreword to *TCP/IP Illustrated*, the Internet's "warts and all" design remains a triumph of engineering adaptability, having grown by factors of a million in size and complexity without collapsing under its own weight.

In short, TCP/IP's history reflects a delicate balance between theory and pragmatism. Its design philosophy privileges robustness and openness over rigid control, making it not just a protocol suite but an enduring model of decentralized engineering.

3.2 Comparison to the OSI Model

The ISO Open Systems Interconnection (OSI) model defines seven conceptual layers, whereas the deployed Internet stack (TCP/IP) uses a pragmatic four-layer view: *Application, Transport, Internet, Link*. TCP/IP emphasizes implementability and encapsulation boundaries that reflected early deployments, rather than strict theoretical separation.

Layer Mapping (OSI → TCP/IP)

OSI	Name	TCP/IP layer (typical placement)
7	Application	**Application**
6	Presentation	**Application**
5	Session	**Application**
4	Transport	**Transport** (TCP, UDP)
3	Network	**Internet** (IP, ICMP, ARP/NDP[1])
2	Data Link	**Link** (Ethernet, 802.11, PPP)
1	Physical	**Link** (Ethernet, 802.11, PPP)

Why TCP/IP Merges Layers.

- **Pragmatic deployment history.** The Internet stack evolved through running code and interop tests, prioritizing the boundaries that mattered for implementation and encapsulation (ports, IP endpoints, link framing).
- **End-to-end focus.** Reliability and most application semantics live at the edges; routers provide best-effort forwarding. This favors a lean set of network-facing layers (Link/Internet) and rich edge layers (Transport/Application).
- **Presentation & Session folded into Applications.** Data representation, encryption, and conversation management are handled by libraries and application protocols (e.g., TLS in-app; HTTP/2 streams), so TCP/IP treats them as part of the *Application* layer rather than separate layers.
- **Data Link + Physical combined.** Implementations expose a single interface (NIC/driver) that provides framed service to IP; physical details (voltages, RF) are hidden under the *Link* abstraction (e.g., Ethernet, 802.11, PPP).
- **Clear demux points.** TCP/UDP ports (Transport), IP addresses/protocol numbers (Internet), and EtherType/802.11 frame types (Link) are the practical multiplexing/demultiplexing boundaries that code relies on.

Notes on Ambiguous Mappings. Some widely used mechanisms straddle boundaries:

- **TLS/DTLS.** Often taught as "presentation/security," but operationally runs in applications or atop UDP (DTLS), reinforcing TCP/IP's folded Application layer.
- **ICMP/ICMPv6.** Control messaging for IP (errors, diagnostics) is considered part of the Internet layer rather than an "extra" OSI layer.
- **ARP / Neighbor Discovery.** Bridge addressing and IP addressing via resolution live at the Link↔Internet boundary (integrated into the stack rather than a separate OSI layer).

[1] ARP operates between the Link and Internet layers in IPv4; Neighbor Discovery serves a similar role in IPv6.

Encapsulation in Practice. TCP/IP layering is enforced by encapsulation: Application data → Transport segment (TCP/UDP) → IP datagram (Internet) → Link frame. Each boundary provides a demultiplexing handle (port, protocol number, EtherType) and hides lower-level details, aligning with the implementation-focused treatment of layering in practice.

3.3 Encapsulation and Protocol Stack Operation

A central idea of the TCP/IP architecture is that communication occurs through a series of **encapsulated layers**. Each layer provides a specific service to the layer above it and relies on the layer below it for data transport. When an application sends data, each lower layer adds its own header information, forming a protocol data unit (PDU) appropriate to that layer. At the receiver, this process is reversed (*decapsulation*), with each layer removing its header and processing only the information relevant to its function.

Concept. The encapsulation process can be summarized as follows:

- The **Application layer** generates the message (e.g., an HTTP request).
- The **Transport layer** (TCP or UDP) adds a header containing source and destination ports, sequence numbers, and reliability information—creating a *segment*.
- The **Internet layer** (IP) adds its own header containing source and destination IP addresses—creating a *datagram*.
- The **Link layer** encapsulates the datagram within a frame, adding physical addresses (MAC addresses) and an error-detection field—creating a *frame*.

This nested structure allows each layer to operate independently. For example, TCP does not need to know whether the underlying link is Ethernet or Wi-Fi; it only needs a reliable mechanism for delivering bytes between hosts identified by IP addresses.

Decapsulation. At the destination, the process reverses:

1. The Link layer checks the frame's integrity (using CRC) and passes the payload up.
2. The Internet layer examines the IP header to determine routing and passes the segment upward.
3. The Transport layer verifies port numbers, sequence numbers, and reassembles the byte stream.
4. The Application layer receives the original message.

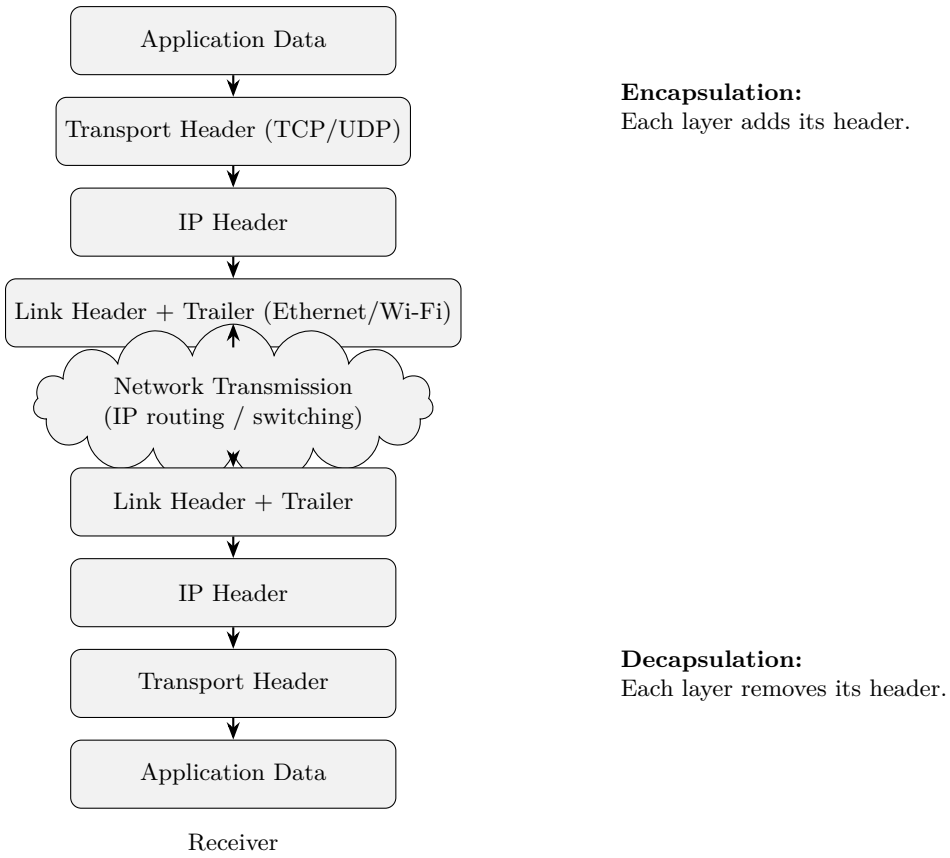


Figure 8: Encapsulation at the sender, transmission across the medium (cloud), and decapsulation at the receiver. The cloud intentionally sits between the two Link layers.

Key Observation. Encapsulation is what enables modularity: each layer only needs to understand its own headers and services, not the internals of other layers. This principle is what allows protocols like HTTP to function identically whether they ride over Ethernet, Wi-Fi, or cellular links.

3.4 Common Protocols by Layer

Each layer of the TCP/IP suite has a well-defined set of protocols that collectively enable end-to-end communication. The following overview lists the most common and historically important examples, emphasizing their primary roles and interactions across the stack.

Layer	Representative Protocols	Primary Functions
Application	HTTP, HTTPS, FTP, SMTP, POP3, IMAP, DNS, DHCP, SNMP, NTP, SSH, Telnet	Defines the semantics of networked applications: file transfer, web browsing, email, remote login, name resolution, configuration, and time synchronization. Operates over transport-layer sockets identified by port numbers (e.g., 80 = HTTP, 25 = SMTP, 53 = DNS).
Transport	TCP, UDP, SCTP, QUIC	Provides end-to-end data delivery and multiplexing. TCP offers reliable, connection-oriented transfer; UDP offers connectionless best-effort service; SCTP supports multistreaming for signaling; QUIC (built atop UDP) integrates TLS and congestion control for modern web transport.
Internet	IPv4, IPv6, ICMPv4, ICMPv6, IGMP, MLD	Handles addressing and routing across networks. IP provides best-effort packet delivery; ICMP reports errors and diagnostics (used by `ping`, `traceroute`); IGMP (IPv4) and MLD (IPv6) manage multicast group memberships.
Link	Ethernet (IEEE 802.3), Wi-Fi (IEEE 802.11), PPP, ARP, VLAN (802.1Q)	Manages framing, addressing, and access to the physical medium. ARP (Address Resolution Protocol) maps IP addresses to hardware addresses; PPP supports point-to-point serial links; Ethernet and Wi-Fi define MAC addressing and medium access control.

Layer Interdependence.

- **Encapsulation:** Each higher-layer PDU is wrapped within a lower-layer header (e.g., TCP segment → IP datagram → Ethernet frame).
- **Identifiers:** The Link layer uses MAC addresses, the Internet layer uses IP addresses, and the Transport layer uses port numbers.
- **Adjunct protocols:** Some protocols (e.g., ICMP, ARP) sit "between" canonical layers and are indispensable for diagnostics and address resolution.
- **Security extensions:** Protocols such as TLS, IPsec, and SSH overlay or augment existing layers to provide encryption, integrity, and authentication.

Example Stack in Action. A typical web request encapsulates these layers:

- **Application:** HTTP over TLS (HTTPS)
- **Transport:** TCP port 443
- **Internet:** IPv6 datagram (Next Header = TCP)
- **Link:** Ethernet II frame (Type = 0x86DD for IPv6)

Each layer performs its task independently, ensuring modularity and interoperability across the global Internet.

3.5 Example: Packet Journey Through the Stack

The interaction between the layers becomes clearer when viewed in the context of an everyday network operation such as retrieving a web page. The following sequence traces a simplified HTTPS (HTTP over TCP/IP) request from a client to a server, showing how data moves downward through the sender's stack and upward through the receiver's stack.

Scenario. A user opens a browser and visits `https://example.com`. The local machine acts as the client, while the remote web server hosts the resource. The journey proceeds as follows:

1. **Application Layer.** The browser (application) generates an HTTP `GET` request message, asking the server for a specific resource (e.g., `/index.html`). The message may be encrypted using TLS if the URL begins with `https://`.
2. **Transport Layer.** TCP (port 443 for HTTPS) establishes a reliable connection with the server's transport layer using the three-way handshake (`SYN`, `SYN-ACK`, `ACK`). The HTTP request is segmented and encapsulated with TCP headers (sequence and acknowledgment numbers, port numbers, etc.).
3. **Internet Layer.** The TCP segment is passed to IP, which adds its own header including source and destination IP addresses. The IP layer determines the next hop toward the server—possibly through multiple routers.
4. **Link Layer.** The IP packet is encapsulated in a link-layer frame (Ethernet, Wi-Fi, etc.). A destination MAC address is resolved via ARP if necessary, and the frame is transmitted over the physical medium.
5. **Across the Network.** Routers examine only the IP header and forward the packet toward the destination, potentially over several hops. Each router decapsulates the link header/trailer, makes a routing decision, and re-encapsulates the packet in a new link frame for the next hop.
6. **Arrival at the Server.** The server's NIC receives the frame and passes the payload up the stack. Each layer removes its respective header—Link → IP → TCP—and eventually the application layer (web server) receives the raw HTTP request.
7. **Response.** The process reverses for the HTTP response: the server's application constructs a response (e.g., a web page), encapsulates it through the same layers, and transmits it back to the client.

Round Trip. Upon receiving the response, the client decapsulates each layer in reverse order, reconstructing the original HTTP reply, which the browser then renders for the user.

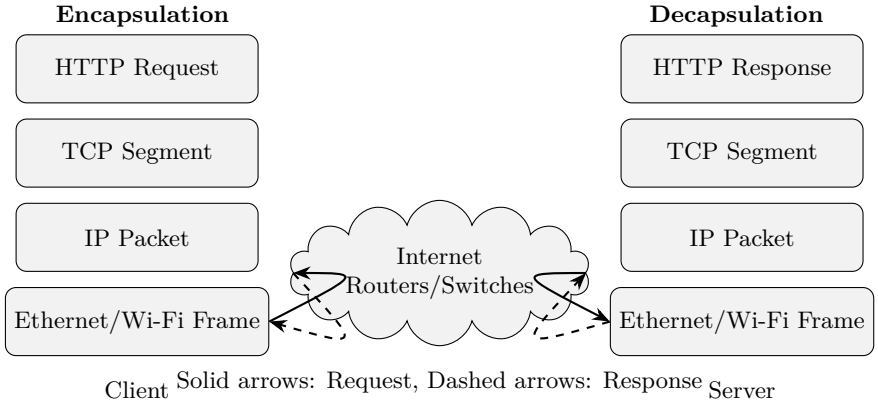


Figure 9: Data flow in an HTTPS web request: the client's application data passes down through TCP/IP layers, across the network, and back up the server's stack in reverse.

Summary. This process illustrates the modular, layered operation of TCP/IP:

- Each layer contributes specific control information by adding its own header.
- The network forwards packets based solely on IP-level information.
- Each endpoint independently handles its own encapsulation and decapsulation.

Together, these operations form the core of all Internet communication, regardless of application or medium.

4 Network Layer (Internet Layer)

4.1 Motivation for the Network (Internet) Layer

The **Network Layer**—often called the *Internet Layer* in the TCP/IP model—addresses the fundamental problem of interconnecting multiple, heterogeneous networks into a unified system of global communication. While the Link Layer enables communication between directly connected devices (for example, within a single Ethernet or Wi-Fi segment), most communication requires reaching hosts that are *not* on the same local network. The Network Layer provides the abstraction necessary to deliver data across arbitrary physical boundaries.

The Problem: Beyond Local Connectivity. Link-layer technologies such as Ethernet and Wi-Fi provide only *local* delivery; their addressing schemes (e.g., MAC addresses) are flat and limited to a single broadcast domain. If every network were isolated, two hosts on different LANs could never communicate directly. A mechanism was needed to:

- Uniquely identify each host globally.
- Route packets across multiple interconnected networks.
- Abstract away differences among link technologies (Ethernet, optical fiber, wireless, etc.).

The Network Layer provides this functionality by defining a common logical addressing and routing system—**the Internet Protocol (IP)**—that enables universal reachability.

Core Functions Introduced.

- **Logical addressing:** Each host is assigned an IP address that identifies its location in the global network topology, independent of the physical medium.
- **Routing:** Routers determine the best path to deliver packets between distant networks, forwarding them hop by hop toward the destination.
- **Fragmentation and reassembly:** IP allows large datagrams to traverse networks with smaller maximum transmission units (MTUs) by fragmenting them as needed.
- **Error reporting and diagnostics:** Companion protocols such as ICMP report unreachable destinations and timing information, enabling operational visibility.

Decoupling Applications from Infrastructure. One of the key motivations for introducing a network layer was to *decouple higher-layer protocols from the underlying hardware.* Applications like HTTP, SMTP, or DNS should not need to care whether the physical network is Ethernet, Wi-Fi, or a satellite link. By abstracting communication into logical endpoints (IP addresses), the Network Layer allows these applications to function seamlessly across any medium.

The "Internetworking" Concept. The term *Internet* itself comes from "inter-networking"—connecting multiple independent networks through a common addressing and forwarding scheme. This architecture supports scalability and heterogeneity: organizations can deploy their own local technologies internally but still exchange data using IP as the lingua franca of network connectivity.

End-to-End Perspective. At the Network Layer, the unit of transfer is the *datagram.* Each datagram is independent and contains enough information (source and destination addresses) to be delivered without reference to previous packets. This **connectionless** approach allows for flexibility, fault tolerance, and distributed routing decisions—key to the Internet's resilience.

In Summary.

- The Link Layer enables local communication.
- The Network Layer enables global communication.
- IP serves as the universal protocol binding heterogeneous systems together.

Without the Network Layer, the Internet would devolve into isolated local segments with no scalable way to reach remote destinations.

4.2 IPv4 Header Structure and Addressing

The Internet Protocol version 4 (IPv4) is the foundational packet format that defines how data is routed across the Internet. It provides the structure and semantics necessary for addressing, fragmentation, and forwarding.

Role of the IPv4 Header. Each IP datagram begins with a fixed header that contains information used by routers and hosts to forward, deliver, and process packets. Unlike the Link Layer, which operates within a single local network, IP is responsible for end-to-end delivery across potentially many heterogeneous networks.

The header conveys information such as the datagram's total length, source and destination IP addresses, fragmentation instructions, and time-to-live (TTL) limits that prevent indefinite circulation of lost packets.

IPv4 Header Fields. The IPv4 header is at least 20 bytes long (without options). Its fields are summarized below:

- **Version (4 bits):** Identifies the IP protocol version; for IPv4, this value is 4.
- **IHL (Internet Header Length, 4 bits):** Indicates the header length in 32-bit words (minimum = 5).
- **Type of Service (ToS) / DSCP:** Used for quality of service and prioritization.
- **Total Length:** Specifies the entire datagram size in bytes (header + data).
- **Identification, Flags, Fragment Offset:** Used for fragmentation and reassembly when packets exceed the MTU of a link.
- **Time to Live (TTL):** Limits the datagram's lifetime by counting down hops.
- **Protocol:** Indicates the encapsulated transport protocol (e.g., 6 for TCP, 17 for UDP).
- **Header Checksum:** Ensures the integrity of the IPv4 header only.
- **Source Address:** The 32-bit IP address of the sender.
- **Destination Address:** The 32-bit IP address of the receiver.
- **Options (optional):** Allows for additional routing, debugging, or security parameters.

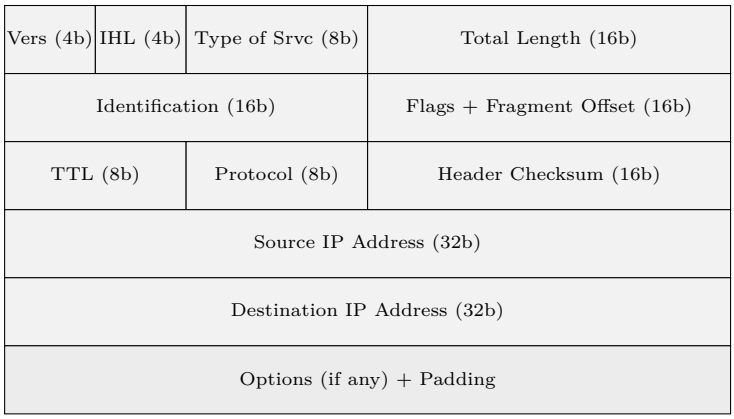


Figure 10: IPv4 header structure aligned to 32-bit boundaries (minimum 20 bytes; Options row present only when IHL > 5).

Addressing. An IPv4 address is a 32-bit number typically written in dotted decimal notation (e.g., 192.168.1.10). Each address is divided into:

- A **network portion** that identifies the subnet.
- A **host portion** that identifies the specific device within that subnet.

Routers use the network portion to forward packets toward the correct destination network, while the host portion is resolved locally (e.g., via ARP) once the packet reaches the destination subnet.

Classful and Classless Addressing. Historically, IPv4 addresses were divided into classes (A, B, C) with fixed boundaries between network and host bits. This scheme was later replaced by **Classless Inter-Domain Routing (CIDR)**, which allows variable-length subnet masks (VLSM) for more efficient allocation of address space. For example:

$$192.168.1.0/24 \rightarrow \text{Network address with 24 bits of prefix, 8 bits for hosts.}$$

In Summary. The IPv4 header provides all the information required to deliver datagrams across interconnected networks:

- Logical addressing ensures global identification of endpoints.
- Fragmentation allows adaptation to different link MTUs.
- TTL and checksum mechanisms ensure robustness and integrity.

Despite its simplicity, IPv4 remains one of the most enduring and widely deployed protocol formats in the history of networking.

4.3 IPv6: Motivation and Design

IPv6 was designed to overcome fundamental limitations of IPv4 while simplifying forwarding in the core of the network.

Why IPv6? (Contrast with IPv4)

- **Address space:** 128-bit addresses ($\approx 3.4 \times 10^{38}$) remove scarcity and NAT pressure; supports hierarchical aggregation at Internet scale.
- **Simpler core:** No header checksum and no in-network fragmentation (routers never fragment) make forwarding faster and easier to implement.
- **Clean extensibility:** Options are moved out of the base header into a *chain of extension headers*, so the fixed header stays small and fast to parse.
- **Built-in neighbor functions:** ARP is replaced by *Neighbor Discovery (ND)* over ICMPv6; multicast replaces broadcast.

IPv6 Base Header (fixed 40 bytes). Fields and semantics:

- **Version (4b):** Always 6 for IPv6.
- **Traffic Class (8b):** QoS/DSCP semantics similar to IPv4 ToS.
- **Flow Label (20b):** Marks packets that belong to the same flow so routers can apply consistent treatment (e.g., ECMP hashing, QoS). Endpoints set it; routers keep it intact.
- **Payload Length (16b):** Bytes following the base header (i.e., extension headers + upper-layer data).
- **Next Header (8b):** Identifies the next header type (another extension header or a transport header like TCP=6/UDP=17).
- **Hop Limit (8b):** Like IPv4 TTL—decremented each hop to prevent loops.
- **Source Address (128b), Destination Address (128b).**

Extension Headers. Instead of a single, variable "Options" area (as in IPv4), IPv6 uses a *linked list* of headers, each indicated by the previous header's *Next Header* value. Common types:

- **Hop-by-Hop Options** (processed by every router—use sparingly).
- **Routing** (source routing information).
- **Fragment** (used by *end hosts* only—routers do not fragment).
- **Destination Options** (processed only at the destination).
- **AH/ESP** (IPsec authentication and encryption).

Autoconfiguration (SLAAC). IPv6 supports *Stateless Address Autoconfiguration*: a host

1. forms a **link-local** address (prefix `fe80::/10`) from its interface identifier;
2. uses **Neighbor Discovery** (ICMPv6) for Duplicate Address Detection and router discovery;

3. learns prefixes from Router Advertisements, then derives **global unicast** addresses (optionally with privacy/temporary IDs).

DHCPv6 can supply additional configuration (DNS, etc.), but basic addressing works without a stateful server.

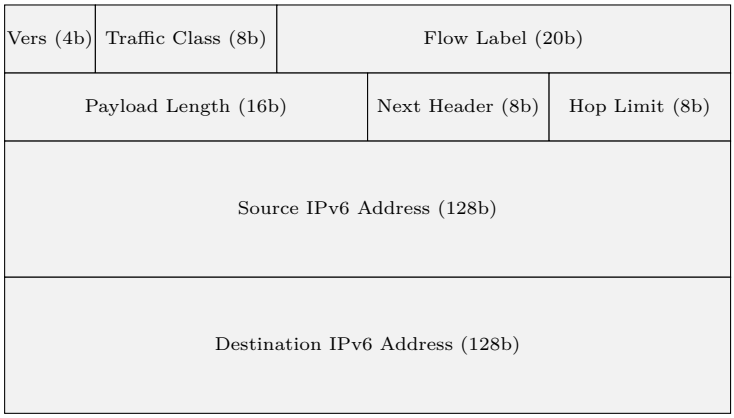


Figure 11: IPv6 base header (fixed 40 bytes). The 128-bit Source and Destination addresses are shown as compact two-row blocks; options live in separate *extension headers*.

Figure 12: Extension headers form a chain: each header's *Next Header* points to the next element (another EH or the transport header).

Key Takeaways.

- IPv6 keeps the *base header fixed and lean*; optional features live in *extension headers*.
- **Flow Label** enables per-flow treatment without deep inspection.
- **No router fragmentation** (endpoints use PMTUD); **no header checksum**.
- **SLAAC + ND** provide robust, server-less address configuration and neighbor resolution.

4.4 Fragmentation and MTU Discovery

In practice, every physical network technology has a maximum transmission unit (**MTU**)—the largest frame size it can carry. When an IP datagram exceeds this size, it must either be split into smaller pieces (*fragmentation*) or retransmitted in a smaller size after learning the path's MTU (*Path MTU Discovery*).

The Need for Fragmentation. An IPv4 datagram may traverse links with differing MTUs (e.g., 1500 bytes on Ethernet, 576 bytes on PPP). If the packet exceeds the next hop's MTU, routers can fragment it into multiple pieces, each carrying a fragment of the payload plus its own IP header. Each fragment is identified by:

- the same **Identification** number (to match fragments),
- the **Fragment Offset** (location within the original datagram, in 8-byte units),
- and the **More Fragments (MF)** flag, set to 1 in all but the last fragment.

Reassembly occurs only at the destination host, never at routers, since fragments may take different paths and arrive out of order:contentReference[oaicite:2]index=2.

Drawbacks of Fragmentation. While functionally correct, fragmentation introduces inefficiency:

- Each fragment adds a 20-byte IP header, increasing overhead.
- Loss of any single fragment forces retransmission of the entire datagram.

- Firewalls and NAT devices often handle fragments poorly because only the first fragment contains transport-layer ports.

Therefore, modern hosts strive to *avoid* fragmentation altogether by discovering the path MTU.

Path MTU Discovery (PMTUD). PMTUD determines the smallest MTU along the route between source and destination. The sender transmits packets with the IPv4 "Don't Fragment" (DF) flag set. If a router encounters a packet larger than its outgoing interface's MTU, it discards the packet and sends an **ICMP "Fragmentation Needed"** (Type 3, Code 4) message containing the next-hop MTU. The sender reduces its packet size accordingly, caching the new MTU value for that destination. Per RFC 1191, the cached MTU is considered stale after roughly 10 minutes and must be rediscovered:contentReference[oaicite:3]index=3:contentReference[oaicite:4]index=4.

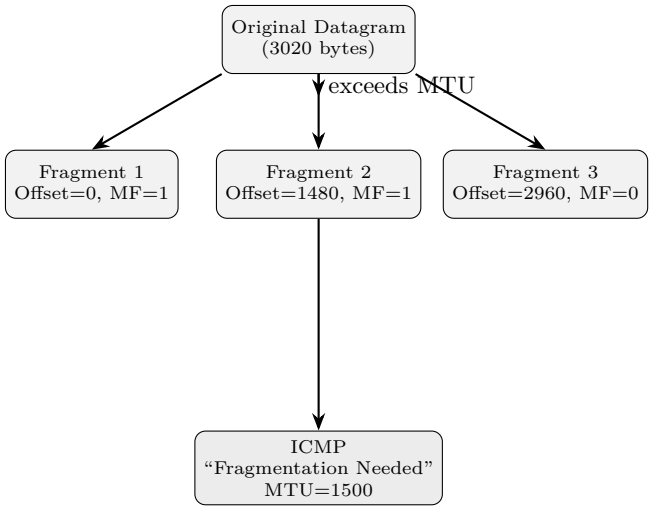
PMTUD with TCP and UDP. For TCP, PMTUD interacts with the segment size selection process:

- TCP begins with the smaller of the interface MTU or the peer's announced MSS.
- If a "Packet Too Big" message arrives, TCP reduces the segment size (usually MTU minus header overhead) and retransmits.
- The process repeats until no further ICMP errors are received.

Because UDP applications often specify datagram size manually, PMTUD is less adaptive unless implemented by the IP layer itself.

IPv6 and Fragmentation. In IPv6, routers *never* fragment packets. If a packet is too large, the router sends an **ICMPv6 "Packet Too Big" (PTB)** message to the source, which must fragment at the host level using the dedicated *Fragment Extension Header*. This simplification reduces router load and improves forwarding efficiency.

Alternative Approach: PLPMTUD. The **Packetization Layer Path MTU Discovery** (PLPMTUD, RFC 4821) avoids reliance on ICMP messages, which may be filtered by firewalls. Instead, it dynamically probes path capacity at the transport layer (e.g., TCP) by adjusting segment size and observing whether transmissions are acknowledged.



Router returns ICMP message → sender lowers MTU

Figure 13: IPv4 fragmentation example and ICMP feedback during Path MTU Discovery.

Summary.

- IPv4 allows fragmentation by routers; IPv6 limits it to the source.
- PMTUD prevents fragmentation by discovering the smallest MTU along the route.
- ICMP messages (or PLPMTUD probes) inform senders of path constraints.
- Modern stacks cache MTU data per destination and age it periodically.

4.5 ICMP and Error Reporting

The **Internet Control Message Protocol (ICMP)** operates as an adjunct to IP, providing feedback about network delivery issues and limited diagnostic capability. While IP is designed as a best-effort, connectionless service, ICMP provides the minimal signaling required to make it usable and debuggable.

Purpose. ICMP does not transport user data—it carries control and error messages about the delivery of IP datagrams. Its messages are encapsulated directly within IP packets (Protocol = 1 for ICMPv4, 58 for ICMPv6).

Typical roles:

- Reporting unreachable destinations, expired lifetimes, or other forwarding problems.
- Providing diagnostics via *Echo Request/Reply* (used by `ping`).
- Supporting network-layer mechanisms such as *Path MTU Discovery*.

Message Format. Each ICMP message consists of:

- **Type (8 bits)** — Identifies the message class (e.g., Echo Reply = 0, Echo Request = 8).
- **Code (8 bits)** — Provides subtype information (e.g., "port unreachable" vs "network unreachable").
- **Checksum (16 bits)** — Detects corruption in the ICMP header and data.
- **Rest of Header + Data** — Varies by message type.

ICMP messages are never sent in response to other ICMP messages, to avoid infinite loops.

Common Message Types.

1. **Echo Request / Echo Reply (Types 8 and 0).** Used for reachability testing (`ping`). The sender transmits an Echo Request; if the destination is reachable and operational, it replies with an Echo Reply containing the same identifier and sequence number.
2. **Destination Unreachable (Type 3).** Indicates that a packet could not be delivered. The *Code* field distinguishes causes:
   - Code 0 – Network unreachable
   - Code 1 – Host unreachable
   - Code 3 – Port unreachable (common for UDP)
   - Code 4 – Fragmentation needed and DF set (used in PMTUD)

   The original packet's IP header and first 8 bytes of data are returned for identification.
3. **Time Exceeded (Type 11).** Sent when a packet's Time-To-Live (TTL) or Hop Limit reaches zero before reaching its destination. This mechanism enables `traceroute`, which deliberately sends packets with incrementally small TTL values and records the ICMP responses from each hop.

Relationship to IPv6. ICMPv6 (Protocol = 58) merges the traditional ICMP functions with new mechanisms: Neighbor Discovery (replacing ARP), Router Advertisements, and Multicast Listener Discovery. Its structure is similar to ICMPv4 but with new types for ND, error, and informational messages.

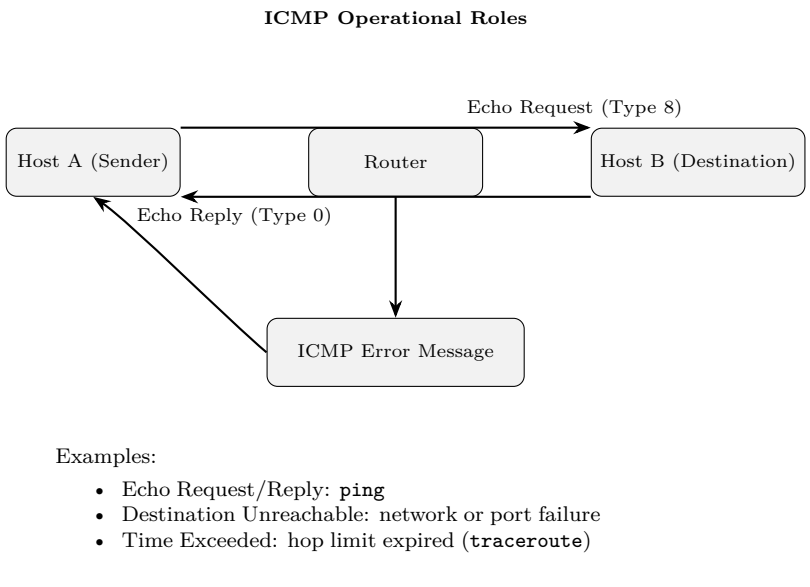


Figure 14: ICMP messages provide diagnostic and error reporting: Echo Request/Reply (reachability) and error signaling for unreachable or expired packets.

Summary.

- ICMP complements IP by reporting delivery problems and enabling diagnostics.
- Echo Request/Reply verify reachability and latency.
- Destination Unreachable signals routing or transport-layer errors.
- Time Exceeded enables path tracing and prevents infinite circulation.

Despite being a control protocol, ICMP is indispensable for debugging, dynamic routing feedback, and error recovery in the IP suite.

4.6 ARP and Neighbor Discovery Protocol (NDP)

IP addressing provides logical, network-level identifiers for hosts, but frames on a local network (e.g., Ethernet or Wi-Fi) must still be sent to **physical** (MAC) addresses. Thus, a mechanism is required to map IP addresses to hardware addresses at the Link Layer. In IPv4 this is handled by the **Address Resolution Protocol (ARP)**, while IPv6 replaces ARP with the more capable **Neighbor Discovery Protocol (NDP)**.

ARP (Address Resolution Protocol). ARP is a simple, broadcast-based protocol defined in RFC 826. Its goal is to resolve a target IPv4 address to a corresponding MAC address on the same LAN.

Operation:

1. When a host needs to send a packet to another IP on the same network, it first checks its *ARP cache*.
2. If no entry exists, it broadcasts an **ARP Request** frame: "Who has 192.168.1.20? Tell 192.168.1.10."
3. The target host replies with a unicast **ARP Reply**, containing its MAC address.
4. The sender stores the mapping in its ARP cache for future use.

ARP Table Example:

IPv4 Address	MAC Address
192.168.1.1	00:1A:2B:3C:4D:5E
192.168.1.20	08:00:27:AC:FE:91

Limitations of ARP.

- Broadcast traffic does not scale well in large networks.
- No authentication—vulnerable to ARP spoofing or poisoning.
- Separate protocols are needed for address autoconfiguration and neighbor reachability.

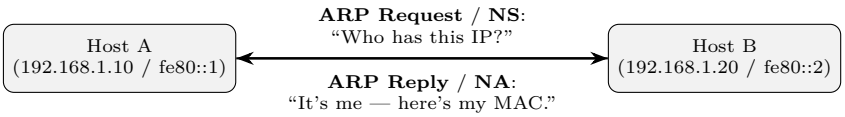
NDP (Neighbor Discovery Protocol). IPv6 eliminates ARP and uses the **Neighbor Discovery Protocol**, defined in RFC 4861, which operates using ICMPv6 messages sent to **multicast** (not broadcast) addresses. NDP combines several IPv4-era mechanisms—ARP, ICMP Router Discovery, and even DHCP functions—into one unified framework.

Core Message Types:

- **Neighbor Solicitation (NS):** "Who has this IPv6 address?"
- **Neighbor Advertisement (NA):** Response providing the corresponding MAC address.
- **Router Solicitation (RS):** Sent by hosts to discover local routers.
- **Router Advertisement (RA):** Sent periodically or on-demand by routers, providing prefix and configuration info (used for SLAAC).

Advantages of NDP over ARP.

- Uses **multicast** rather than broadcast, reducing unnecessary traffic.
- Integrates address autoconfiguration and router discovery.
- Includes **Neighbor Unreachability Detection** to verify that cached entries are still valid.
- Provides optional **Secure Neighbor Discovery (SEND)** for cryptographic protection against spoofing.



Example exchanges:

- IPv4: ARP Request → ARP Reply (broadcast → unicast)
- IPv6: Neighbor Solicitation → Neighbor Advertisement (multicast → unicast)
- Routers also send Router Advertisements for prefix discovery.

Figure 15: ARP (IPv4) and NDP (IPv6) both resolve network-layer addresses to link-layer identifiers. ARP uses broadcast; NDP uses ICMPv6 multicast messages.

Summary.

- ARP resolves IPv4 addresses to MAC addresses using broadcast.
- NDP performs the same role for IPv6 using ICMPv6 multicast.
- NDP integrates autoconfiguration and reachability testing.
- Both maintain a local cache of IP–MAC mappings to reduce overhead.

4.7 Router Data Plane

The **data plane** of a router is responsible for the actual forwarding of packets. While the **control plane** decides *where* packets should go (through routing protocols and table updates), the data plane handles the per-packet, high-speed operation of *moving* packets from an input interface to the appropriate output interface.

Core Responsibilities.

- **Packet reception:** Frames are received at a router's input interface, checked for integrity, and stripped of link-layer headers.
- **Header inspection:** The router examines the network-layer header (e.g., IPv4 or IPv6) to determine the destination address.
- **Forwarding decision:** Using the *Forwarding Information Base (FIB)*—a hardware-optimized subset of the routing table—the router identifies the best output interface and next-hop address.
- **Encapsulation and transmission:** The router decrements the packet's TTL/Hop Limit, recalculates the header checksum (IPv4), encapsulates the datagram in a new link-layer frame, and sends it out the correct interface.

Data Plane vs. Control Plane.

Control Plane: Runs slower, software-based logic responsible for computing routes (e.g., OSPF, BGP, RIP).

Data Plane: Runs at line rate in hardware or fast path ASICs, forwarding packets according to the FIB and access control lists (ACLs).

Forwarding Process. Routers typically implement the following steps for each incoming packet:

1. Verify and decapsulate the frame at the link layer.
2. Perform a lookup in the FIB using the packet's destination IP address.
3. Decrement the TTL/Hop Limit and update checksums if necessary.
4. Apply policy filters or QoS classification (optional).
5. Re-encapsulate the packet into a new frame and transmit via the chosen output interface.

Performance Considerations.

- **Hardware acceleration:** Modern routers implement the data plane in ASICs or network processors for line-rate forwarding.
- **Parallel pipelines:** Separate ingress and egress pipelines allow for simultaneous packet handling.
- **Queue management:** Packets waiting for output interfaces are placed in buffers; algorithms such as **Weighted Fair Queuing (WFQ)** or **RED (Random Early Detection)** control congestion.

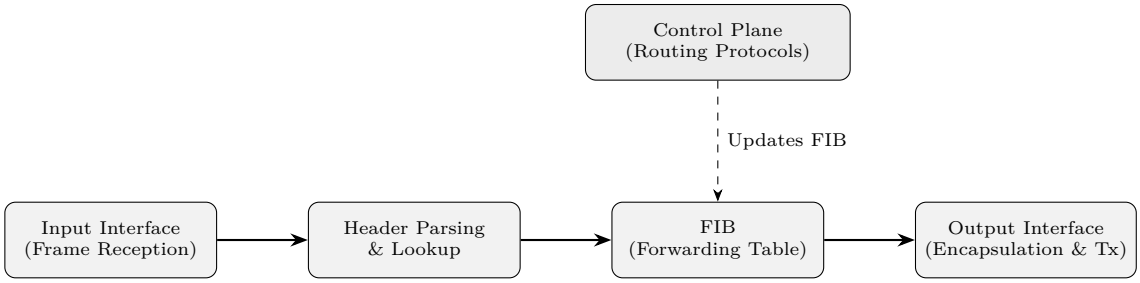


Figure 16: Simplified router data-plane pipeline: incoming frames are parsed, the FIB is consulted, and packets are forwarded to the proper output interface.

Key Takeaways.

- The data plane implements the router's forwarding logic at high speed, based on precomputed tables from the control plane.
- IPv4 requires TTL decrement and checksum recomputation at each hop; IPv6 omits the checksum for efficiency.
- Packet queuing and QoS policies are part of the egress data-plane operation.

4.8 Routing Control Plane

While the data plane is responsible for per-packet forwarding, the **control plane** determines the *routing information* used to populate the router's forwarding tables. The control plane executes routing protocols, exchanges topology data with peers, and computes the best paths to each destination network.

Overview. Routing protocols fall into three broad families:

- **Link-state protocols** (e.g., OSPF, IS-IS) — routers flood link connectivity information and each independently computes shortest paths.

- **Path-vector protocols** (e.g., BGP) — routers advertise entire paths between autonomous systems, enforcing policy and reachability across domains.
- **Centralized control** (e.g., SDN) — a logically centralized controller programs forwarding rules directly into network devices.

OSPF (Open Shortest Path First) — Link-State.

- Each router discovers its directly connected neighbors and links, generating a **Link-State Advertisement (LSA)**.
- LSAs are flooded throughout the routing area so all routers share the same complete network map (the *link-state database*).
- Each router runs Dijkstra's **Shortest Path First (SPF)** algorithm locally to compute the least-cost route to all destinations.
- Areas and hierarchical design (Area 0 as the backbone) improve scalability.

OSPF provides rapid convergence and loop-free paths within an autonomous system. Its operation depends on consistent LSAs and periodic hello messages to maintain neighbor relationships.

BGP (Border Gateway Protocol) — Path-Vector.

- BGP governs routing between autonomous systems (ASes) on the global Internet.
- Each router advertises **network prefixes** along with the full **AS path** to reach them.
- Path selection favors policies (local preference, AS-path length, MED) rather than pure shortest paths.
- Because it exchanges incremental updates and avoids full topology flooding, BGP scales to hundreds of thousands of routes.

Where OSPF builds distributed topology knowledge, BGP acts as a policy-driven information exchange system, ensuring stability and controllable route propagation between administrative domains.

SDN (Software-Defined Networking) — Centralized Control.

- SDN separates the **control plane** from the **data plane**.
- A logically centralized **SDN controller** maintains a global network view and installs forwarding rules in switches or routers using protocols such as **OpenFlow**, **gRPC**, or **NETCONF**.
- Enables centralized policy enforcement, dynamic traffic engineering, and rapid innovation without reconfiguring distributed routers.

In SDN, forwarding devices (switches) are simplified: they forward according to rules pushed by the controller, rather than running traditional routing protocols.

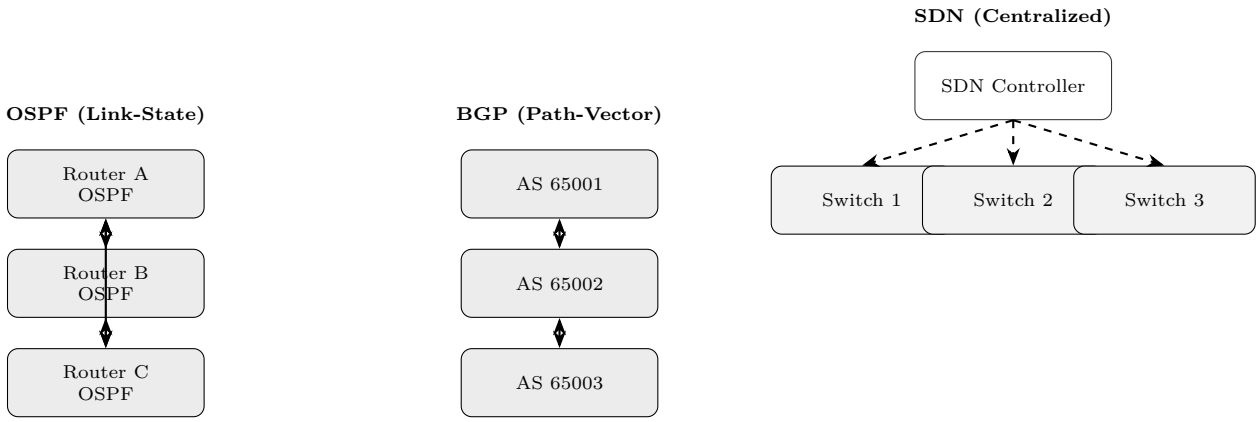


Figure 17: Control-plane paradigms: OSPF routers flood link-state information; BGP routers exchange AS-paths; SDN uses a centralized controller to install forwarding rules.

Comparison Summary.

Aspect	OSPF	BGP	SDN
Scope	Intra-domain	Inter-domain	Any (centralized)
Information shared	Link states	AS paths	Global topology in controller
Decision location	Each router	Each AS router	Centralized controller
Algorithm type	Dijkstra SPF	Policy/Path-vector	Programmable control logic
Convergence speed	Fast	Moderate (policy-driven)	Instant via reprogramming
Scalability	High (area hierarchy)	Very high (Internet-scale)	Controller-limited

Key Insights.

- The control plane defines the network's *logic*, computing routes and policies that the data plane enforces.
- OSPF exemplifies distributed computation and rapid convergence.
- BGP exemplifies policy-driven, large-scale federation of networks.
- SDN exemplifies centralized, programmable network management.

4.9 NAT and Private Addressing

As the IPv4 address space became exhausted, network designers sought methods to allow multiple internal devices to share a single public IP address. **Network Address Translation (NAT)** and **private addressing** together became the cornerstone of IPv4 scalability, enabling billions of devices to access the Internet despite the 32-bit address limit.

Private Address Ranges (RFC 1918). Certain IPv4 ranges are reserved for internal (non-routable) use within private networks. Packets carrying these addresses are not forwarded across the public Internet.

Block	Address Range	Typical Use
10.0.0.0/8	10.0.0.0 – 10.255.255.255	Large enterprise networks
172.16.0.0/12	172.16.0.0 – 172.31.255.255	Medium-sized private networks
192.168.0.0/16	192.168.0.0 – 192.168.255.255	Home and small-office networks

Hosts using these private addresses communicate externally through a gateway performing *translation* between private and public address spaces.

Network Address Translation (NAT) Operation. NAT modifies packet headers as they traverse a router or firewall:

1. Outgoing packets have their **source IP address** replaced with the router's public IP.
2. The NAT device records this mapping (internal IP + port ↔ external port) in a translation table.
3. Return traffic arriving on that external port is translated back to the original internal host.

Example:

```
Private: 192.168.1.10:52344  →  Public: 203.0.113.5:40001
Private: 192.168.1.11:52345  →  Public: 203.0.113.5:40002
```

This method is formally called **Port Address Translation (PAT)** or **NAT Overload**, since multiple internal clients can share one external IP by using unique port numbers.

Types of NAT.

- **Static NAT:** One-to-one mapping between private and public IPs (rare today).
- **Dynamic NAT:** Uses a pool of public addresses, assigning them temporarily.
- **PAT / NAPT:** Many-to-one mapping using ports (most common form in routers and firewalls).

Impact on Protocols. NAT disrupts the end-to-end transparency of IP networking:

- Applications embedding IP addresses in payloads (e.g., FTP, SIP, H.323) require special handling or helper protocols like **ALG (Application Layer Gateway)**.
- ICMP and some UDP-based protocols may fail if NAT tables expire before response packets return.
- Peer-to-peer or incoming connections are difficult without port forwarding or traversal methods such as **STUN**, **TURN**, or **ICE**.

NAT in IPv6. IPv6 restores abundant addressing (128 bits) and reestablishes the end-to-end model, removing the need for NAT. Nevertheless, enterprise deployments may still use **NPTv6 (Network Prefix Translation)** to map between internal and external IPv6 prefixes for administrative purposes.

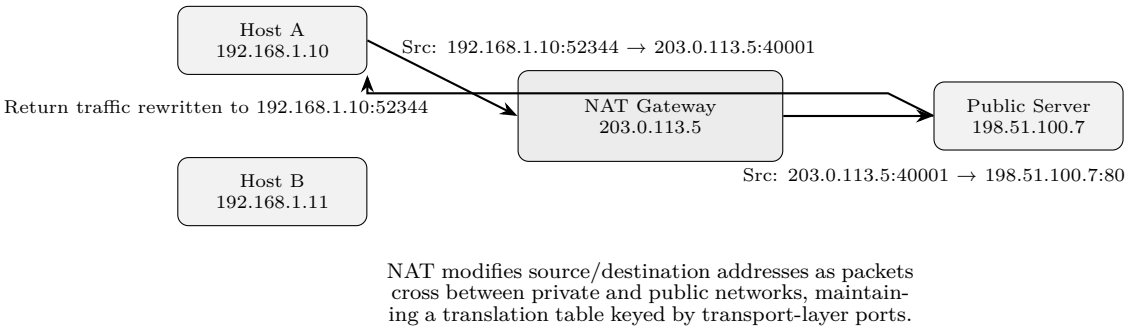


Figure 18: NAT operation: multiple private hosts share one public IP address using port-based translation.

Key Insights.

- NAT and private addressing extended IPv4's lifespan but broke end-to-end transparency.
- Most NAT devices perform port-level multiplexing (PAT/NAPT).
- IPv6 largely eliminates the need for NAT through vast address space and prefix-based routing.
- Nevertheless, NAT remains widespread in home and enterprise networks for isolation and policy enforcement.

4.10 Subnets, CIDR, and Supernetting Examples

IPv4 and IPv6 address spaces are structured hierarchically, allowing large blocks to be subdivided or aggregated for efficient routing. Subnetting divides a network into smaller parts; **CIDR (Classless Inter-Domain Routing)** generalizes this to arbitrary prefix lengths; and **supernetting** performs the reverse operation—combining adjacent subnets into a larger aggregate.

Subnetting. Originally, IPv4 networks were classified by fixed-size "classes" (A, B, C), but this was wasteful. Subnetting allows an administrator to divide a large network into smaller, more manageable segments by extending the **network prefix**.

- The subnet mask identifies which bits of the address denote the network versus the host.
- Example: `192.168.1.0/24` means the first 24 bits are the network portion.
- Subdividing into four /26 subnets yields:
  - `192.168.1.0/26` → hosts 0–63
  - `192.168.1.64/26` → hosts 64–127
  - `192.168.1.128/26` → hosts 128–191
  - `192.168.1.192/26` → hosts 192–255

Each subnet functions as an independent broadcast domain, often mapped to a physical or virtual LAN.

CIDR (Classless Inter-Domain Routing). CIDR, introduced in RFC 1519, replaced class-based addressing by representing any network as a prefix of arbitrary length:

$$\text{IP address/prefix length (e.g., 10.0.0.0/8 or 172.16.0.0/12)}$$

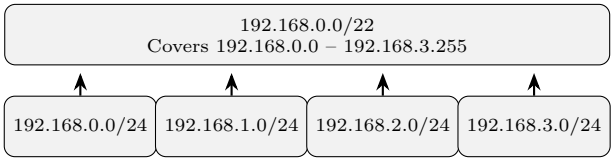
This notation allows flexible allocation of address blocks to match organizational needs and reduces routing table size through aggregation.

CIDR Example:

Network:	192.168.0.0/22
Covers:	192.168.0.0 through 192.168.3.255
Equivalent to:	Four contiguous /24 networks (192.168.0–3.0/24)

Supernetting (Route Aggregation). Supernetting combines adjacent subnets with identical prefix patterns into a single, larger route advertisement.

- Example: Four /24 networks (192.168.0.0–192.168.3.0) can be aggregated into one /22 block.
- Reduces the number of routing entries and improves lookup efficiency.
- Widely used by ISPs to advertise summarized routes.



Four contiguous /24 subnets combined into one /22 supernet for efficient route aggregation.

Figure 19: CIDR and supernetting relationship: four /24 subnets aggregated into a single /22 prefix.

Prefix Arithmetic. For IPv4, each bit in the mask represents a binary division:

$$\text{Number of hosts} = 2^{(32-\text{prefix length})} - 2$$

Example: A /26 network has $2^6 - 2 = 62$ usable host addresses. For IPv6, subnets are typically assigned on /64 boundaries, allowing for 2^{64} interface identifiers per subnet.

Key Insights.

- CIDR unifies subnetting and supernetting under a single prefix-based model.
- Route aggregation minimizes global routing table size.
- Subnetting enables internal structure and isolation in large organizations.
- IPv6 simplifies addressing hierarchy, but CIDR principles still apply.

4.11 Troubleshooting Tools

Network troubleshooting relies on diagnostic utilities that test connectivity, routing, name resolution, and packet-level behavior. These tools operate across various layers of the TCP/IP stack, helping engineers isolate failures in configuration, reachability, or performance.

1. `ping` — Connectivity and Latency.

- Uses **ICMP Echo Request (Type 8)** and **Echo Reply (Type 0)** messages.
- Verifies that an IP host is reachable and measures round-trip time (RTT).
- Common failure causes:
  - Host or router down
  - ICMP blocked by firewall
  - No route to destination

Example:

```
C:\> ping 8.8.8.8
Reply from 8.8.8.8: bytes=32 time=14ms TTL=117
```

2. `traceroute` (Linux) / `tracert` (Windows) — Path Discovery.

- Discovers the sequence of routers between source and destination.
- Sends packets with incrementally increasing TTL (hop limit) values.
- Each router that decrements TTL to zero replies with **ICMP Time Exceeded (Type 11)**.
- The final hop replies with **ICMP Echo Reply**.
- Useful for identifying routing loops, bottlenecks, or asymmetrical paths.

Example Output (abbreviated):

```
$ traceroute 8.8.8.8
1  192.168.1.1      1.0 ms
2  10.0.0.1         5.3 ms
3  203.0.113.5      8.7 ms
4  8.8.8.8          15.4 ms
```

3. `nslookup` / `dig` — DNS Resolution.

- Test domain name resolution from host to DNS server.
- `nslookup` is interactive and simple; `dig` provides detailed query sections.
- Useful for verifying DNS propagation or misconfiguration.

Example:

```
$ dig www.example.com
;; ANSWER SECTION:
www.example.com.  86400  IN  A  93.184.216.34
```

4. `netstat` — Socket and Routing Table Inspection.

- Displays active TCP/UDP sessions, listening ports, and protocol statistics.
- Can list the local routing table and interface statistics.
- Useful for checking which process owns a given port or whether connections are established.

Example:

```
$ netstat -an | grep 443
tcp  0  0  192.168.1.10:443  203.0.113.5:55322  ESTABLISHED
```

5. `ipconfig` / `ifconfig` / `ip` — Interface Configuration.

- Display or modify IP configuration and interface status.
- Linux's modern `ip` tool replaces legacy `ifconfig`:

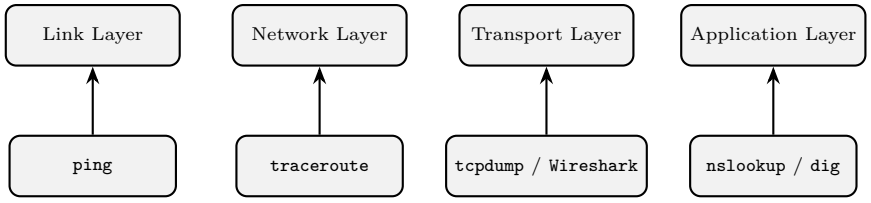
  ```
  $ ip addr show
  2: eth0: inet 192.168.1.100/24 brd 192.168.1.255 scope global
  ```

- Reveals subnet masks, MAC addresses, and routing table entries.

6. `tcpdump` / `Wireshark` — Packet Capture.

- Capture and inspect live network traffic at the packet level.
- `tcpdump` is CLI-based; `Wireshark` provides a GUI for protocol dissection.
- Useful for debugging connectivity issues, latency, or malformed packets.
- Example:

```
$ sudo tcpdump -i eth0 host 8.8.8.8
```



Troubleshooting tools correspond to different layers of the TCP/IP stack, from low-level connectivity to high-level name resolution.

Figure 20: Common troubleshooting tools aligned with TCP/IP layers.

Key Insights.

- `ping` and `traceroute` test connectivity and routing using ICMP.
- `nslookup`/`dig` verify DNS resolution.
- `netstat` and `ip` reveal host-side configuration.
- `tcpdump`/`Wireshark` provide packet-level visibility.
- Layer-based reasoning helps isolate where communication breaks down.

4.12 Quiz: Network Layer

Network Layer Quiz

Question 1. We noted that IP prefixes can be aggregated. Which of the following are equivalent?

- (10.0.0.0/16, 10.1.0.0/16, 10.2.0.0/16, 10.3.0.0/16) and 10.0.0.0/18
- Neither
- **(Correct)** (10.0.0.0/16, 10.1.0.0/16, 10.2.0.0/16, 10.3.0.0/16) and 10.0.0.0/14

Explanation: These four /16 blocks can be summarized as a single /14 prefix, since their first 14 bits are common.

Question 2. What is the service model offered by the Internet Protocol?

- Guaranteed delivery of packets
- Reliable stream of in-order packets
- **Best Effort**

Explanation: IP offers a best-effort datagram service — no guarantees of delivery, order, or integrity.

Question 3. Which of the following is a prefix for which `1.2.3.4` is covered?

- **1.2.0.0/22**
- 1.0.0.0/16
- 1.2.4.0/24
- 1.0.0.0/24

Explanation: The address 1.2.3.4 lies in the range 1.2.0.0–1.2.3.255, matching the /22 prefix.

Question 4. In the data plane, the destination IP address is looked up in what table?

- Routing Table
- MAC Learning Table
- **Forwarding Table**
- ARP Table

Explanation: The router performs a Longest Prefix Match (LPM) in the forwarding table (FIB).

Question 5. If two forwarding table entries overlap (e.g., 11.11.11.0/24 and 11.11.0.0/16), which one is chosen?

- The oldest entry
- The newest entry
- **The entry with the longest prefix**
- Round robin selection

Explanation: The most specific (longest) prefix always wins — this is the LPM rule.

Question 6. For a switching fabric, what is a benefit of a crossbar over a bus?

- **It can support multiple packets being switched simultaneously**
- It is standardized by the IETF
- It uses thicker wires

Explanation: Crossbars allow concurrent input–output transfers, increasing throughput.

Question 7. BGP (path vector) or OSPF (link state) or both? — Knows entire topology.

- BGP
- **OSPF**
- Both

Explanation: OSPF floods full topology information; BGP exchanges only paths.

Question 8. BGP (path vector) or OSPF (link state) or both? — Handles failure of devices.

- BGP
- OSPF
- **Both**

Explanation: OSPF reconverges on intra-domain link/node failures via LSAs and SPF recomputation (fast). BGP also handles failures (inter-domain) by withdrawing routes after session loss/keepalive timeout and reselecting paths—just typically slower and policy-driven.

Question 9. BGP (path vector) or OSPF (link state) or both? — May pick a different best route than its neighbor.

- **BGP**
- OSPF
- Both

Explanation: BGP uses local routing policy, so choices differ between peers.

Question 10. BGP (path vector) or OSPF (link state) or both? — Is the main protocol used in the Internet.

- **BGP**
- OSPF

- Both

Explanation: BGP is the Internet's inter-domain routing protocol.

Question 11. BGP (path vector) or OSPF (link state) or both? — Exchanges with neighbor all routes known.

- BGP
- **OSPF**
- Both

Explanation: OSPF floods LSAs describing the full network graph; BGP advertises selected routes.

Question 12. BGP (path vector) or OSPF (link state) or both? — Works in a decentralized manner.

- BGP
- OSPF
- **Both**

Explanation: Both operate without central control; each router runs local algorithms.

Question 13. Before sending an IP packet, which protocol resolves the MAC address of the destination or next hop?

- **ARP**
- DHCP
- TCP
- DNS

Explanation: ARP (Address Resolution Protocol) maps IP to MAC for local delivery.

Question 14. Which best describes how `ping` works?

- **IP header specifies ICMP as the protocol, and sends an ICMP request to the destination, which replies with an ICMP reply.**
- Ethernet header specifies ICMP as the type, and routers respond based on ICMP header.
- IP options field has a flag marking ping requests.
- Separate link-layer management channel for ICMP.

Explanation: Ping uses ICMP Echo Request and Reply messages to test reachability and round-trip time.

5 Transport Layer (TCP/UDP)

Introducing the Transport Layer: Process-to-Process Communication

The **Transport Layer** sits above the Network Layer and provides logical communication between *processes* running on different hosts. Whereas the Network Layer delivers datagrams from one host to another, the Transport Layer ensures that data originating from a specific application on one machine is delivered correctly to the corresponding application on another. In other words, it converts host-to-host delivery into *process-to-process* delivery.

Position in the Stack. The Transport Layer receives data from the Application Layer and segments it into manageable units (segments or datagrams) before passing them to the Network Layer for transmission. At the destination, it reassembles received segments, handles reliability if required, and delivers the data to the correct process identified by a port number. This layer thus acts as the bridge between network connectivity and user applications.

Design Objectives. Key goals of the transport layer include:

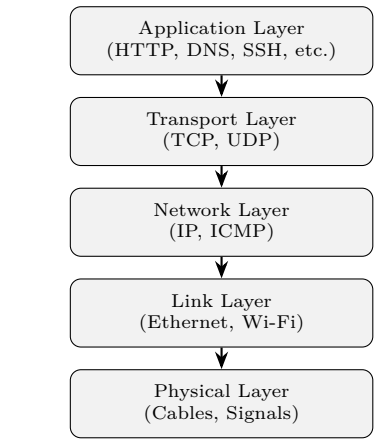
- **End-to-end communication:** Abstracts the physical and routing complexities of the underlying network.
- **Reliability:** Detects and recovers from losses, errors, or reordering (TCP).
- **Flow control:** Ensures that a fast sender does not overwhelm a slow receiver.
- **Congestion control:** Adjusts sending rates to avoid overloading the network.
- **Multiplexing:** Allows multiple applications to share a single network connection simultaneously.

Major Protocols. The Internet supports two main transport protocols:

- **UDP (User Datagram Protocol):** Lightweight, connectionless, and best-effort—used by latency-sensitive applications such as DNS, streaming, and gaming.
- **TCP (Transmission Control Protocol):** Reliable, connection-oriented, and ordered—used by web browsers, email, and file transfers.

Both coexist on the Internet, trading off reliability versus latency according to application needs.

Layer Relationship. In the TCP/IP model, the Transport Layer corresponds roughly to Layer 4 of the OSI model. It relies on the Network Layer for addressing and routing, while providing a uniform interface to the Application Layer above it. This separation allows innovation in application design (HTTP, DNS, SSH, QUIC) without altering lower network mechanisms.



The transport layer abstracts network delivery into process-to-process communication and underpins all Internet applications.

Figure 21: Position of the Transport Layer within the TCP/IP stack.

5.1 Service Model and Multiplexing

The **Transport Layer** provides logical communication between processes, not just hosts. While the Network Layer (e.g., IP) delivers datagrams between devices, the Transport Layer distinguishes among multiple concurrent applications by using **ports** and **sockets** to perform **multiplexing** (combining many streams into one) and **demultiplexing** (delivering to the correct receiver).

Service Model. The transport layer offers an abstract service to applications:

- **Connection-oriented (TCP):** Reliable, ordered, byte-stream service.
- **Connectionless (UDP):** Unreliable, message-oriented datagram service.

Applications use these services to communicate as if a direct link existed between endpoints, regardless of underlying path or topology.

Ports. Ports are 16-bit numerical identifiers that differentiate concurrent processes on a host:

- Range: 0–65535
- **Well-known ports (0–1023):** Assigned to standard services (e.g., 80 for HTTP, 25 for SMTP, 443 for HTTPS)
- **Registered ports (1024–49151):** Used by user applications or vendors.
- **Ephemeral ports (49152–65535):** Dynamically allocated for outgoing connections.

Sockets and Endpoints. A **socket** represents a communication endpoint defined by a combination of:

$$(\text{IP address}, \text{Transport protocol}, \text{Port number})$$

Each side of a transport connection has its own socket. For example, a TCP session can be identified by a 4-tuple:

$$(\text{Source IP}, \text{Source Port}, \text{Destination IP}, \text{Destination Port})$$

This 4-tuple uniquely identifies a flow between two processes across the Internet.

Multiplexing and Demultiplexing.

- **Multiplexing:** The transport layer gathers data from multiple application processes, attaches headers containing source ports, and sends them to the network layer.
- **Demultiplexing:** The receiving host examines the destination port and delivers each segment to the correct application socket.

Thus, many applications (web browsers, SSH sessions, email clients) can simultaneously share one IP address without interfering.

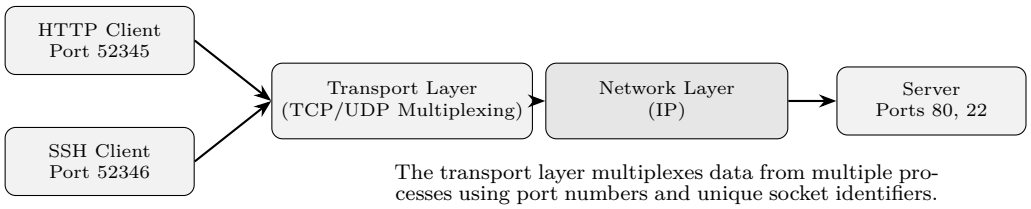


Figure 22: Multiplexing and demultiplexing: multiple applications share one host IP through distinct transport-layer ports.

Example.

$$\begin{aligned} &\text{Client Socket:} \quad (192.168.1.10, 52345, 93.184.216.34, 80) \quad \text{HTTP session} \\ &\text{Client Socket:} \quad (192.168.1.10, 52346, 203.0.113.5, 22) \quad \text{SSH session} \end{aligned}$$

Key Insights.

- Ports identify processes; IP addresses identify hosts; together they form sockets.
- Multiplexing allows multiple concurrent connections over a single IP.
- TCP and UDP manage connections independently per socket pair.
- Sockets form the boundary where the application interacts with the transport service.

5.2 UDP: Simplicity and Use Cases

The **User Datagram Protocol (UDP)** provides a minimal, message-oriented transport service. Unlike TCP, UDP is *connectionless* and offers **no built-in reliability, ordering, or congestion control**. Its low overhead and preservation of message boundaries make it ideal for applications that can tolerate loss or implement their own recovery logic.

Service Characteristics.

- **Connectionless:** No handshake; a datagram can be sent at any time.
- **Datagram semantics:** Each *send* corresponds to one *receive* (message boundaries preserved).
- **Best-effort delivery:** No retransmissions, no in-order guarantees.
- **Low overhead:** 8-byte header; minimal per-flow state in endpoints and middleboxes.
- **Checksum:** End-to-end integrity over data *and* a pseudo-header (source/destination IP, etc.). Mandatory in IPv6; optional (but widely used) in IPv4.

Header Format (8 bytes).

Source Port (16b)	Destination Port (16b)
Length (16b)	Checksum (16b)

When to Choose UDP.

- **DNS:** Very small queries/responses; app-level retries are cheap; latency matters.
- **Streaming media (RTP), gaming, voice/video:** Timeliness outweighs perfect reliability; late data is as bad as lost data.
- **Low-latency telemetry / real-time control:** Minimal overhead and head-of-line blocking avoidance.
- **Custom protocols and modern transports: QUIC** (HTTP/3) builds reliability, security (TLS 1.3), and congestion control *over* UDP in user space.

Design Trade-offs and Best Practices.

- **App-level reliability (if needed):** Implement sequence numbers, ACK/NACK, and retransmissions in the application or use a library (e.g., QUIC, RTP with FEC).
- **Congestion awareness:** Even with UDP, senders should employ congestion control to be network-friendly (rate limiting, pacing).
- **MTU considerations:** Keep datagrams below the path MTU to avoid IP fragmentation; fragmentation loss drops the entire message.
- **NAT/firewalls:** UDP "flows" are tracked with idle timeouts; keep-alives or STUN/TURN/ICE may be required for traversal.
- **Security:** Use **DTLS** or **QUIC** for confidentiality/integrity when needed; plain UDP provides neither.

Typical Examples.

- **DNS over UDP:** Client → server query on port 53; small response returns to client's ephemeral port.
- **RTP over UDP:** Continuous media packets at fixed intervals; late packets are discarded rather than retransmitted.

Key Takeaways.

- UDP's strength is **simplicity and low latency**; it shifts reliability and ordering to the application (or higher-layer protocols like QUIC).
- It is well-suited to **loss-tolerant** or **delay-sensitive** workloads and for transports designed in user space.
- Careful sizing (avoid fragmentation), NAT keep-alives, and optional security layers are essential in production use.

5.3 TCP Overview and Header Fields

The **Transmission Control Protocol (TCP)** provides reliable, ordered, and full-duplex communication between processes. Unlike UDP, which treats each message independently, TCP establishes a logical *connection* and maintains extensive state at both endpoints.

Core Principles. TCP transforms an unreliable IP service into a reliable, byte-stream channel. It accomplishes this through:

- **Sequencing:** Every byte in the stream has a unique sequence number.
- **Acknowledgments:** Receivers explicitly confirm the highest in-order byte received.
- **Flow Control:** Using a window mechanism to regulate sender rate.
- **Congestion Control:** Adjusting transmission based on perceived network congestion.

TCP Header Structure. The TCP header (20–60 bytes) carries all necessary information for reliability and flow control.

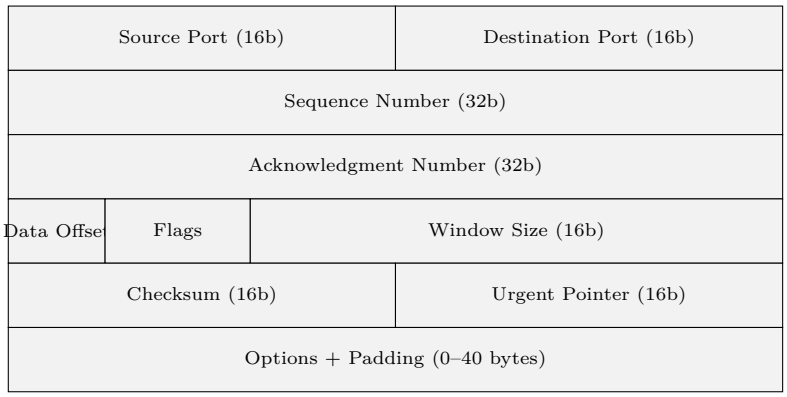


Figure 23: Simplified structure of the TCP header.

Sequence and Acknowledgment Numbers.

- **Sequence Number (32 bits):** Identifies the first byte of data in the current segment relative to the stream's start. Each byte is numbered, allowing ordered reconstruction and detection of loss or duplication.
- **Acknowledgment Number (32 bits):** Indicates the next expected sequence number. In other words, all bytes up to (ACK–1) have been successfully received.
- **Initial Sequence Number (ISN):** Randomly chosen for each direction during connection setup (SYN exchange) to avoid collisions and improve security.

Control Flags (6 + 2 ECN bits). TCP includes a small bit field where each bit represents a control signal:

- **URG:** Urgent pointer valid (rarely used today).
- **ACK:** Acknowledgment field valid; set in nearly all segments after setup.
- **PSH:** Push data immediately to the application (hint, not a guarantee).
- **RST:** Reset connection due to error.
- **SYN:** Synchronize sequence numbers during connection setup.

- **FIN:** Sender finished sending data.
- **ECE / CWR:** Explicit Congestion Notification signaling (optional).

Flow Control and Window Advertisement. TCP provides end-to-end flow control via a 16-bit **Window Size** field:

- Specifies how many bytes (beyond the ACKed sequence) the receiver can currently buffer.
- Prevents a fast sender from overwhelming a slow receiver.
- The sender computes a *usable window* = (advertised window) − (unacknowledged bytes).
- With the **Window Scale** option (RFC 7323), the effective window can exceed 65,535 bytes to support high-bandwidth links.

Additional Fields.

- **Checksum:** Covers header, data, and pseudo-header (source/dest IPs) for end-to-end integrity.
- **Urgent Pointer:** Valid only if URG is set; identifies end of urgent data.
- **Options:** Extend capabilities—e.g., Maximum Segment Size (MSS), Timestamps, SACK, and Window Scaling.

Bidirectional Nature. Each TCP connection maintains independent sequence and acknowledgment spaces in both directions. Every data segment carries not only its sequence number but also an acknowledgment of data flowing the other way, making TCP **full-duplex and symmetric**:contentReference[oaicite:3]index=3.

Key Insights.

- TCP headers are richer than UDP to support reliability, sequencing, and flow control.
- The sliding window mechanism and ACKs are essential for maintaining throughput and reliability.
- Control flags drive connection setup (SYN), teardown (FIN), and error recovery (RST).

5.4 Connection Management

TCP uses explicit control signaling to establish and terminate reliable connections between hosts. This management process ensures that both sides agree on initial sequence numbers, synchronize state, and cleanly release resources.

Connection Establishment — The Three-Way Handshake. The three-way handshake coordinates sequence numbers and ensures both endpoints are ready to send and receive.

1. **SYN:** The client sends a segment with the SYN flag set and its initial sequence number (ISN_C).
2. **SYN–ACK:** The server responds with a segment that both acknowledges the client's ISN (ACK = $\text{ISN}_C + 1$) and includes its own ISN_S.
3. **ACK:** The client acknowledges the server's ISN (ACK = $\text{ISN}_S + 1$), completing the handshake.

At this point, both sides know each other's sequence number spaces and can begin full-duplex data transfer.

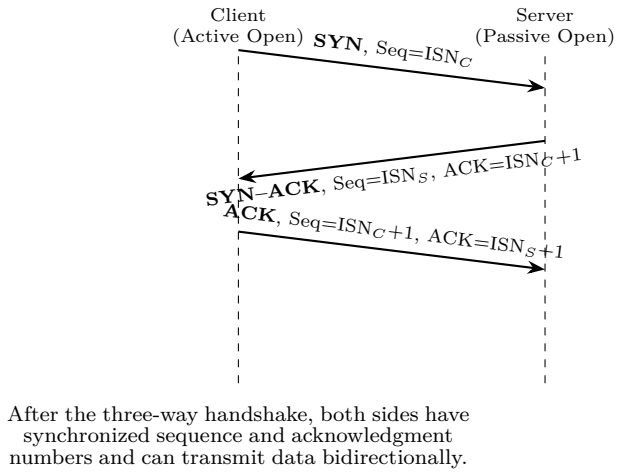


Figure 24: TCP three-way handshake establishing a connection between client and server.

Simultaneous Open. Though rare, both hosts can send SYNs simultaneously. Each then acknowledges the other's SYN, resulting in a valid connection with both sides transitioning from SYN-SENT to ESTABLISHED.

Connection Termination — Graceful Teardown. TCP uses a four-segment exchange to close each half of a connection cleanly:

1. Host A sends **FIN** to indicate it has no more data to send.
2. Host B replies with **ACK**, acknowledging the FIN.
3. When Host B is ready to close its side, it sends its own **FIN**.
4. Host A replies with **ACK**, completing termination.

Because each direction closes independently, TCP supports **half-close** (one side can stop sending while still receiving).

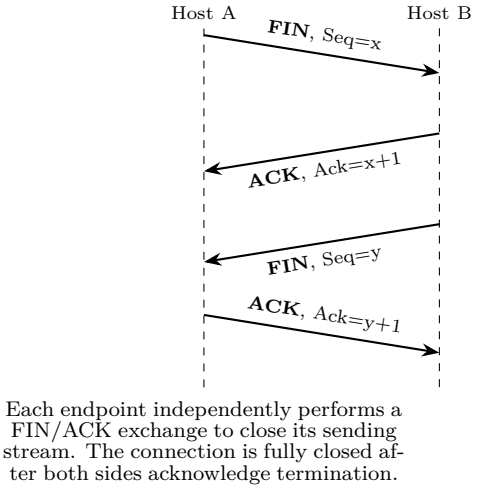


Figure 25: TCP connection teardown: four-segment exchange for a full-duplex close.

States and Transitions. TCP maintains a **finite state machine (FSM)** to track connection progress:

CLOSED → SYN-SENT → ESTABLISHED → FIN-WAIT-1 → CLOSE-WAIT → TIME-WAIT → CLOSED

The **TIME-WAIT** state ensures late or duplicate segments from old connections are discarded safely before freeing socket resources.

Key Insights.

- The 3-way handshake ensures reliable synchronization of sequence spaces.
- Teardown requires each direction to close independently (half-close model).
- TIME-WAIT is essential for correctness, not a "bug"; it prevents old packets from contaminating new sessions.
- Connection management contributes to TCP's reliability and robustness but adds latency to setup and teardown.

5.5 Reliable Data Transfer

A central function of TCP is to transform the unreliable, best-effort service of IP into a reliable, ordered byte stream. This reliability is achieved through four core mechanisms: **sequence numbering**, **acknowledgments**, **retransmissions**, and **cumulative acknowledgments**. Together, they ensure data arrives intact, in order, and without duplication.

Sequence Numbers. Each byte in the TCP data stream is assigned a unique **sequence number**. The **Sequence Number** field in the TCP header identifies the position of the first byte in that segment's payload. This numbering enables:

- Detection of missing or duplicate data.
- Reordering of out-of-sequence segments.
- Sliding-window control of in-flight data.

Example: If a segment carries 1000 bytes starting with sequence number 5001, the next segment should begin at 6001.

Acknowledgments. TCP employs **positive acknowledgments** with **cumulative semantics**. Each segment's **Acknowledgment Number** field indicates the next byte expected from the peer. Thus, an ACK=6001 implies that all bytes up to 6000 were received successfully.

- ACKs are piggybacked on data segments whenever possible.
- If no data is ready to send, standalone ACKs are transmitted (pure ACKs).
- The absence of acknowledgment within a timeout triggers retransmission.

Retransmissions. TCP uses retransmissions to recover from packet loss:

- **Timeout-based:** If an ACK is not received within a calculated **Retransmission Timeout (RTO)**, the unacknowledged segment is resent.
- **Fast Retransmit:** If three duplicate ACKs (same ACK number) are received, TCP infers a loss and retransmits immediately, avoiding the full timeout.
- Retransmitted segments carry the same sequence numbers as the originals to maintain stream integrity.

Cumulative Acknowledgments. TCP's ACK scheme is cumulative — it acknowledges *all* data up to the highest contiguous byte received:

- Reduces ACK traffic (a single ACK can confirm multiple segments).
- Simplifies receiver logic: only the next expected sequence is tracked.
- However, it does not individually acknowledge out-of-order segments.

To improve efficiency during high loss or reordering, TCP extensions such as **Selective Acknowledgment (SACK)** allow explicit reporting of non-contiguous data blocks.

Sliding Window Mechanism. Reliable delivery is managed using a **sliding window**, which defines how many bytes can be "in flight" (sent but not yet acknowledged). When ACKs arrive, the window slides forward, allowing the sender to transmit more data.

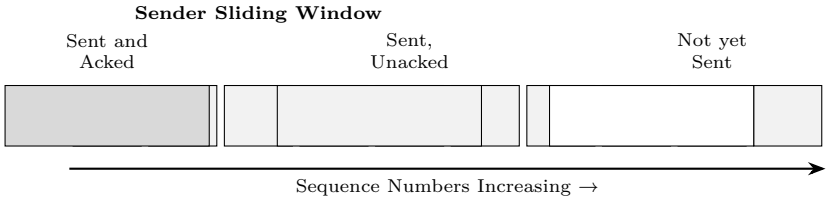


Figure 26: Sliding window at sender: ACKs advance the window, allowing new data to be transmitted.

Retransmission Timer and RTT Estimation. The retransmission timeout (RTO) dynamically adapts to round-trip delay:

$$\text{RTO} = \text{SRTT} + 4 \times \text{RTTVAR}$$

where SRTT is the smoothed RTT estimate and RTTVAR measures RTT variance. This adaptive algorithm (RFC 6298) prevents premature retransmissions and congestion collapse.

Duplicate Acknowledgments. If the receiver gets an out-of-order segment (due to loss or reordering), it immediately re-sends an ACK for the last in-order byte received. Three identical ACKs trigger **Fast Retransmit**, allowing recovery before timeout.

Key Insights.

- Sequence and acknowledgment numbers form the backbone of TCP reliability.
- Cumulative ACKs confirm contiguous progress; duplicate ACKs signal possible loss.
- Retransmissions (timeout and fast) recover missing data efficiently.
- The sliding window balances throughput and reliability, pacing transmission based on feedback.
- Adaptive RTO estimation is critical for stability under varying network conditions.

5.6 Flow Control via Sliding Window

TCP's **flow control** ensures that a fast sender does not overwhelm a slow receiver. This mechanism operates end-to-end, using the **receiver window (rwnd)** to advertise how much additional data the receiver can buffer at any given time.

Motivation. Network congestion control protects the network path, but *flow control* protects the receiving host. Each endpoint maintains a receive buffer; when it fills, the receiver must slow down the sender. Without flow control, an overrun buffer would cause packet loss and retransmissions — wasting bandwidth and CPU.

Receiver Window (rwnd). The receiver window represents how much additional unacknowledged data the receiver is prepared to accept:

$$rwnd = R_{buf} - (R_{next} - R_{acked})$$

where:

- R_{buf}: total receive buffer size,
- R_{next}: next expected sequence number,
- R_{acked}: highest in-order byte acknowledged to the application.

The receiver advertises its current *rwnd* in every TCP segment (in the **Window Size** field). The sender then limits the amount of in-flight data to the smaller of *rwnd* and the congestion window (*cwnd*).

Sender Behavior. The sender tracks how much data is outstanding:

$$\text{Effective send window} = \min(\text{rwnd}, \text{cwnd})$$

When the receiver's buffer fills, *rwnd* may reach zero, forcing the sender to pause transmission. To detect when the receiver can resume, the sender transmits small **window probe** segments periodically.

Buffer Management at the Receiver. The receiver's buffer temporarily holds incoming data until it can be delivered to the application. Typical stages:

1. Data arrives and is validated (sequence range, checksum).
2. In-order data is delivered immediately to the application.
3. Out-of-order segments are queued until missing data arrives.
4. As the application reads from the buffer, space is freed and *rwnd* increases.

Window Advertisement Dynamics. The receiver continually updates its advertised window in outgoing ACKs:

$$\text{Advertised Window} = R_{buf} - (\text{Bytes received but not yet read})$$

Thus, TCP flow control is entirely feedback-driven — the sender always respects the most recent window received.

Example. Consider a receiver with a 32 KB buffer:

- Initially, $rwnd = 32$ KB. The sender may transmit up to 32 KB of data before pausing.
- As data fills the buffer, *rwnd* shrinks.
- Once the application consumes 16 KB, the receiver advertises $rwnd = 16$ KB.
- The sender can then transmit another 16 KB of data.

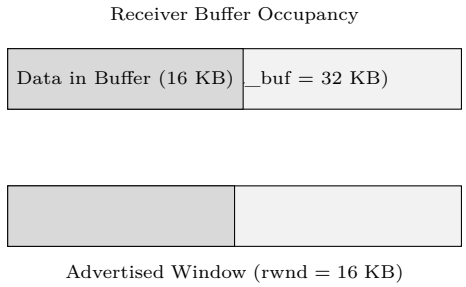


Figure 27: Receiver window shrinks as buffer fills, then grows as data is read by the application.

Zero Window and Probing. If the receiver buffer becomes full, it may advertise a **zero window**, halting transmission. The sender enters a *persist state*, sending periodic 1-byte **window probes** to elicit an updated window value. This prevents a deadlock in which both sides stop sending (receiver's ACKs are suppressed, sender waits indefinitely).

Window Scaling. Since the TCP window field is only 16 bits (max 65,535 bytes), high-bandwidth networks use the **Window Scale option** (RFC 7323). This option extends the effective window size:

$$\text{Effective Window} = \text{Advertised Window} \times 2^{\text{scale}}$$

allowing windows of several megabytes for gigabit links with high latency.

Key Insights.

- The receiver window implements flow control by advertising how much buffer space remains.
- The sender's usable window = $\min(\text{rwnd}, \text{cwnd})$, ensuring both receiver and network are protected.
- Window probes prevent deadlock when the receiver advertises zero window.
- Window scaling is vital for high-speed, high-latency (long fat) networks.

5.7 Congestion Control Algorithms

TCP's **congestion control** protects the network from overload by dynamically adjusting the sender's transmission rate. While flow control prevents receiver buffer overflow, congestion control prevents *network* buffer overflow—especially in routers and switches. All major TCP implementations regulate congestion through an adaptive variable called the **congestion window** (*cwnd*).

Basic Concept. At any time, a sender may have at most:

$$\text{in-flight data} \leq \min(\text{rwnd}, \text{cwnd})$$

where:

- rwnd: receiver-advertised window (flow control)
- cwnd: sender-imposed congestion limit (network control)

The sender increases or decreases *cwnd* based on perceived congestion, using packet loss or delay as feedback.

1. Slow Start. At connection startup, TCP must discover the available capacity. It begins conservatively, with a small initial congestion window (typically 10 MSS in modern TCP), and increases exponentially until a loss occurs or a threshold is reached.

$$cwnd \leftarrow cwnd + 1 \text{ MSS per ACK}$$

$$\text{After each RTT: } cwnd \approx 2 \times cwnd_{\text{previous}}$$

When congestion (loss or timeout) occurs, TCP infers the network is saturated and reduces *cwnd*. The variable ssthresh (slow-start threshold) defines when exponential growth transitions to linear growth.

2. Congestion Avoidance. Once $cwnd \geq$ ssthresh, TCP enters **congestion avoidance**, where growth becomes additive and losses trigger multiplicative decreases (AIMD — Additive Increase, Multiplicative Decrease):

$$cwnd \leftarrow cwnd + \frac{1}{cwnd} \text{ per ACK}$$

$$\text{on loss: } cwnd \leftarrow \frac{1}{2} \times cwnd$$

This steady probing of capacity maintains fairness among competing flows.

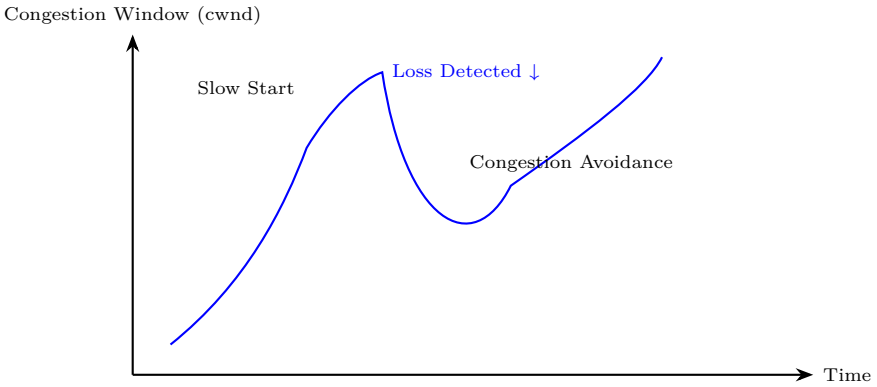


Figure 28: TCP congestion window evolution through Slow Start and Congestion Avoidance (AIMD behavior).

3. Fast Retransmit and Fast Recovery. Timeouts are costly, so TCP introduces faster recovery mechanisms based on duplicate ACKs:

- When 3 duplicate ACKs are received, TCP assumes a segment was lost (not reordered) and immediately retransmits it — **Fast Retransmit**.
- Upon detecting this loss, TCP halves *cwnd* (multiplicative decrease) and enters **Fast Recovery**.
- During recovery, each duplicate ACK inflates *cwnd* temporarily, allowing the sender to continue transmitting new segments until the loss is recovered.

Once an ACK acknowledging all outstanding data arrives, TCP exits Fast Recovery and resumes additive increase.

4. Variants of TCP Congestion Control.

TCP Reno (RFC 5681). Classic AIMD algorithm with Fast Retransmit and Fast Recovery.

- Exponential growth in Slow Start, linear in Congestion Avoidance.
- On loss: $cwnd \to cwnd/2$, ssthresh $= cwnd/2$.
- Suitable for moderate bandwidth-delay products.

TCP NewReno. Enhances Reno's Fast Recovery by handling multiple packet losses within one window without waiting for a full timeout.

- Partial ACKs trigger additional retransmissions within the same recovery episode.
- Avoids re-entering Slow Start after each lost segment.

TCP CUBIC (Linux default). CUBIC (RFC 8312) replaces AIMD's linear growth with a cubic function of time since the last loss:

$$cwnd(t) = C(t - K)^3 + cwnd_{\max}$$

where K ensures $cwnd$ returns to its previous maximum smoothly. CUBIC is less sensitive to RTT variations and achieves higher throughput on long-fat networks.

TCP BBR (Bottleneck Bandwidth and RTT). Developed by Google, BBR (RFC 9330) abandons loss-based control entirely. Instead, it estimates:

- **Bottleneck bandwidth (BtlBw)** — measured as delivered data / RTT.
- **RTT (min_rtt)** — lowest observed round-trip time.

BBR then regulates the send rate to match the estimated delivery rate, keeping queues nearly empty and latency low. It operates in repeating phases (Startup, Drain, ProbeBW, ProbeRTT) to continuously refine its model.

Comparison Summary.

Algorithm	Feedback Type	Behavior Summary
TCP Reno	Loss-based	AIMD; halves cwnd on loss; linear recovery
TCP NewReno	Loss-based	Improved multiple loss recovery
TCP CUBIC	Loss-based (time function)	Cubic growth; RTT-independent fairness
TCP BBR	Model-based (bandwidth/RTT)	Maintains high throughput, low latency

Key Insights.

- Congestion control prevents buffer overflow and global collapse.
- Slow Start probes capacity; Congestion Avoidance maintains equilibrium.
- Fast Retransmit and Fast Recovery improve responsiveness.
- Reno and NewReno dominate legacy networks; CUBIC and BBR define modern Internet performance.
- The interaction between **flow control (rwnd)** and **congestion control (cwnd)** determines overall TCP throughput.

5.8 Performance Enhancements

While TCP's core design guarantees reliability, additional mechanisms are necessary to improve efficiency and performance on real networks. These **performance enhancements** reduce unnecessary transmissions, better utilize available bandwidth, and adapt to modern high-speed, high-latency environments.

1. Nagle's Algorithm and Delayed ACKs. TCP's byte-stream nature can cause "tinygram" problems — sending many small segments (e.g., one keystroke per packet) leads to inefficiency and congestion.

Nagle's Algorithm (RFC 896):

- Combines small outgoing segments until the previous segment is acknowledged.
- Prevents excessive small packets (e.g., Telnet, SSH, chat traffic).
- Rule: If unacknowledged data exists, buffer small writes until an ACK arrives or enough data accumulates to fill a segment (MSS).

$$\text{Send if (segment empty)} \vee \text{(new data fills MSS)}$$

Benefit: Reduces overhead and congestion on slow or high-latency links. **Tradeoff:** Increases latency for interactive applications.

Delayed ACKs (RFC 1122):

- The receiver waits briefly (up to 200 ms) before sending an ACK.
- Goal: Combine ACKs with outbound data (piggybacking) or acknowledge multiple segments at once.
- Reduces reverse-path traffic by up to 50%.

Interaction Issue: If both Nagle's algorithm and Delayed ACKs are enabled, a "feedback deadlock" may occur—small sends await ACKs that are delayed. Interactive applications (e.g., SSH, Telnet) often disable Nagle via:

```
setsockopt(TCP_NODELAY)
```

2. Selective Acknowledgments (SACK). Standard TCP uses **cumulative ACKs**, which can only confirm data up to the first gap in the sequence. When multiple packets are lost within one window, this leads to redundant retransmissions.

Selective Acknowledgment (SACK, RFC 2018) allows receivers to explicitly report which non-contiguous blocks of data were received successfully:

- Sender retransmits only missing segments rather than the entire window.
- SACK information is carried as TCP options in ACKs.
- Requires both peers to negotiate SACK capability during the handshake.

Example SACK Block:

$$\text{SACK} = [\text{Left Edge: } 5001, \text{ Right Edge: } 7001)$$

indicating all bytes in that range have arrived, even if earlier bytes are missing.

With SACK, receiver explicitly reports received segments (gray) so sender retransmits only lost blocks (white).

Figure 29: Selective Acknowledgment (SACK) identifies exactly which segments arrived.

3. TCP Timestamps and RTT Estimation. Accurate RTT estimation is critical for setting retransmission timeouts (RTO) and avoiding spurious retransmissions.

TCP Timestamps Option (RFC 7323):

- Each segment carries a 32-bit **TSval** (timestamp value) and echoes the peer's last **TSecr**.
- Enables precise RTT measurement even when ACKs are delayed or packets are retransmitted.
- Prevents *sequence number wrapping* issues in high-speed links (PAWS — Protect Against Wrapped Sequence numbers).

RTT Estimation (RFC 6298): TCP maintains smoothed estimates:

$$SRTT = (1 - \alpha) \times SRTT + \alpha \times RTT_{\text{sample}}$$

$$RTTVAR = (1 - \beta) \times RTTVAR + \beta \times |RTT_{\text{sample}} - SRTT|$$

$$RTO = SRTT + 4 \times RTTVAR$$

Typical parameters: $\alpha = 1/8, \beta = 1/4$.

Combined Impact.

- Nagle's algorithm and Delayed ACKs optimize bandwidth at low cost but may introduce latency.
- SACK dramatically improves recovery from multiple losses.
- TCP timestamps refine RTT estimation and enable safe operation on high-speed networks.

Key Insights.

- Nagle's Algorithm reduces network overhead from small packets.
- Delayed ACKs minimize reverse-path load.
- SACK allows efficient loss recovery and is standard in modern TCP.
- Timestamps improve performance and stability across diverse network conditions.

5.9 Connection Termination and TIME_WAIT

TCP connections are explicitly terminated to ensure that both sides have successfully transmitted all data and acknowledged all bytes. The teardown process also protects against delayed or duplicated packets from previous connections that could otherwise corrupt new sessions.

Graceful Termination Recap. As described earlier, TCP termination uses a **four-segment exchange**:

1. One endpoint (typically the client) sends a **FIN** to signal that it has finished sending data.
2. The peer replies with an **ACK**, acknowledging the FIN.
3. The peer later sends its own **FIN** when ready to close.
4. The original sender replies with a final **ACK**.

This half-duplex teardown allows each side to independently finish transmission, providing clean, reliable closure.

TIME_WAIT State. After sending the final ACK, the endpoint that initiated the close enters the **TIME_WAIT** state (also called *2MSL wait*). This state lasts for twice the **Maximum Segment Lifetime (MSL)**, typically between 1–4 minutes.

Purpose of TIME_WAIT:

- Ensures that delayed segments from the just-closed connection expire before a new one reuses the same socket pair (IP, port).
- Guarantees reliable connection termination by allowing retransmission of the final ACK if the peer's last FIN is lost.

If TIME_WAIT did not exist:

- Stray packets from an old session could be misinterpreted by a new connection with the same port numbers.
- The peer might retransmit its FIN and receive no valid response, leaving the connection in an inconsistent state.

Duration and Implementation. The TIME_WAIT timer equals **2 × MSL**, where MSL is the maximum lifetime of a TCP segment in the network (commonly 30 or 60 seconds). Typical TIME_WAIT durations:

- Linux: 60 seconds (configurable via `/proc/sys/net/ipv4/tcp_fin_timeout`)
- BSD/Windows: 2 minutes (default)

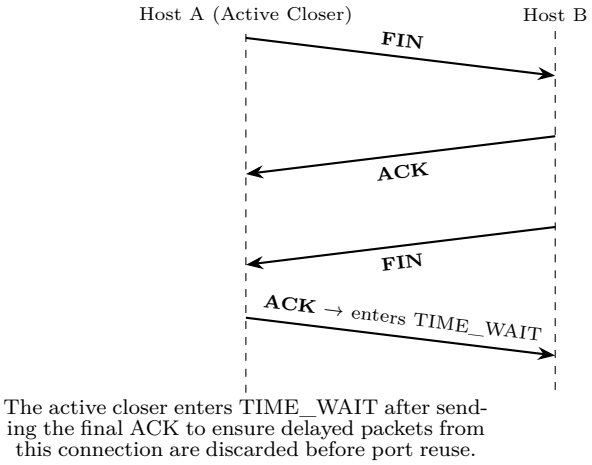


Figure 30: TCP teardown sequence and TIME_WAIT behavior.

RST (Reset) Handling. A TCP segment with the **RST flag** immediately aborts a connection, discarding any in-flight data. It is used when:

- A segment arrives for a nonexistent or already-closed connection.
- An application calls `SO_LINGER` with zero timeout (abortive close).

While efficient for error handling, careless RSTs can corrupt active sessions — for instance, middleboxes or firewalls injecting RSTs to block traffic.

Common Pitfalls and Misconceptions.

- **TIME_WAIT Accumulation:** Servers handling thousands of short-lived connections (e.g., web servers) may appear to have many sockets stuck in TIME_WAIT. This is normal — not a leak — and protects session integrity.
- **Premature Port Reuse:** Some implementations (e.g., Linux with `tcp_tw_reuse`) allow reusing TIME_WAIT sockets prematurely for outbound connections, but this can cause rare data corruption.
- **Asymmetric Close:** Only the side performing the final ACK enters TIME_WAIT. Passive closers typically go directly to CLOSED.
- **RST vs FIN:** FIN allows graceful half-close; RST aborts immediately, possibly discarding unsent data.

Key Insights.

- The TIME_WAIT state ensures complete connection reliability and protects against delayed segments.
- The side that sends the last ACK must remain in TIME_WAIT to retransmit it if necessary.
- TIME_WAIT is a feature, not a bug; suppressing it can cause protocol violations.
- RST is used for abrupt termination and error signaling but should be used judiciously.

5.10 Modern Extensions

TCP has served as the foundation of reliable Internet transport for over four decades. However, as applications evolved toward lower latency, mobility, and encrypted transport, new extensions and alternatives emerged to address TCP's limitations. Three notable developments are **QUIC (HTTP/3)**, **Multipath TCP (MPTCP)**, and **TCP Fast Open (TFO)**.

1. QUIC and HTTP/3. **QUIC** (Quick UDP Internet Connections) is a transport protocol originally developed by Google and later standardized by the IETF (RFC 9000). It operates in user space over UDP but provides TCP-like reliability, multiplexing, and encryption by design. HTTP/3, the latest version of the web protocol, runs exclusively over QUIC.

Key Features:

- **Connection Establishment:** Combines transport and cryptographic handshakes (TLS 1.3) into a single round-trip (1-RTT) or even 0-RTT for resumed connections.
- **Stream Multiplexing:** Multiple independent streams per connection—loss on one stream does not block others (solving TCP's "head-of-line blocking").
- **Connection Migration:** Connections survive IP or network changes (e.g., mobile handoff between Wi-Fi and LTE).
- **User-Space Evolution:** Runs over UDP, enabling fast protocol updates without kernel modifications.

Advantages:

- Lower latency for web applications.
- Built-in encryption and integrity (TLS 1.3 mandatory).
- Better performance on lossy or mobile links.

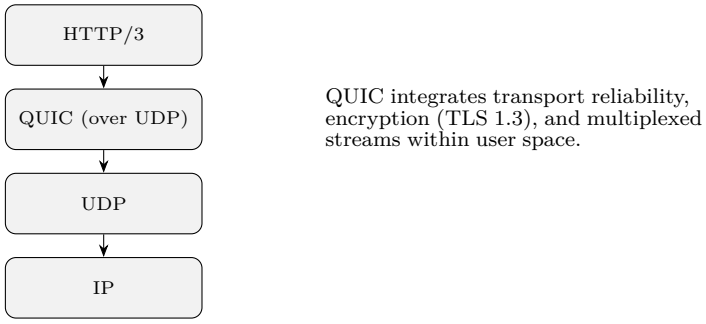


Figure 31: HTTP/3 over QUIC vs. traditional TCP + TLS + HTTP layering.

2. Multipath TCP (MPTCP). Traditional TCP uses a single path per connection, bound to specific source and destination IPs. **Multipath TCP (RFC 8684)** extends TCP to use multiple network interfaces (e.g., Wi-Fi, cellular, Ethernet) simultaneously within a single logical connection.

Core Mechanisms:

- A connection is divided into multiple **subflows**, each with its own TCP 4-tuple (IP/port pair).
- Endpoints coordinate subflows using the MPTCP options (MP_CAPABLE, MP_JOIN, etc.).
- A data sequence mapping layer reassembles packets across paths, maintaining a unified byte stream.

Benefits:

- Improved throughput via path aggregation (sum of available bandwidths).
- Seamless handover — traffic migrates if one link drops.
- Enhanced resilience for mobile or multipath environments.

Use Cases:

- iOS and Linux support MPTCP for applications like Siri and multipath VPNs.
- Data centers use MPTCP to exploit redundant paths.

3. TCP Fast Open (TFO). Standard TCP requires a 3-way handshake before any application data can be exchanged, adding one full RTT to the connection setup time. **TCP Fast Open (RFC 7413)** allows data to be sent in the initial SYN packet, reducing latency for short connections.

Mechanism:

1. During the first connection, the client and server perform a normal handshake. The server sends a **Fast Open Cookie** to the client.
2. On subsequent connections, the client includes the cookie in its SYN along with early data.
3. If the cookie validates, the server immediately delivers data to the application while completing the handshake.

Advantages:

- Reduces startup latency by up to 1 RTT for repeated connections.
- Especially beneficial for short-lived HTTP or API transactions.

Security Considerations:

- Cookies prevent IP spoofing, but early data can be replayed if not properly protected.
- Servers must handle partial or failed TFO handshakes safely.

Comparison Summary.

Extension	Primary Goal	Key Benefits
QUIC / HTTP/3	Reduce latency; multiplex streams	0-RTT setup, TLS 1.3 built-in, mobility
MPTCP	Path redundancy	Aggregated bandwidth, fault tolerance
TCP Fast Open	Reduce handshake delay	Early data on SYN; lower RTT cost

Key Insights.

- Modern extensions aim to reduce latency, improve mobility, and enhance resilience beyond classic TCP.
- QUIC rethinks transport entirely by combining reliability, security, and multiplexing in user space.
- MPTCP extends traditional TCP semantics to support simultaneous multi-interface usage.
- TCP Fast Open optimizes short-lived connection performance, especially for web and API traffic.

5.11 Quiz: Transport Layer Review

Transport Layer Quiz

1. Which of the following best characterizes the main property of the network and transport layers that is true for all protocols?

- Network layer provides addressing that is only locally unique; Transport layer provides addressing that is globally unique.
- Network layer provides best effort; Transport layer provides error correction.
- **Network layer provides host-to-host communication; Transport provides process-to-process.**
- Network layer uses numbers to represent addresses; Transport layer uses meaningful strings.

Explanation: The network layer moves packets between hosts; the transport layer identifies applications or processes within those hosts (via ports).

2. Which of the following is true?

- Both UDP and TCP are connection-oriented.
- UDP is connection-oriented; TCP is connectionless.
- **UDP is connectionless; TCP is connection-oriented.**
- Both UDP and TCP are connectionless.

Explanation: UDP sends datagrams with no connection setup; TCP establishes a reliable, stateful connection.

3. Which of the following could occur in an IP network that requires TCP to provide a mechanism for reliable communication? (select all that apply)

- **Error in the packet (bit flip)**
- An administrator may monitor the traffic
- **Congestion can occur in the switches/routers buffers**
- **Different packets can take different paths (even to the same destination)**

Explanation: TCP must handle corruption, reordering, and loss due to congestion or path variability.

4. Which header fields in TCP are key to reliable communication?

- Congestion Control
- **Sequence number and Acknowledgement number**
- BufferReservation flag and Allocation request fields during setup
- RequestForRetransmit flag

Explanation: TCP uses sequence and acknowledgement numbers to ensure reliable, in-order delivery and retransmission of lost segments.

5. Assume data is being transferred in one direction — from host A to host B. Under which condition is flow control needed?

- **A can send at 100 Gbps, B can receive at 1 Gbps**
- A uses an ISP with only 10 Mbps upload speed
- A can send at 1 Gbps, B can receive at 100 Gbps
- B uses an ISP with only 10 Mbps download speed

Explanation: Flow control protects a slower receiver's buffer from being overrun by a faster sender.

6. What is congestion collapse?

- Throughput gradually decreases as a link nears full capacity.
- **The goodput of a network decreases when the offered load increases past a certain point.**
- Routers explicitly signal to the sender that they are congested.
- Applications fail due to packet loss from congestion.

Explanation: When senders keep retransmitting lost packets during heavy congestion, the network spends capacity on useless traffic—reducing goodput.

7. In TCP's congestion control, what are feedback signals that congestion exists and the sender should reduce its rate? (select all that apply)

- **Receiving a duplicate acknowledgement**
- **Timeout waiting for an acknowledgement**
- Receiving a datagram requesting port number change
- Receiving a datagram with an in-order sequence number

Explanation: Duplicate ACKs and timeouts signal likely packet loss, prompting congestion window reduction.

8. What does the congestion window in TCP control?

- The rate at which the receiver delivers data to the application.
- The time of day during which the network is congested.
- The amount of time the sender waits before sending more packets.
- **How much data is allowed to be in flight at one time.**

Explanation: The congestion window limits the total unacknowledged data outstanding, effectively controlling TCP's sending rate.

6 Application Layer

6.1 Motivation and Problems

While the Transport Layer provides reliable process-to-process delivery, it remains agnostic to the *meaning* of data. The **Application Layer** sits at the top of the stack, where user-facing programs (browsers, mail clients, file transfer tools) interact with the network. Its protocols define how applications encode, identify, and exchange data over the Internet.

1. Why the Application Layer Exists. Each application has unique requirements that TCP or UDP alone cannot fulfill:

- **Naming:** Mapping human-friendly names (like `example.com`) to IP addresses.
- **Message Semantics:** Defining request/response formats (e.g., HTTP headers, email MIME boundaries).
- **Session Context:** Handling user authentication, cookies, state, and security.
- **Efficiency:** Adapting to latency, loss, and multiplexing in ways that depend on the application domain.

Thus, the Application Layer provides the abstractions needed for coherent, interoperable communication between distributed programs.

2. DNS vs. IP Addressing. At the network layer, communication requires IP addresses—numerical identifiers such as:

$$192.0.2.17 \quad \text{or} \quad 2001 : db8 :: 42$$

However, IP addresses are difficult for humans to remember and change frequently as servers move or scale.

The Domain Name System (DNS) provides a hierarchical, distributed naming service that maps human-readable domain names to IP addresses. Example resolution process:

1. A user enters `www.example.com`.
2. The client queries a recursive resolver.
3. The resolver traverses the hierarchy:
    - Root (`.`)
    - Top-Level Domain (`.com`)
    - Authoritative name server for `example.com`
4. The IP address is returned to the client and cached.

DNS decouples service identity from location, enabling load balancing, content distribution, and fault tolerance.

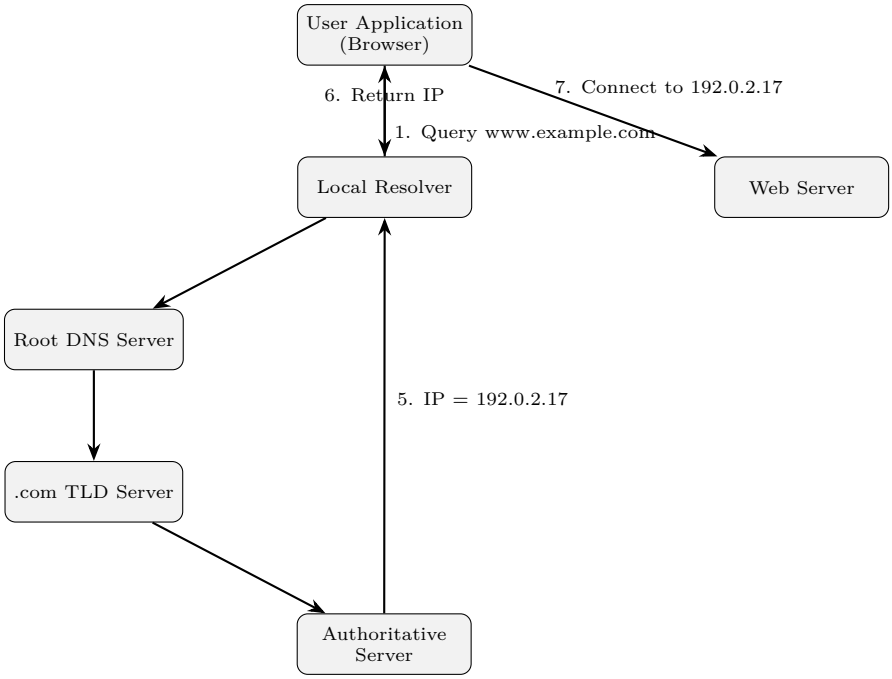


Figure 32: DNS resolution workflow leading to TCP connection establishment.

3. Socket API Abstraction. To make transport-layer functionality accessible to user programs, operating systems expose the **socket API**. Sockets provide a uniform programming interface for creating connections, sending data, and receiving responses.

Example workflow in pseudocode:

```
socket()          // Create endpoint
connect()         // Establish TCP connection
send(), recv()    // Transmit and receive bytes
close()           // Terminate connection
```

Key abstractions:

- Each socket represents a 5-tuple (protocol, local IP, local port, remote IP, remote port).
- Applications use hostnames rather than IPs—DNS resolution occurs transparently via library calls.
- The OS handles retransmissions, acknowledgments, and congestion control under the hood.

This API allows applications like browsers, mail clients, or SSH to function identically across platforms and networks.

4. Application-Level Framing. While TCP provides a byte stream, applications must decide how to delineate and interpret messages—this is the challenge of **application-level framing (ALF)**.

- Protocols like HTTP and SMTP define clear message boundaries using headers and delimiters (e.g., CRLF).
- Binary protocols (e.g., DNS, TLS) encode message length explicitly.
- Poor framing can lead to head-of-line blocking or security vulnerabilities (e.g., injection, truncation attacks).

Example:

```
HTTP Request:
GET /index.html HTTP/1.1
```

```
Host: example.com
User-Agent: curl/8.0
```

Applications that design framing carefully can handle partial reads, multiplex multiple messages, and optimize latency.

Key Insights.

- The Application Layer bridges human and network representations of communication.
- DNS abstracts naming, while the socket API abstracts connection and transport.
- Application-level framing defines how byte streams become meaningful messages.
- Together, these abstractions make the Internet scalable, human-usable, and protocol-agnostic.

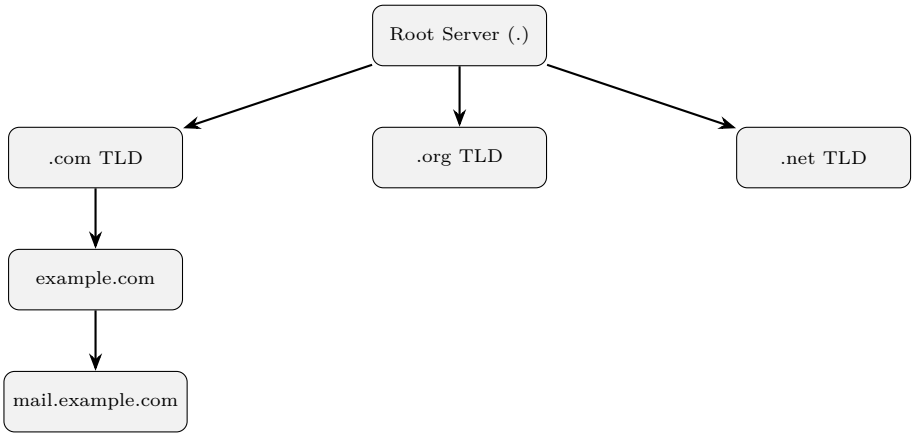
6.2 Domain Name System (DNS)

The **Domain Name System (DNS)** is the Internet's distributed database for translating human-readable domain names (e.g., `example.com`) into machine-usable IP addresses (e.g., `192.0.2.17`). It is one of the oldest and most fundamental application-layer protocols, designed for scalability, fault tolerance, and decentralization.

1. Hierarchical Namespace. DNS organizes names into a global **tree hierarchy**:

- The root domain (`.`) is at the top.
- Below it are **Top-Level Domains (TLDs)** such as `.com`, `.org`, `.edu`, country codes (`.uk`, `.jp`), and newer generic TLDs (`.io`, `.app`).
- Each domain can delegate responsibility to subdomains (e.g., `example.com` delegates `mail.example.com`).

The structure is mirrored by the hierarchy of **name servers** that store and serve DNS data. Each server knows only a portion of the namespace, enabling distributed control and fault isolation.



DNS hierarchy mirrors domain ownership and delegation. Each zone maintains authority over its subdomains.

Figure 33: Hierarchical structure of the DNS namespace.

2. Resolution Process. DNS resolution is typically recursive — a client queries a local resolver, which traverses the hierarchy on its behalf.

1. The client asks its configured resolver (often provided by the ISP or OS).
2. The resolver contacts a **root name server** for the TLD's location.
3. It queries the **TLD server** for the domain's authoritative server.
4. Finally, it queries the **authoritative server** for the requested record.
5. The answer is returned to the resolver, which caches it and forwards it to the client.

Each DNS response includes a **Time to Live (TTL)** value indicating how long the result can be cached.

3. Caching and Performance. Caching is essential to DNS scalability and latency reduction:

- **Stub resolvers** (on clients) cache results temporarily.
- **Recursive resolvers** (e.g., Google DNS, Cloudflare 1.1.1.1) maintain large caches to serve repeated queries efficiently.
- Expired records (TTL reached zero) trigger fresh lookups.

Negative caching (RFC 2308) stores information about failed lookups (e.g., non-existent domains) to reduce repeated queries.

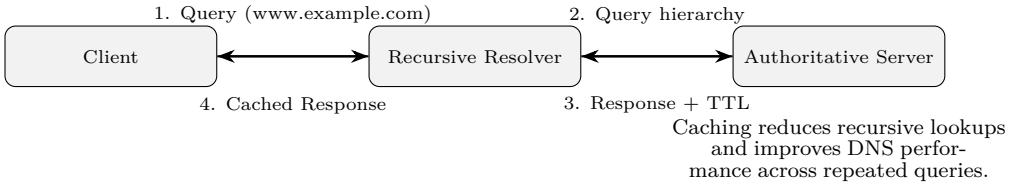


Figure 34: Recursive DNS resolution and caching behavior.

4. DNS Record Types. Each DNS record associates a name with a specific type of resource. Common record types include:

Record Type	Meaning	Example
A	IPv4 address mapping	`example.com → 192.0.2.1`
AAAA	IPv6 address mapping	`example.com → 2001:db8::1`
CNAME	Canonical name (alias)	`www → example.com`
MX	Mail exchange server	`10 mail.example.com`
NS	Authoritative name server	`ns1.example.com`
TXT	Arbitrary text data (e.g., SPF, DKIM)	`v=spf1 include:example.com`
SOA	Start of Authority (zone metadata)	Primary server, serial number, TTLs
PTR	Reverse lookup (IP → name)	`1.2.0.192.in-addr.arpa → example.com`
SRV	Service locator record	`_sip._tcp.example.com → host:port`

Each zone maintains an **SOA record** that defines administrative parameters — such as the primary name server, contact email, and zone serial number (for replication between primary and secondary servers).

5. Security and Integrity. Traditional DNS responses are unauthenticated and can be spoofed. **DNSSEC (DNS Security Extensions)** introduces digital signatures to verify authenticity and integrity:

- Each zone signs its records with a private key.
- Resolvers validate using public keys distributed through the hierarchy.

This prevents cache poisoning attacks and ensures trustworthy name resolution.

Key Insights.

- DNS provides scalable, distributed name resolution via hierarchical delegation.
- Caching is crucial for efficiency but governed by TTL-based freshness control.
- Multiple record types encode different resource relationships and services.
- DNSSEC enhances integrity and trust in modern deployments.

6.3 Socket Programming

Most Internet applications—web servers, email systems, chat platforms—are built around the **client–server model**. The **socket API** provides a uniform interface between applications and the transport layer (TCP or UDP). Through sockets, programs can establish connections, exchange data, and gracefully close sessions.

1. The Client–Server Paradigm. In the TCP/IP model:

- The **server** binds to a well-known port, listens for incoming connections, and handles requests.
- The **client** initiates a connection to the server's IP address and port, exchanges data, and closes the session.

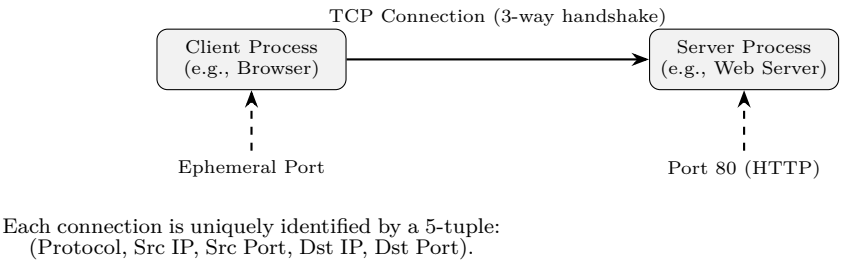


Figure 35: Client–server communication using sockets over TCP.

2. Socket Lifecycle. Each side of the connection uses a system-level file descriptor called a *socket*. The typical TCP socket workflow is:

Step	Client	Server
1	`socket()` — create endpoint	`socket()` — create endpoint
2	`connect()` — initiate connection	`bind()` — assign port/IP
3		`listen()` — wait for clients
4		`accept()` — accept connection
5	`send()`, `recv()` — data exchange	`send()`, `recv()` — data exchange
6	`close()` — terminate connection	`close()` — terminate connection

3. Minimal TCP Example (C). A simplified TCP echo example shows how sockets are used directly via system calls.

Listing 1: Simple TCP server in C

```c
// server.c
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in addr = {AF_INET, htons(8080), INADDR_ANY};
bind(sockfd, (struct sockaddr*)&addr, sizeof(addr));
listen(sockfd, 5);

int client = accept(sockfd, NULL, NULL);
char buf[1024];
int n = read(client, buf, sizeof(buf));
write(client, buf, n);      // echo back
close(client);
close(sockfd);
```

Listing 2: Simple TCP client in C

```c
// client.c
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
struct sockaddr_in srv = {AF_INET, htons(8080)};
inet_pton(AF_INET, "127.0.0.1", &srv.sin_addr);
connect(sockfd, (struct sockaddr*)&srv, sizeof(srv));

send(sockfd, "Hello", 5, 0);
char buf[1024];
int n = recv(sockfd, buf, sizeof(buf), 0);
printf("Server replied: %.*s\n", n, buf);
close(sockfd);
```

Notes:

- Both client and server use the same API calls; only connection direction differs.
- Servers often handle multiple clients concurrently using threads, forks, or event loops.
- TCP handles retransmission, flow control, and teardown automatically.

4. Python Socket Example. Python's `socket` library provides a higher-level but similar API:

Listing 3: Simple TCP echo server and client in Python

```
# server.py
import socket
s = socket.socket()
s.bind(('localhost', 8080))
s.listen(1)
conn, addr = s.accept()
print('Connected by', addr)
data = conn.recv(1024)
conn.sendall(data)
conn.close()

# client.py
import socket
s = socket.socket()
s.connect(('localhost', 8080))
s.sendall(b'Hello world')
print('Received:', s.recv(1024))
s.close()
```

5. UDP Example (Connectionless). UDP sockets are simpler—no handshake, no reliability, no streams:

Listing 4: Simple UDP echo server in Python

```
# udp_server.py
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('localhost', 9999))
while True:
data, addr = s.recvfrom(1024)
s.sendto(data.upper(), addr)
```

Listing 5: Simple UDP client in Python

```
# udp_client.py
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.sendto(b'ping', ('localhost', 9999))
data, _ = s.recvfrom(1024)
print('Server:', data.decode())
```

6. Application-Level Considerations.

- **Blocking vs Non-Blocking I/O:** By default, `recv()` waits for data. Non-blocking sockets or event loops (e.g., `select()`, `epoll()`, `asyncio`) improve scalability.
- **Serialization:** Real-world applications transmit structured data—JSON, Protobuf, ASN.1—rather than raw bytes.
- **Security:** TLS (via `ssl.wrap_socket()`) can encrypt sockets at the application layer.

Key Insights.

- Socket programming exposes the TCP/UDP abstraction to applications.
- The client–server model enables scalable, reusable services.
- Sockets unify networking APIs across languages and systems.
- Most modern protocols (HTTP, SSH, DNS, SMTP) build upon these same primitives.

6.4 Application Protocols

Application-layer protocols define the rules for exchanging structured information between networked programs. They build upon transport services (TCP or UDP) to provide semantics such as requests, replies, authentication, and session management. This section surveys several cornerstone protocols — HTTP, SMTP, DHCP, SSH — and concludes with modern web APIs built on REST and JSON.

1. HTTP: The Web's Core Protocol. The **Hypertext Transfer Protocol (HTTP)** underpins the World Wide Web. It follows a simple request–response model where a client (browser) sends a request and the server replies with a structured response.

Request Example (HTTP/1.1):

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: curl/8.0
Accept: text/html
```

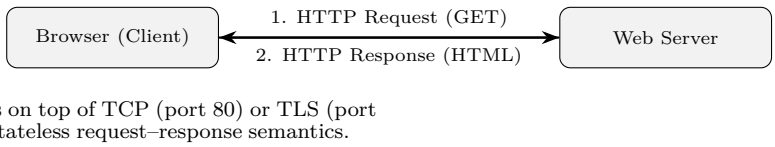
Response Example:

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 1256

<html> ... </html>
```

Key Features:

- **Statelessness:** Each request is independent; cookies and sessions emulate state.
- **Persistence:** HTTP/1.1 introduced keep-alive connections to reuse TCP sessions.
- **HTTPS:** Combines HTTP with TLS for encryption and authentication.
- **Evolution:** HTTP/2 multiplexes multiple streams per connection (binary framing), and HTTP/3 runs atop QUIC (UDP-based).



HTTP operates on top of TCP (port 80) or TLS (port 443), using stateless request–response semantics.

Figure 36: Basic HTTP client–server exchange.

2. SMTP: Sending Email. The **Simple Mail Transfer Protocol (SMTP)** (RFC 5321) handles outbound email delivery. Clients submit messages to a mail server, which relays them across the Internet using DNS `MX` records.

Example Session:

```
S: 220 mail.example.com ESMTP
C: HELO client.example.org
C: MAIL FROM:<alice@example.org>
C: RCPT TO:<bob@example.com>
C: DATA
C: Subject: Greetings
C:
C: Hello Bob!
C: .
S: 250 Message accepted
```

Characteristics:

- Text-based, line-oriented command protocol.
- Works over TCP port 25 (or port 587 for authenticated submission).
- Security via **STARTTLS** for opportunistic encryption.
- Relies on other protocols for retrieval — IMAP or POP3.

3. DHCP: Dynamic Host Configuration Protocol. **DHCP** (RFC 2131) automates IP address configuration for hosts joining a network. It runs over UDP (client port 68, server port 67) and enables plug-and-play connectivity.

Message Flow:

1. **DISCOVER:** Client broadcasts to locate DHCP servers.
2. **OFFER:** Server proposes an IP configuration (address, subnet, gateway, DNS).
3. **REQUEST:** Client requests one of the offered addresses.
4. **ACK:** Server confirms lease.

Leases include duration, renewal (T1, T2 timers), and optional configuration data. DHCP reduces manual setup and supports dynamic reassignment in LANs.

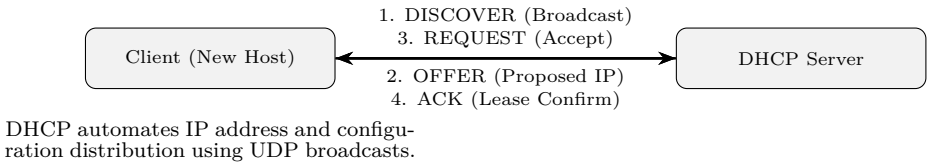


Figure 37: Simplified DHCP handshake.

4. SSH: Secure Shell. **SSH** (RFC 4251) provides encrypted, authenticated remote login and command execution. It replaces insecure tools like Telnet and rlogin.

Features:

- Uses TCP port 22.
- Employs asymmetric cryptography for authentication and key exchange.
- Encrypts the entire session (including terminal I/O and file transfers via SFTP/SCP).
- Supports tunneling and port forwarding for secure remote access.

Session Flow:

1. TCP connection established.
2. Protocol negotiation (version, algorithms).
3. Key exchange (Diffie–Hellman).
4. Authentication (password, key, or host-based).
5. Encrypted channel established.

5. Modern REST and JSON APIs. Most contemporary web and mobile applications communicate via **RESTful APIs** built atop HTTP. **Representational State Transfer (REST)** defines a uniform interface for resources identified by URLs and manipulated using HTTP verbs.

Typical REST Methods:

- `GET` — retrieve data.
- `POST` — create new resource.
- `PUT/PATCH` — update existing resource.
- `DELETE` — remove resource.

Example JSON Exchange:

```
Request:
POST /api/user HTTP/1.1
Host: api.example.com
Content-Type: application/json

{"name": "Alice", "email": "alice@example.com"}

Response:
HTTP/1.1 201 Created
Content-Type: application/json
```

```
{"id": 42, "status": "success"}
```

Key Ideas:

- **Statelessness:** Each request contains all context.
- **Format Negotiation:** Commonly JSON, XML, or Protobuf.
- **Security:** HTTPS for transport, tokens (OAuth2, JWT) for authentication.

Beyond REST: Modern systems may use alternatives like:

- **GraphQL:** Client-specified queries over HTTP.
- **gRPC:** Binary RPC protocol using HTTP/2 + Protobuf for efficiency.
- **WebSockets:** Persistent full-duplex channel for real-time apps.

Key Insights.

- Application protocols define semantic communication above transport services.
- HTTP dominates the modern Internet, extended by RESTful and encrypted variants.
- SMTP and DHCP illustrate classic control and data-plane protocols.
- SSH provides secure interactive sessions; REST and JSON enable interoperable APIs.
- Together, they represent the diversity of the Application Layer's mission: human, service, and machine communication.

6.5 Quiz: Application Layer Review

Application Layer Quiz

1. In DNS, how do DNS resolvers know about root servers?

- It uses ARP to query on its local area network.
- It queries DNS with the fully qualified domain name of the root servers (a.root-servers.net, b.root-servers.net, etc.)
- The don't – DNS resolvers don't know about root servers, which are run as a trusted and private Internet service only available to ISPs.
- **The IP addresses are hard coded within the resolver's code/configuration.**

Explanation: Root server IPs are built into resolver software (often in a "root hints" file) and updated occasionally.

2. In DNS, what happens when a DNS server does not know the IP address of the domain in the query?

- **Responds with the name of another resolver that it knows as authoritative for part of the domain.**
- Hangs the server.
- Responds with the IP address of the resolver.
- Silently discards the request.

Explanation: Recursive or iterative DNS resolution involves referring to other authoritative servers step by step until an answer is found.

3. In sockets programming, which would be an appropriate sequence of functions called?

- socket(), accept(), listen(), bind()
- bind(), send(), accept(), recv()
- **socket(), bind(), listen(), accept()**
- bind(), recv(), accept(), send()

Explanation: A typical server first creates a socket, binds it to an address, listens for incoming connections, and then accepts one.

4. True or false: socket programming is limited to being used in the C/C++ language.

- True
- **False**

Explanation: Many languages (Python, Java, Go, Rust, etc.) expose sockets APIs modeled after the C BSD-sockets interface.

5. Application protocol defines which of the following?

- Programming language of client and server
- **Message syntax and semantics**
- Message transmission time
- Amount of buffering in the Transport stack

Explanation: Application-layer protocols define how messages are structured and interpreted — not implementation details like timing or buffering.

6. For HTTP messages that use JSON, which of the following can be expressed (select all that apply)?

- **integers**
- **arrays**
- **objects**
- **strings**

Explanation: JSON supports numbers, strings, booleans, nulls, arrays, and objects (key–value pairs).

7. What role does the Protobufs compiler play in the gRPC workflow?

- It provides a language-neutral acceleration library for client/server applications.
- **It generates libraries in different languages that provide stubs for serializing and deserializing data that can be used by client and server software.**
- It transforms JSON data formatted messages into a binary form.
- It performs validation of JSON messages.

Explanation: The `protoc` compiler generates code in multiple languages from a `.proto` definition file, creating type-safe APIs for serialization.

8. Which is the biggest distinguishing difference between JSON and Protobufs?

- JSON is supported in many languages, protobufs are only supported in Go and Python.
- JSON is an open standard, protobufs require a paid license from Google to use.
- JSON is popular, protobufs are not.
- **JSON is textual, protobufs are binary.**

Explanation: JSON is human-readable and text-based; Protobufs are compact, binary, and schema-driven for efficiency.

7 TCP/IP in Operation

7.1 Packet Capture and Dissection

Understanding TCP/IP "in operation" requires observing how packets actually flow across the network. **Packet capture** tools such as **Wireshark** and `tcpdump` reveal the structure of live network traffic, showing headers, payloads, and timing relationships between layers.

1. Conceptual Overview. Every network interface (physical or virtual) transmits and receives Ethernet frames. A packet capture tool records these frames before they are processed by the operating system's networking stack. The captured data includes:

- Link-layer framing (Ethernet or Wi-Fi headers).
- Network-layer information (IP addresses, TTL, fragmentation).
- Transport-layer data (TCP/UDP ports, flags, sequence numbers).
- Application-layer payloads (HTTP requests, DNS queries, etc.).

Because of TCP/IP layering, each packet can be dissected hierarchically — each layer's header encapsulates the one above.

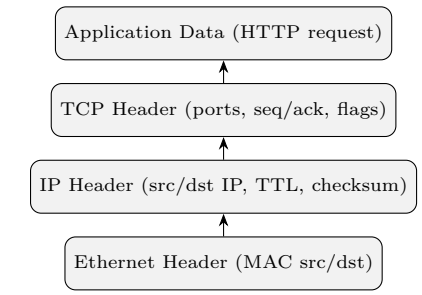


Figure 38: Layered dissection of a network packet.

2. Capturing Packets with Wireshark. **Wireshark** is a graphical packet analyzer that lets users observe live traffic or analyze saved capture files (`.pcap`, `.pcapng`).

Typical workflow:

1. Launch Wireshark and select an interface (e.g., `eth0`, `wlan0`).
2. Start a capture, then generate some traffic (e.g., load a web page or ping a host).
3. Stop the capture and apply filters such as:
   - `ip.addr == 8.8.8.8` — show packets to/from Google DNS.
   - `tcp.port == 80` — show HTTP over TCP.
   - `dns` — show only DNS queries and replies.
4. Click a packet to expand its layered dissection.

Each captured packet shows:

- **Frame** — timestamp, total length.
- **Ethernet** — source/destination MACs, EtherType.
- **IP** — source/destination addresses, TTL, protocol.
- **TCP/UDP** — port numbers, sequence/ack numbers, flags.
- **Application Data** — payload (HTTP headers, etc.).

3. Example: HTTP Request Capture. Suppose a client browses `http://example.com`. Wireshark's view might look like this (simplified):

```
No.   Time       Source        Destination   Protocol  Info
1     0.000000   192.168.1.10  93.184.216.34 TCP       54732 → 80 [SYN]
```

```
2     0.001202   93.184.216.34 192.168.1.10  TCP       80 → 54732 [SYN, ACK]
3     0.001458   192.168.1.10  93.184.216.34 TCP       54732 → 80 [ACK]
4     0.002850   192.168.1.10  93.184.216.34 HTTP      GET /index.html HTTP/1.1
5     0.004112   93.184.216.34 192.168.1.10  HTTP      HTTP/1.1 200 OK
6     0.004350   192.168.1.10  93.184.216.34 TCP       54732 → 80 [ACK]
```

Interpretation:

- Packets 1–3: Three-way TCP handshake (SYN, SYN–ACK, ACK).
- Packet 4: HTTP GET request sent by the client.
- Packet 5: HTTP response (status 200 OK).
- Packet 6: Final ACK confirming receipt.

This shows how protocols at multiple layers (TCP and HTTP) interact in real time.

4. Using Command-Line Tools. For text-based environments or scripting, `tcpdump` provides similar functionality:

```
sudo tcpdump -i eth0 -n tcp port 80
```

`tcpdump` output can be piped to Wireshark for later inspection:

```
sudo tcpdump -i eth0 -w capture.pcap
wireshark capture.pcap
```

5. Practical Analysis Tips.

- Use filters (display vs capture) to reduce noise and focus on specific flows.
- Follow TCP streams in Wireshark to reassemble bidirectional conversations.
- Check sequence and acknowledgment numbers to detect retransmissions or loss.
- Analyze timing graphs (RTT, throughput) for performance bottlenecks.
- Be cautious with privacy: packet captures may contain credentials or plaintext data.

Key Insights.

- Packet capture provides direct visibility into network behavior and protocol interaction.
- Wireshark enables hierarchical inspection—link, IP, TCP, and application layers.
- Capturing and dissecting traffic bridges theory (headers, encapsulation) with practice.
- Tools like `tcpdump` complement GUI analysis for automation or remote use.

7.2 Common Debugging Tools

Networking issues often manifest as timeouts, dropped packets, or unreachable hosts. To diagnose such problems, system administrators and engineers rely on a suite of low-level command-line tools that directly interact with the network stack. These tools provide visibility into reachability, routing paths, socket states, and live traffic.

1. `ping`: Basic Reachability Test. The simplest and most ubiquitous tool, `ping` uses the Internet Control Message Protocol (ICMP) to test whether a destination host is reachable and measure round-trip time (RTT).

```
ping www.example.com
PING www.example.com (93.184.216.34): 56 data bytes
64 bytes from 93.184.216.34: icmp_seq=0 ttl=56 time=11.2 ms
64 bytes from 93.184.216.34: icmp_seq=1 ttl=56 time=11.0 ms
--- www.example.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1002ms
rtt min/avg/max = 11.0/11.1/11.2 ms
```

Interpretation:

- Each line represents an ICMP Echo Reply.
- RTT indicates latency between sender and target.
- Packet loss suggests congestion or filtering.
- TTL (time-to-live) gives an indirect sense of hop count.

Usage Notes:

- `ping -c 5 <host>` — send exactly 5 pings.
- `ping -t <host>` (Windows) — continuous ping until stopped.
- If all packets are lost, the destination may be down or blocked by a firewall.

2. `traceroute`: Path Discovery. When connectivity is partial or slow, `traceroute` (Linux/macOS) or `tracert` (Windows) reveals the route packets take through intermediate routers.

How it works:

- Sends probe packets with gradually increasing TTL values.
- Each router decrements TTL by one; when it reaches zero, the router replies with an ICMP "Time Exceeded" message.
- The source uses these responses to map each hop along the path.

```
traceroute www.example.com
1  192.168.1.1      1.2 ms
2  10.12.0.1        8.5 ms
3  203.0.113.1     12.4 ms
4  93.184.216.34   15.0 ms
```

Interpretation:

- Each line lists one hop: IP, hostname (if resolvable), and RTT.
- "*" means no ICMP response (firewall or rate limit).
- Traceroute is invaluable for detecting routing loops or asymmetric paths.

3. `netstat`: Socket and Interface Status. The `netstat` command reports active connections, listening ports, routing tables, and interface statistics.

```
netstat -tuln
Proto Recv-Q Send-Q Local Address   Foreign Address  State
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN
tcp        0      0 192.168.1.10:54732 93.184.216.34:80 ESTABLISHED
udp        0      0 0.0.0.0:68      0.0.0.0:*
```

Common Flags:

- `-t` — TCP sockets
- `-u` — UDP sockets
- `-l` — Listening sockets
- `-n` — Numeric (don't resolve names)

Interpretation:

- Useful for verifying which services are running and on what ports.
- Helps detect stale connections or unexpected listeners.
- Network daemons (e.g., SSH, HTTP) typically appear in LISTEN state.

4. `tcpdump`: Packet Capture from CLI. `tcpdump` is the command-line equivalent of Wireshark for text-based or remote debugging.

```
sudo tcpdump -i eth0 -n host 93.184.216.34 and port 80
tcpdump: listening on eth0, link-type EN10MB (Ethernet)
12:21:33.511127 IP 192.168.1.10.54732 > 93.184.216.34.80: Flags [S]
12:21:33.512920 IP 93.184.216.34.80 > 192.168.1.10.54732: Flags [S.], seq 0
12:21:33.513180 IP 192.168.1.10.54732 > 93.184.216.34.80: Flags [.], ack 1
```

Usage:

- `tcpdump -i any -n tcp port 80` — capture HTTP traffic on all interfaces.
- `tcpdump -w capture.pcap` — save output for later analysis in Wireshark.
- `tcpdump -A port 25` — print payload as ASCII (useful for SMTP).

Interpretation:

- Displays real-time packet headers.
- Timestamps and flags (SYN, ACK, FIN) reveal connection behavior.
- Great for low-level debugging of TCP handshakes and retransmissions.

5. `ss`: Modern Replacement for `netstat`. **ss** (Socket Statistics) is a faster, more modern utility included with Linux's `iproute2` suite.

```
ss -tuna
Netid State  Recv-Q Send-Q Local Address:Port  Peer Address:Port
tcp   LISTEN 0      128    0.0.0.0:22          0.0.0.0:*
tcp   ESTAB  0      0      192.168.1.10:54732  93.184.216.34:80
```

Advantages over `netstat`:

- Provides socket states directly from kernel data structures.
- Supports advanced filters (e.g., by process, user, or connection state).
- Much faster on systems with thousands of connections.

Key Insights.

- **ping** and **traceroute** verify reachability and routing.
- **netstat** and **ss** inspect local socket states and services.
- **tcpdump** captures packet-level details for in-depth analysis.
- These tools form the foundation of network debugging and performance monitoring.

7.3 End-to-End Example

To bring together all the ideas of the TCP/IP model, let's trace what happens when a web browser on a client machine requests a web page using HTTP.

1. Scenario. A user enters `http://example.com` in their browser. The browser must retrieve `/index.html` from the web server at IP address `93.184.216.34`.

1. The browser formulates an **HTTP GET request**.
2. The HTTP message is sent via a **TCP socket** (port 80).
3. TCP segments the data and adds reliability information.
4. IP encapsulates the TCP segment in an IP packet for routing.
5. The Link layer frames the packet into an Ethernet frame for transmission.

2. Application Layer (HTTP).

```
GET /index.html HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0
Accept: text/html
```

Role: Defines the actual application data being sent — the HTTP request.

3. Transport Layer (TCP). TCP Header (20 bytes typical):

- Source port: 54732
- Destination port: 80
- Sequence number: 1001
- Acknowledgment number: 5001
- Flags: ACK, PSH
- Window size: 64240

Purpose:

- Ensures reliable delivery and ordering.
- Identifies the conversation via port numbers.

4. Internet Layer (IP). IPv4 Header Fields:

- Source IP: 192.168.1.10
- Destination IP: 93.184.216.34
- TTL: 64
- Protocol: 6 (TCP)
- Header checksum: 0x7b2c

Purpose:

- Provides logical addressing and routing.
- Independent of the underlying network technology.

5. Link Layer (Ethernet). Ethernet Frame Header:

- Destination MAC: 00:11:22:33:44:55
- Source MAC: 66:77:88:99:AA:BB
- EtherType: 0x0800 (IPv4)

The frame is transmitted as a stream of bits over the physical medium.

6. Encapsulation Summary. Each layer wraps the higher layer's data within its own header:

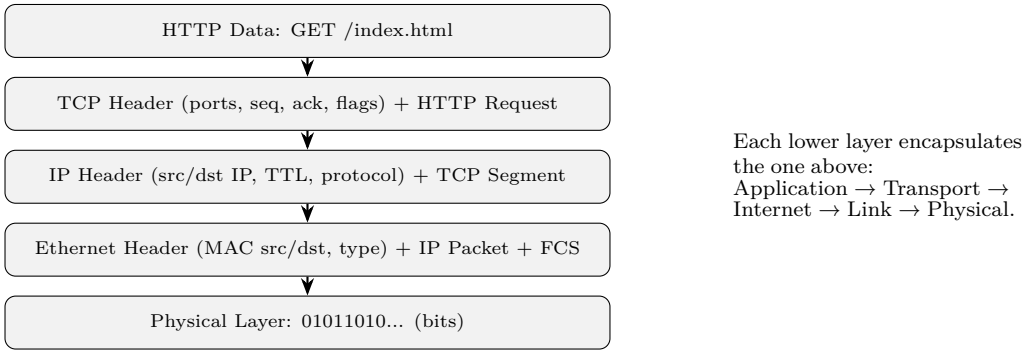


Figure 39: Encapsulation of an HTTP GET through the TCP/IP stack.

7. Decapsulation at the Server. At the destination:

1. The Ethernet driver strips the frame header.
2. The IP layer verifies checksum and TTL, passes payload to TCP.
3. TCP reassembles the stream, acknowledges receipt.
4. The web server receives the HTTP request and responds.

8. Response Path. The process reverses symmetrically:

- The server sends an HTTP 200 OK response over the same TCP connection.
- Each layer adds its own header again.
- The client's TCP stack confirms receipt via ACKs.

Key Insights.

- Every Internet communication follows this encapsulation pattern.
- Each protocol layer adds metadata required for its function (addressing, reliability, framing).
- Encapsulation and decapsulation guarantee modularity and interoperability across systems.

8 Network Security

8.1 Motivation and Security Properties

The openness that made the Internet scalable and interoperable also made it vulnerable. TCP/IP was originally designed for a cooperative academic environment—trust was implicit, and threats were minimal. Today, however, the same global reach exposes every connected device to potential attack, interception, or impersonation.

1. The Need for Security. Modern networks transmit sensitive data: passwords, banking transactions, corporate communications, and personal identifiers. Without security mechanisms, adversaries can:

- **Eavesdrop** on unencrypted traffic (sniffing credentials or private messages).
- **Modify** packets in transit (man-in-the-middle or injection attacks).
- **Impersonate** trusted hosts or users (spoofing, phishing, credential theft).
- **Disrupt** availability (denial-of-service, flooding, or routing attacks).

The goal of network security is therefore to ensure that data remains safe, authentic, and available even when traversing untrusted networks.

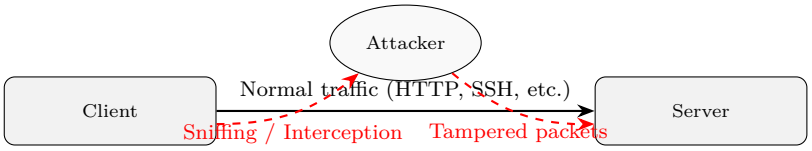


Figure 40: Example threat: interception or modification of data in transit.

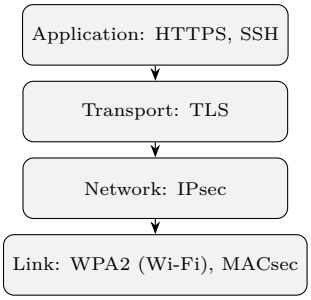
2. Core Security Properties. Network security aims to preserve several key properties often summarized as the **CIA triad**, along with extensions relevant to distributed systems.

- **Confidentiality:** Only authorized parties can read the data. Achieved through *encryption* (e.g., TLS, IPsec, SSH).
- **Integrity:** Data cannot be altered undetectably in transit. Ensured through *checksums, hashes, and message authentication codes (MACs).*
- **Availability:** Services remain accessible and functional when needed. Protected through redundancy, rate limiting, and DoS mitigation.
- **Authentication:** Participants can verify each other's identities. Implemented via passwords, certificates, or cryptographic handshakes.
- **Authorization:** Even authenticated users can only perform permitted actions.
- **Non-repudiation:** A sender cannot later deny having sent a valid message; relies on *digital signatures.*

3. Security in the TCP/IP Context. While TCP/IP's core design did not include built-in security, later extensions introduced cryptographic protections:

- **TLS/SSL** at the transport layer secures applications like HTTPS and IMAPS.
- **IPsec** provides encryption and authentication at the network layer.
- **SSH** combines both transport and application layer protections.

Security is thus layered—mirroring the architecture of TCP/IP itself.



Security mechanisms can exist at multiple layers:
- Link-layer encryption (WPA2)
- Network-layer (IPsec)
- Transport-layer (TLS)
- Application-layer (SSH, HTTPS)

Figure 41: Security can be implemented at multiple layers of the protocol stack.

4. Balancing Usability and Security. Security introduces computational and operational overhead: encryption consumes CPU, authentication adds latency, and strong policies can reduce convenience. The art of network security lies in finding a balance between:

- Performance and protection.
- Openness and control.
- Automation and verification.

Key Insights.

- Modern networking must assume adversarial conditions.
- Security properties—confidentiality, integrity, availability, authentication, authorization—form the backbone of trustworthy communication.
- Each layer of TCP/IP can contribute to overall security.
- The challenge is not only *achieving* these properties but doing so without sacrificing scalability or usability.

8.2 Cryptography Basics

Cryptography provides the mathematical foundation for nearly all network security mechanisms. Its primary goal is to transform readable data (*plaintext*) into an unreadable form (*ciphertext*) such that only authorized parties can recover the original information.

1. Symmetric-Key Cryptography. In a **symmetric cryptosystem**, the same secret key is used for both encryption and decryption. This model is efficient and forms the backbone of most bulk data encryption methods.

Examples:

- **AES (Advanced Encryption Standard):** The most widely used symmetric cipher, supporting 128-, 192-, and 256-bit keys.
- **3DES (Triple DES):** An older algorithm applying DES three times to strengthen security.
- **ChaCha20:** A modern stream cipher optimized for software performance.

Operation:

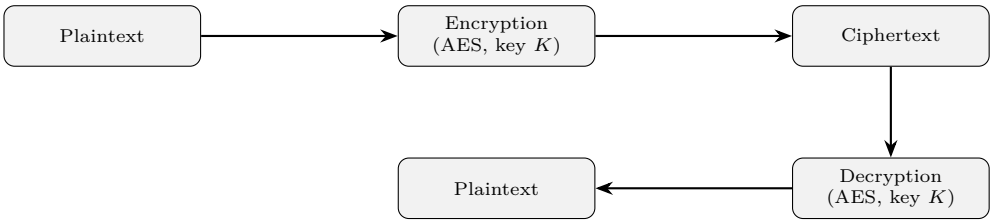


Figure 42: Symmetric encryption: the same secret key is used for both directions.

Pros:

- High performance and suitable for encrypting large data volumes.

- Simple key management in small systems.

Cons:

- Key distribution is a major challenge—how do two parties share a secret key securely?

2. Asymmetric (Public-Key) Cryptography. An **asymmetric cryptosystem** uses two mathematically linked keys:

- A **public key**, which can be widely distributed.
- A **private key**, which must be kept secret.

Use Cases:

- Encrypt with the recipient's public key → ensures *confidentiality*.
- Sign with the sender's private key → ensures *authenticity and integrity*.

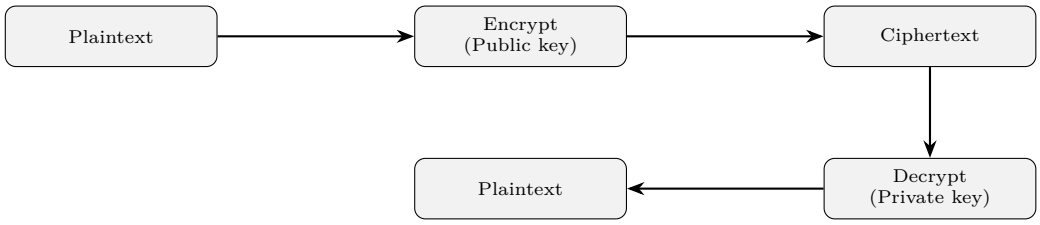


Figure 43: Asymmetric encryption: distinct public and private keys.

Examples:

- **RSA (Rivest–Shamir–Adleman):** Relies on the difficulty of factoring large integers.
- **Elliptic Curve Cryptography (ECC):** Provides comparable security with shorter keys.
- **Diffie–Hellman Key Exchange:** Enables two parties to derive a shared secret over an insecure channel.

3. Digital Signatures. Public-key systems can be used "in reverse" to provide authentication and non-repudiation:

- Sender signs a message with their private key.
- Anyone with the sender's public key can verify the signature.

This guarantees that:

1. The message came from the stated sender (authenticity).
2. The content has not been altered (integrity).
3. The sender cannot deny authorship later (non-repudiation).

4. Hash Functions. A **cryptographic hash function** maps data of arbitrary length to a fixed-size digest:

$$h = H(m)$$

where H is a one-way, collision-resistant function (e.g., SHA-256). Hashes are central to integrity checks, password storage, and digital signatures.

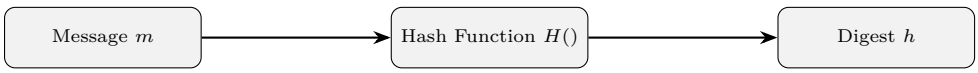


Figure 44: Hashing produces a fixed-size digest verifying message integrity.

5. Hybrid Cryptosystems. In practice, most real-world systems combine both types of cryptography:

- Use asymmetric cryptography to exchange a session key.
- Use symmetric encryption (e.g., AES) for bulk data.

Example: TLS uses RSA or Diffie–Hellman for key exchange, then encrypts the session using AES or ChaCha20.

Key Insights.

- Symmetric encryption is fast but needs secure key sharing.
- Asymmetric encryption solves key distribution but is computationally expensive.
- Hashes and signatures provide integrity and authenticity.
- Hybrid cryptosystems combine their strengths for efficient, secure communication.

8.3 Integrity and Authentication

After establishing confidentiality through encryption, the next two pillars of secure communication are **integrity** and **authentication**. Integrity ensures that data has not been altered in transit, while authentication confirms that the message genuinely originates from the claimed sender.

1. The Need for Integrity Protection. Encryption alone does not guarantee message integrity. An attacker who cannot read encrypted data may still be able to modify it — flipping bits or reordering packets — unless the receiver can verify that the data is unchanged. Integrity mechanisms use cryptographic digests or message authentication codes (MACs) to detect such tampering.

2. Message Authentication Codes (MACs). A **Message Authentication Code** is a short cryptographic tag appended to a message to ensure both integrity and authenticity. Unlike plain hashes, a MAC requires a secret key shared between sender and receiver:

$$t = \mathrm{MAC}_K(M)$$

where M is the message and K is the secret key. When the receiver recomputes $\mathrm{MAC}_K(M)$ and compares it to t, any mismatch indicates tampering.

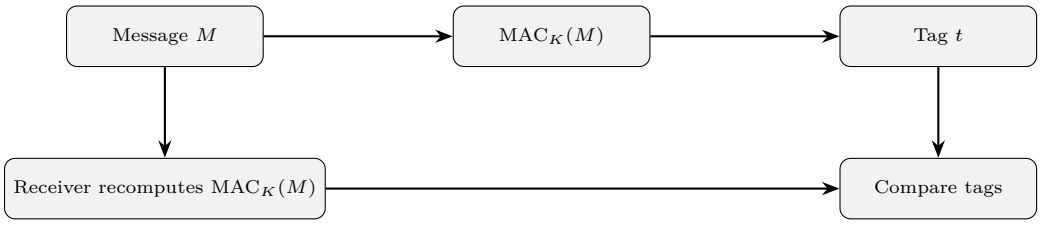


Figure 45: MAC generation and verification using a shared key.

3. HMAC: Keyed Hash Construction. A widely used form of MAC is the **HMAC** (Hash-based Message Authentication Code). It builds on an underlying hash function (e.g., SHA-256) and mixes the key with the message twice for security:

$$\mathrm{HMAC}_H(K, M) = H((K \oplus \mathrm{opad}) \,||\, H((K \oplus \mathrm{ipad}) \,||\, M))$$

where opad and ipad are fixed padding constants.

Properties:

- Resistant to length-extension and forgery attacks.
- Used extensively in TLS, IPsec, and SSH for integrity verification.
- Fast and easy to implement on arbitrary-length messages.

4. CMAC and GMAC. Alternative MACs use block ciphers instead of hash functions:

- **CMAC:** Cipher-based MAC using AES or 3DES for integrity checking.
- **GMAC:** Derived from AES in Galois/Counter Mode (GCM), commonly used in TLS 1.3.

These variants are suited to environments where block ciphers are already deployed for encryption and hardware acceleration is available.

5. Digital Signatures and Authentication. While MACs authenticate messages between two parties sharing a secret key, **digital signatures** use asymmetric cryptography so that anyone can verify authenticity without sharing secrets.

Process:

1. The sender computes a hash of the message $h = H(M)$.
2. The sender signs the hash with their private key to produce $s = \text{Sign}_{K_{priv}}(h)$.
3. The receiver verifies the signature using the sender's public key.

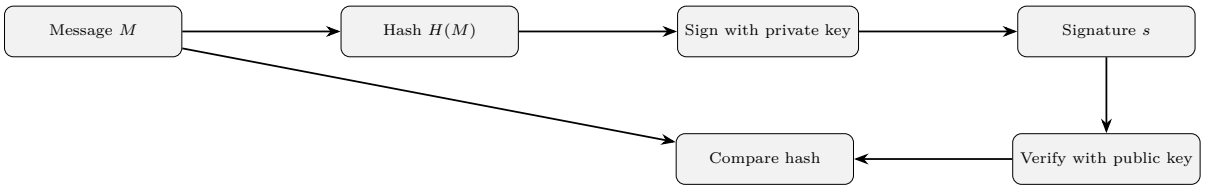


Figure 46: Digital signature generation and verification process.

Properties:

- Only the private key holder can generate valid signatures.
- Public keys enable universal verification (no shared secrets).
- Supports *non-repudiation*—a sender cannot deny authorship.

6. Putting It Together: Integrity + Authentication in Protocols. Real-world secure protocols often combine both symmetric and asymmetric approaches:

- **TLS:** Uses digital certificates for authentication and HMAC for data integrity.
- **IPsec:** Employs HMAC-SHA256 for packet authentication.
- **SSH:** Authenticates users via digital signatures and protects sessions with MACs.

Key Insights.

- Integrity ensures that "what was sent is what was received."
- Authentication ensures that "who sent it is who they claim to be."
- MACs protect shared-key systems; signatures protect public-key systems.
- Modern Internet protocols combine these techniques to secure every communication layer.

8.4 Security Protocols

Modern Internet security is enforced not by a single mechanism but by a layered set of protocols, each designed to secure data at different stages of its transmission. Three of the most important standards are IPsec, RPKI, and TLS/HTTPS.

1. IPsec (Internet Protocol Security). IPsec secures network-layer communication by adding authentication, integrity, and encryption directly to IP packets. It operates in two main modes:

- **Transport Mode:** Protects only the payload of the IP packet; typically used for end-to-end host communication.
- **Tunnel Mode:** Protects the entire packet by encapsulating it inside a new IP packet; used for VPNs and gateway-to-gateway communication.

Key Components:

- **AH (Authentication Header):** Provides authentication and integrity by adding a header with a keyed hash.
- **ESP (Encapsulating Security Payload):** Adds confidentiality through encryption and optional authentication.
- **IKE (Internet Key Exchange):** A control protocol that negotiates cryptographic keys and security associations (SAs).

Figure 47: IPsec encapsulates and secures IP packets at the network layer.

Operation: Two peers use IKE to establish an SA, agreeing on algorithms (AES, HMAC-SHA256, etc.) and keys. Subsequent IP traffic between them is protected by AH or ESP headers, ensuring that even routers in between cannot inspect or modify the payload.

Applications:

- Site-to-site VPNs (corporate WANs).
- Secure tunnels between gateways or firewalls.
- Host-to-host encryption in sensitive environments.

2. RPKI (Resource Public Key Infrastructure). **RPKI** secures the Internet's routing system by preventing prefix hijacking — a common attack where malicious or misconfigured routers announce IP address ranges they do not own. It provides a cryptographic way to verify that an AS (Autonomous System) is authorized to originate specific IP prefixes.

Core Mechanism:

- Regional Internet Registries (RIRs) issue **Route Origin Authorizations (ROAs)**.
- Each ROA binds an AS number to a set of IP prefixes, signed using the RIR's private key.
- Routers use RPKI-aware software to validate BGP updates against these ROAs.

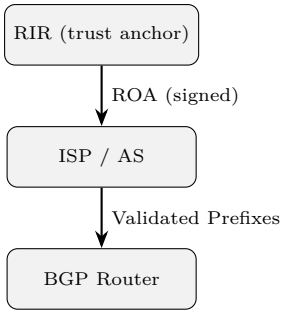


Figure 48: RPKI validation chain for route origin authorization.

Impact:

- Reduces accidental and malicious BGP hijacks.
- Adds accountability to prefix ownership.
- Increasingly deployed across major ISPs and CDNs.

3. TLS / HTTPS (Transport Layer Security). While IPsec secures packets and RPKI secures routing, **TLS** protects the application layer. Operating atop TCP, it provides confidentiality, integrity, and authentication for end-user applications such as HTTPS, SMTP, and IMAP.

TLS Structure:

- **Record Layer:** Handles fragmentation, compression, encryption, and message authentication.

- **Handshake Layer:** Negotiates cryptographic parameters, authenticates endpoints, and establishes session keys.

TLS Handshake Overview:

1. **ClientHello:** Client proposes cipher suites, TLS version, and random nonce.
2. **ServerHello:** Server selects parameters and sends its certificate.
3. **Key Exchange:** Server and client establish a shared session key (RSA or Diffie–Hellman).
4. **ChangeCipherSpec:** Both sides switch to the new encryption state.
5. **Finished:** Each endpoint verifies the handshake integrity.

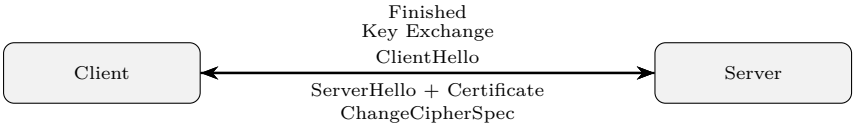


Figure 49: Simplified TLS handshake sequence.

HTTPS Integration:

- HTTPS = HTTP + TLS over TCP port 443.
- Certificates are validated against trusted Certificate Authorities (CAs).
- TLS 1.3 reduces handshake round trips and enforces forward secrecy.

Comparison Summary:

Protocol	Layer	Purpose
IPsec	Network	Encrypts and authenticates IP packets
RPKI	Control Plane	Verifies BGP route authenticity
TLS/HTTPS	Transport / Application	Secures end-to-end user sessions

Key Insights.

- IPsec secures the *path* between endpoints.
- RPKI secures the *routing infrastructure*.
- TLS/HTTPS secures the *applications and data*.
- Combined, they create defense-in-depth across the Internet stack.

8.5 Practical Implementations

While theoretical mechanisms such as IPsec, TLS, and authentication provide the cryptographic foundation of security, real-world protection depends on how these mechanisms are deployed and managed within networks. The three most prevalent implementation domains are firewalls, Virtual Private Networks (VPNs), and zero-trust architectures.

1. Firewalls: Gatekeepers of Network Traffic. A **firewall** is a device or software service that monitors and filters network traffic according to a defined set of rules. Firewalls enforce network segmentation and serve as the first line of defense between trusted and untrusted networks.

Main Types:

- **Packet-Filtering Firewalls:** Inspect headers at the network and transport layers (IP, TCP/UDP). Rules are typically based on source/destination addresses, ports, and protocol type.
- **Stateful Inspection Firewalls:** Track active connections and allow return traffic only for valid sessions.
- **Application Firewalls (Proxies):** Examine payloads at the application layer (e.g., HTTP, DNS) to block malicious requests.
- **Next-Generation Firewalls (NGFW):** Integrate intrusion prevention, TLS decryption, and machine-learning-based threat detection.

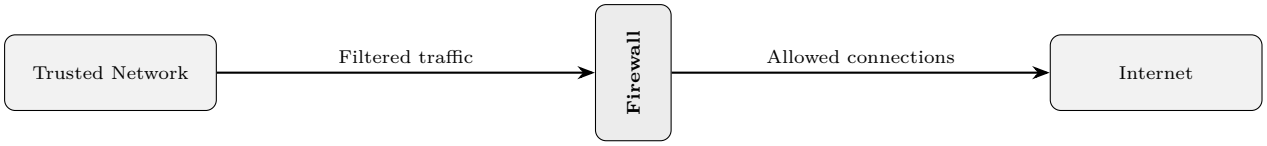


Figure 50: A firewall filtering traffic between a trusted LAN and the Internet.

Key Concepts:

- Default-deny policies are safest: only explicitly permitted traffic passes.
- Firewalls often implement Network Address Translation (NAT) to mask internal IPs.
- Logs and alerts provide early detection of scanning or intrusion attempts.

2. Virtual Private Networks (VPNs). A **VPN** uses tunneling and encryption (often via IPsec or TLS) to create a secure channel between distant nodes over an untrusted network. It ensures confidentiality, integrity, and authentication between the endpoints.

VPN Types:

- **Site-to-Site VPN:** Connects two entire networks (e.g., branch offices) through gateway devices.
- **Remote Access VPN:** Allows individual users to securely access a private network from the Internet.
- **SSL/TLS VPN:** Operates over HTTPS, requiring only a web browser or lightweight client.

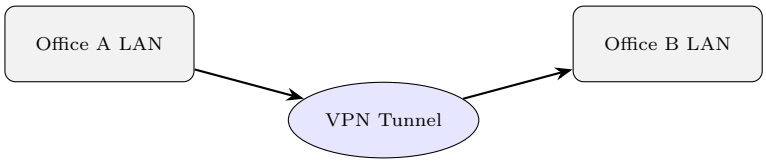


Figure 51: A site-to-site VPN providing encrypted communication across the Internet.

Key Benefits:

- Protects data over public networks.
- Enables remote workforce connectivity.
- Provides segmentation and isolation for sensitive traffic.

Limitations:

- VPN gateways can become bottlenecks or single points of failure.
- Credentials and certificates must be properly managed.
- Does not inherently restrict lateral movement once connected.

3. Zero-Trust Networking. Traditional network security assumes a "trusted inside, untrusted outside" perimeter model. However, modern environments (cloud, BYOD, remote access) have dissolved this boundary. **Zero-trust networking** replaces perimeter-based security with continuous verification and least-privilege access at every step.

Core Principles:

- **Verify Explicitly:** Authenticate and authorize every connection—user, device, or application.
- **Least Privilege:** Grant only the minimal access necessary for the task.
- **Assume Breach:** Design systems as though attackers are already inside.

Implementation Techniques:

- Identity-aware proxies and Single Sign-On (SSO).
- Continuous endpoint posture assessment.
- Microsegmentation—enforcing access policies between workloads.
- Cloud-native access brokers (e.g., BeyondCorp, ZTNA gateways).

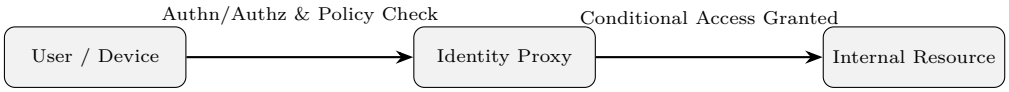


Figure 52: Zero-trust model: every access requires verification and policy enforcement.

Benefits:

- Reduces insider threat and lateral movement.
- Adapts to hybrid cloud and mobile environments.
- Complements traditional firewalls and VPNs rather than replacing them.

Key Insights.

- Firewalls enforce perimeters and traffic policy.
- VPNs secure data-in-transit across untrusted links.
- Zero-trust models redefine security around identity and verification rather than location.
- Together, they form the operational backbone of modern network security.

8.6 Quiz: Network Security Review

Network Security Quiz

1. What are general negatives about symmetric and asymmetric encryption respectively?

- Symmetric encryption is slow; asymmetric encryption requires shared secret keys.
- Symmetric encryption is vulnerable to eavesdropping; asymmetric encryption requires parts of the message to be plaintext.
- **Symmetric encryption requires shared secret keys; asymmetric encryption is slow.**
- Software is generally not available that can perform either symmetric or asymmetric encryption.

Explanation: Symmetric encryption demands a shared key distribution method; asymmetric encryption is computationally heavier but solves the key exchange issue.

2. What mechanism of IPsec keeps communications confidential?

- **Encryption**
- Digital Certificates
- Message Authentication Codes
- Hashing

Explanation: IPsec uses encryption (e.g., AES, 3DES) to maintain confidentiality. MACs and hashing ensure integrity, not secrecy.

3. What is the purpose of the Diffie–Hellman protocol as used in IPsec?

- Exchange security parameters
- **Exchange a shared symmetric key**
- Ensure the network is performant enough for data exchange
- Distribute a public key of a server

Explanation: Diffie–Hellman securely derives a shared symmetric session key over an insecure channel without transmitting the key itself.

4. What is an attack that RPKI protects against?

- Man in the middle attacks

- Eavesdropping
- **Route hijacking**
- Denial of Service

Explanation: RPKI (Resource Public Key Infrastructure) ensures that ASes are authorized to advertise specific IP prefixes, preventing route hijacks.

5. What entities serve as the main certificate authorities in RPKI?

- Network vendors such as Cisco, Juniper, Palo Alto
- Universities
- A government agency
- **Regional Internet Registries**

Explanation: RIRs (ARIN, RIPE NCC, APNIC, AFRINIC, LACNIC) act as the trust anchors that issue digital certificates binding IP resources to ASes.

6. What are three concerns with HTTP that HTTPS helps address?

- Data encoding, data size, and data fidelity
- **Traffic is in plaintext; data has no integrity protection; client can't be sure server is their intended communication partner.**
- DNS requests are UDP, data is encoded inefficiently, APIs are openly accessible
- Traffic is in plaintext, data has no integrity protection, the TCP three-way handshake is insecure

Explanation: HTTPS adds encryption (confidentiality), integrity (via MACs), and authentication (via certificates) to HTTP traffic.

7. What mechanism does Let's Encrypt use to validate the identity of a request for a signed digital certificate?

- An admin from Let's Encrypt sets up a video call with the requester.
- **Admin places a token at a known address on their website, and if Let's Encrypt is able to fetch it, it proves ownership.**
- The Let's Encrypt team performs a background check on the requester.
- Let's Encrypt accepts all requests since it's only for testing HTTPS locally.

Explanation: Let's Encrypt uses automated domain validation (ACME protocol): proving control by serving a token at a specific path or DNS record.

Quiz — Networking and Security Fundamentals

Question 1: Given four hosts (A, B, C, D) connected to a switch with MACs AA:..., BB:..., CC:..., DD:..., and a MAC table with entries for A→Port0, B→Port1, C→Port2, which node has A **not** received a response from?

(a) B

(b) C

(c) D

(d) Can't tell from information provided

Question 2: How does cyclic redundancy check (CRC) differ from parity (single parity bit)? (select all that apply)

(a) ☐ CRC is not used in practice, parity is widely used

(b) ☐ CRC can detect multiple errors, parity can only detect one error

(c) ☐ CRC can only detect errors, parity can also correct them

(d) ☐ CRC is fast to calculate, parity calculations are computationally intensive

Question 3: Which of the following Multiple Access Protocols use channel partitioning? (select all that apply)

(a) ☐ CSMA/CD

(b) ☐ CSMA

(c) ☐ CDMA

(d) ☐ TDMA

Question 4: Loops are bad because they circulate the same traffic over the same links. Which of the following mitigate loops? (select all that apply)

(a) ☐ Time to live field

(b) ☐ Acknowledgement from receiver

(c) ☐ Spanning Tree Protocol

(d) ☐ The redirect option in HTTP

Question 5: Differentiate between link layer addressing and network layer addressing.

(a) Link layer addressing is device-specific, network layer addressing is location-specific.

(b) Link layer addressing is always 48 bits, network layer addressing is always 32 bits.

(c) Link layer addressing is only used in wired networks, network layer addressing is for wireless networks.

Question 6: True or False. In the Internet Protocol, the forwarding table needs one entry per device on the network.

(a) True

(b) False

Question 7: True or False. In Ethernet, a learning switch holds one entry per device it knows about.

(a) True

(b) False

Question 8: Which protocol dynamically assigns IP addresses to devices?

(a) DHCP

(b) ARP

(c) DNS

(d) ICMP

Question 9: Which routing protocol selects the path with the fewest routers between source and destination?

(a) OSPF

(b) DNS

(c) BGP

(d) SHORT

Question 10: Why is IP considered best effort? (select all that apply)

(a) ☐ Timing (packets may get delayed)

(b) ☐ Content (packets may contain other content)

(c) ☐ Order (packets may get re-ordered)

(d) ☐ Delivery (packets may get dropped)

Question 11: The network layer must compute paths and determine output ports. Which pairing is correct?

(a) Computes paths (Data Plane), determines port (Control Plane)

(b) Computes paths (Data Plane), determines port (Data Plane)

(c) Computes paths (Control Plane), determines port (Data Plane)

(d) Computes paths (Control Plane), determines port (Control Plane)

Question 12: Given routes (1.2.2.0/23 → 11.11.11.11) and (1.2.0.0/20 → 22.22.22.22), for packet 1.2.3.4 which next hop is used?

(a) 11.11.11.11

(b) 22.22.22.22

(c) neither

(d) both

Question 13: True or False. In BGP a router learns about all routes known to its neighbor.

(a) true

(b) false

Question 14: True or False. In OSPF a router learns about all routes known to its neighbor.

(a) true

(b) false

Question 15: To monitor reachability at 1-second intervals while changing a network, use:

(a) traceroute

(b) wget

(c) ping

(d) iperf3

Question 16: If throughput/latency changed due to routing, which tool tests that hypothesis?

(a) traceroute

(b) wget

(c) ping

(d) iperf3

Question 17: What is the role of port numbers in transport protocols?

(a) Direct traffic to the correct network interface card

(b) Direct traffic to the correct host

(c) Direct traffic to the correct application

(d) Direct traffic to the correct RPC function

Question 18: What mechanism in TCP helps with out-of-order packets?

(a) sequence number

(b) syn flag

(c) congestion control

(d) flow control

Question 19: True or False. UDP offers reliable transfer but not congestion control.

(a) true

(b) false

Question 20: Which could influence packet arrival order?

(a) Application using asynchronous polling

(b) Routing changes

(c) Speed of packet transmission

(d) The operating system used

Question 21: With TCP, which indicates that a packet may have been lost? (select all that apply)

(a) ☐ Receiver sends negative ack

(b) ☐ A timeout occurs on the sender

(c) ☐ A timeout occurs on the receiver

(d) ☐ Sender receives duplicate ack (same number as before)

Question 22: Which conditions can lead to congestion? (select all that apply)

(a) ☐ Two input streams to same destination IP

(b) ☐ Frames with CRC errors

(c) ☐ Input ports feeding a slower output port

(d) ☐ TTL reaching zero

Question 23: When buffers fill and senders retransmit at same rate, this leads to:

(a) congestion collapse

(b) exponential backoff

(c) network loop

(d) buffer deficit

Question 24: What is the rationale behind multiplicative decrease in TCP?

(a) Sender ensures fair share of bandwidth

(b) Receiver drops connection

(c) Rapid backoff to avoid congestion

(d) Reduce RTT timeout by half

Question 25: True or False. A domain name maps to exactly one IP address.

(a) true

(b) false

Question 26: Which of the following are true of Root DNS servers? (select all that apply)

(a) ☐ Use anycast for replication

(b) ☐ Cache popular domains for quick response

(c) ☐ Run by major cloud providers

(d) ☐ Have IPs configured in resolvers

Question 27: Differences between JSON (HTTP) APIs and gRPC APIs? (select all that apply)

(a) ☐ JSON is text-based; gRPC uses binary serialization

(b) ☐ JSON is used in web services, gRPC is not

(c) ☐ JSON requires JavaScript, gRPC supports many languages

(d) ☐ HTTP uses TCP, gRPC uses UDP

Question 28: Which attack would RPKI mitigate?

(a) Packet inspection attacks

(b) SYN flood attacks

(c) Fake BGP announcement attacks

(d) Buffer overflow attacks

Question 29: In symmetric encryption, what is the greater challenge vs. asymmetric?

(a) Performance of encryption

(b) Distributing a shared key

(c) Ensuring message integrity

Question 30: What is the role of a digital certificate?

(a) Bind public key to identity

(b) Bind public key to certificate authority

(c) Bind IP address to MAC address

(d) Bind shared key to application

Question 31: Which do both TLS and IPsec provide? (select all that apply)

(a) ☐ Message integrity

(b) ☐ Authentication of server

(c) ☐ Service availability

(d) ☐ Message confidentiality

Question 32: Which applies to both RPKI and TLS? (select all that apply)

(a) ☐ Protects against man-in-the-middle attacks

(b) ☐ Improves Internet security

(c) ☐ Relies on certificate authority

(d) ☐ Ensures service availability

Question 33: The Diffie–Hellman protocol is used for:

(a) Providing access control

(b) Exchanging supported algorithm info

(c) Notifying users of breaches

(d) Establishing a shared secret key

Question 34: Networking uses a layered approach (link, network, transport, application). What are its positive and negative impacts?

Short answer: _____

Answer Key

Q1: D

Q2: (b)

Q3: (c), (d)

Q4: (a), (c)

Q5: (a)

Q6: False

Q7: True

Q8: DHCP

Q9: OSPF

Q10: (a), (c), (d)

Q11: (c)

Q12: 11.11.11.11 (longest prefix match /23)

Q13: false

Q14: true

Q15: ping

Q16: traceroute

Q17: (c)

Q18: (a)

Q19: false

Q20: (b)

Q21: (b), (d)

Q22: (a), (c)

Q23: (a)

Q24: (c)

Q25: false

Q26: (a), (d)

Q27: (a)

Q28: (c)

Q29: (b)

Q30: (a)

Q31: (a), (b), (d)

Q32: (a), (b), (c)

Q33: (d)

Q34: Layering simplifies design, enables interoperability and modularity, but can reduce efficiency and introduce redundancy.