# 1. The Need for Networking Models

### What is a Network?

- A network is two or more computing devices interconnected using communication protocols to share resources.
- Resources can include files, printers, online services, etc.
- The largest network in the world is the **internet**.

### Key Organizations Governing the Internet

- **ISOC (Internet Society)**: Promotes open development and evolution of the internet.
- **IETF (Internet Engineering Task Force)**: Develops and promotes internet standards.
- **IAB (Internet Architecture Board)**: Oversees internet standards and architecture.
- **IANA (Internet Assigned Numbers Authority)**: Manages IP addresses, domain names, and protocol assignments.
- **ICANN (Internet Corporation for Assigned Names and Numbers)**: Coordinates and maintains internet namespaces and numerical spaces.

### Early Challenges in Networking

- In the 1970s, vendors used **proprietary networking models** (e.g., AppleTalk, Novell NetWare).
- Devices from different vendors could not communicate.
- This led to the development of **vendor-neutral models** like the **OSI model** and **TCP/IP**.

### Why Networking Models Are Needed

- Ensure interoperability between devices from different vendors.
- Provide a common set of rules (protocols) for communication.
- Support scalability and growth of networks.

# 2. Exploring the OSI Model

### Introduction to the OSI Model

- Developed by the **International Organization for Standardization (ISO)**.
- A **7-layer reference model** that defines how systems communicate over a network.
- Not implemented in real devices but used as a conceptual framework.

## 2.1. The Seven Layers & Protocol Data Units (PDUs)

The model consists of seven layers, each with a unique role. Data at each layer is referred to by a specific name:

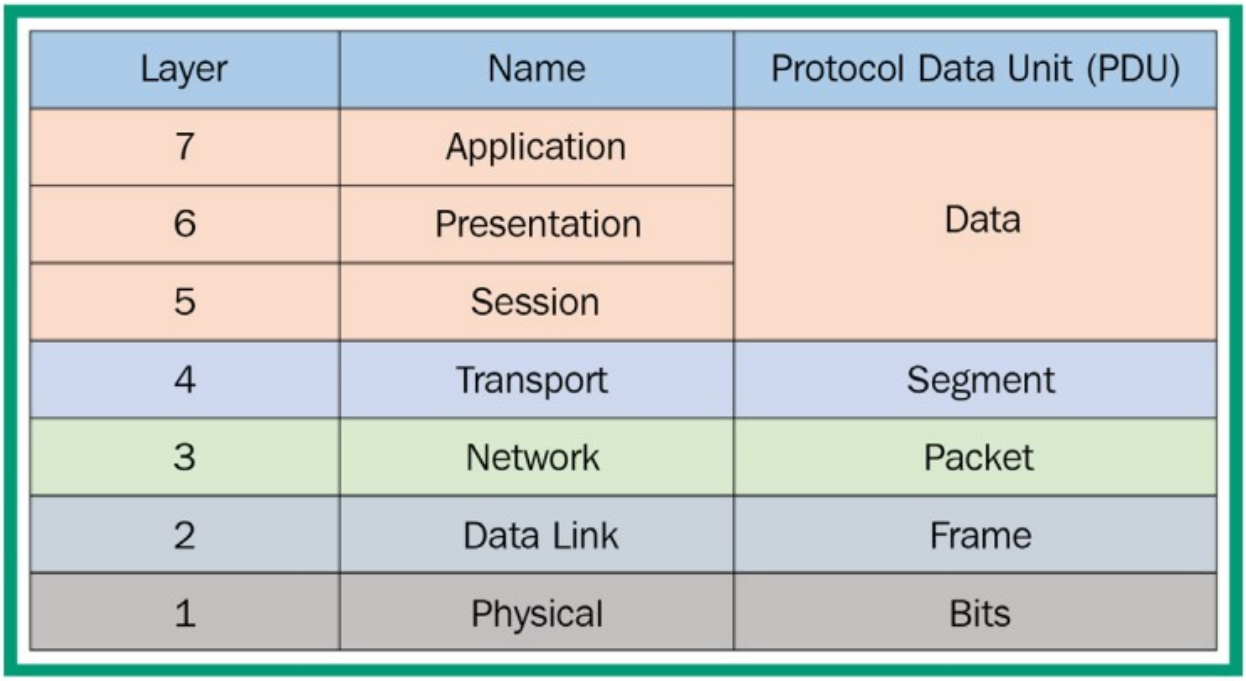



*Figure 1: The seven layers of the OSI model*

- **Encapsulation:** The process of adding headers (and trailers) to the PDU as it moves **down** the OSI stack from the Application to the Physical layer.
- **De-encapsulation:** The process of removing these headers as the PDU moves **up** the OSI stack on the receiving device.

## 2.2. Detailed Layer Breakdown

**Layer 7: Application Layer**

- **Function:** Closest to the end-user. It provides the interface for applications (like web browsers or email clients) to access network services.
- **Key Insight:** Application layer protocols (e.g., HTTP, HTTPS, SMTP) create PDUs that can only be interpreted by the same protocol on the receiving end.
- **Example:** Using a web browser to access `www.comptia.org` uses the HTTPS protocol at this layer.

**Layer 6: Presentation Layer**

- **Function:** Acts as a translator. It ensures data from the Application layer is formatted in a way that the lower layers can understand and that the receiving system can interpret.
- **Responsibilities:**
    - Data formatting (encoding)
    - Data compression
    - Data encryption & decryption

**Layer 5: Session Layer**

- **Function:** Manages the dialog between two devices. It establishes, maintains, and terminates the logical session (connection) between them.
- **Core Functions:** Create a session, maintain the session, terminate a session.
- **Key Insight:** If the session is terminated during data transmission, all data transmission will stop.

**Layer 4: Transport Layer**

- **Function:** Ensures end-to-end delivery of data. Its primary responsibility is to deliver messages between the Application layer and the network.
- **Service Ports:** The Transport Layer uses **logical port numbers** (0-65,535) to ensure data is delivered to the correct application on the destination device.
    - **Well-known ports (0-1,023):** e.g., HTTP (80), HTTPS (443), SMTP (25)
    - **Registered ports (1,024-49,151)**
    - **Dynamic/Private ports (49,152-65,535):** Used as ephemeral source ports.

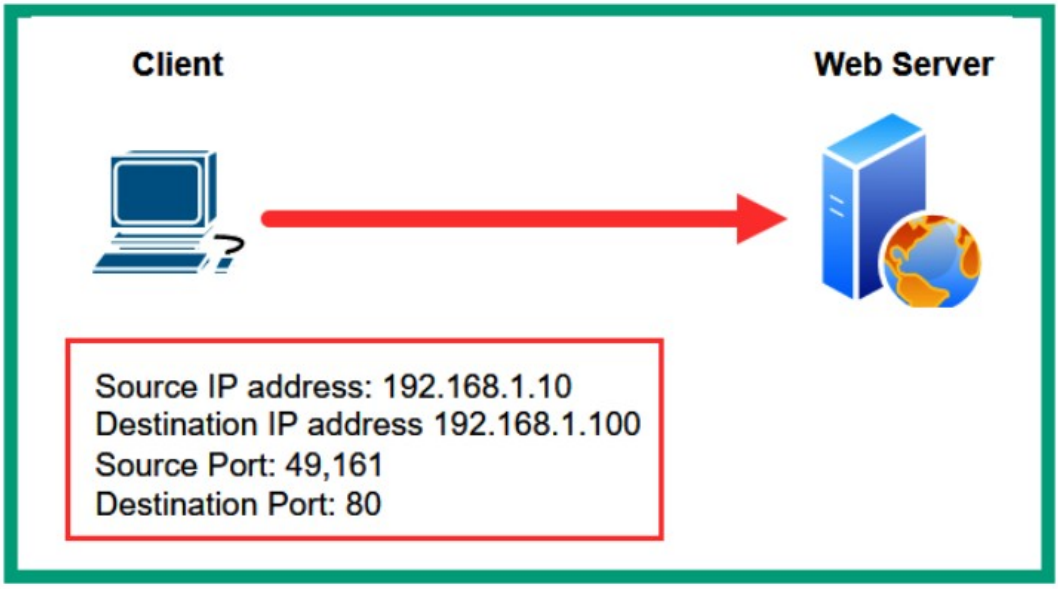



*Figure 2: Client sending a message to the web server*

- **Protocols:** It uses two main protocols:
    - **TCP (Transmission Control Protocol):** Connection-oriented, reliable, uses a **three-way handshake** (SYN, SYN/ACK, ACK), provides guaranteed delivery and flow control. Has more overhead.

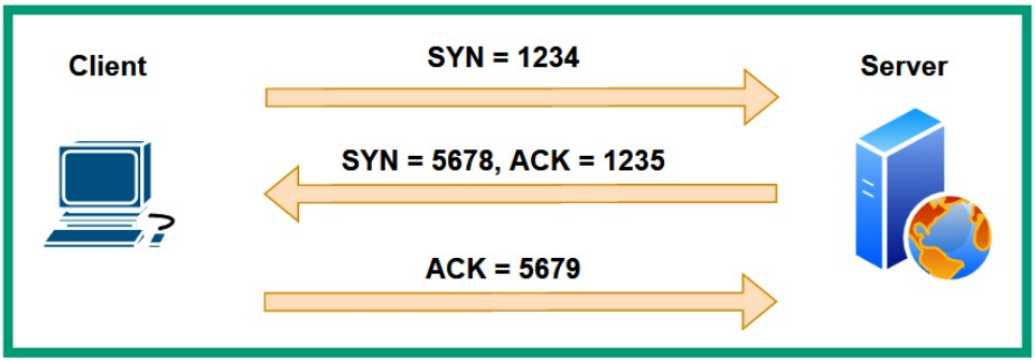



*Figure 3: TCP three-way handshake with sequence numbers*

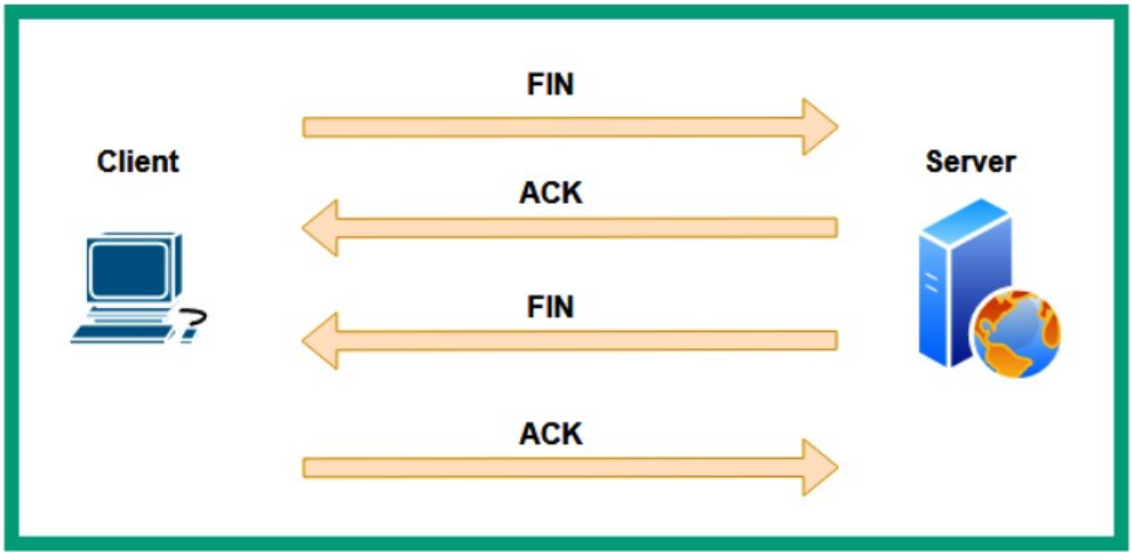



*Figure 4: TCP terminating a connection*

- **UDP (User Datagram Protocol):** Connectionless, unreliable, no handshake, faster, lower overhead. Ideal for time-sensitive applications like VoIP.

**Layer 3: Network Layer**

- **Function:** Responsible for **logical addressing** (IP addresses) and **routing**. It encapsulates the segment into a **packet** by adding a header containing the source and destination IP addresses.
- **Key Device:** Routers operate at this layer.
- **Protocol:** The Internet Protocol (IP) is a connectionless, "best-effort" protocol that does not guarantee delivery.

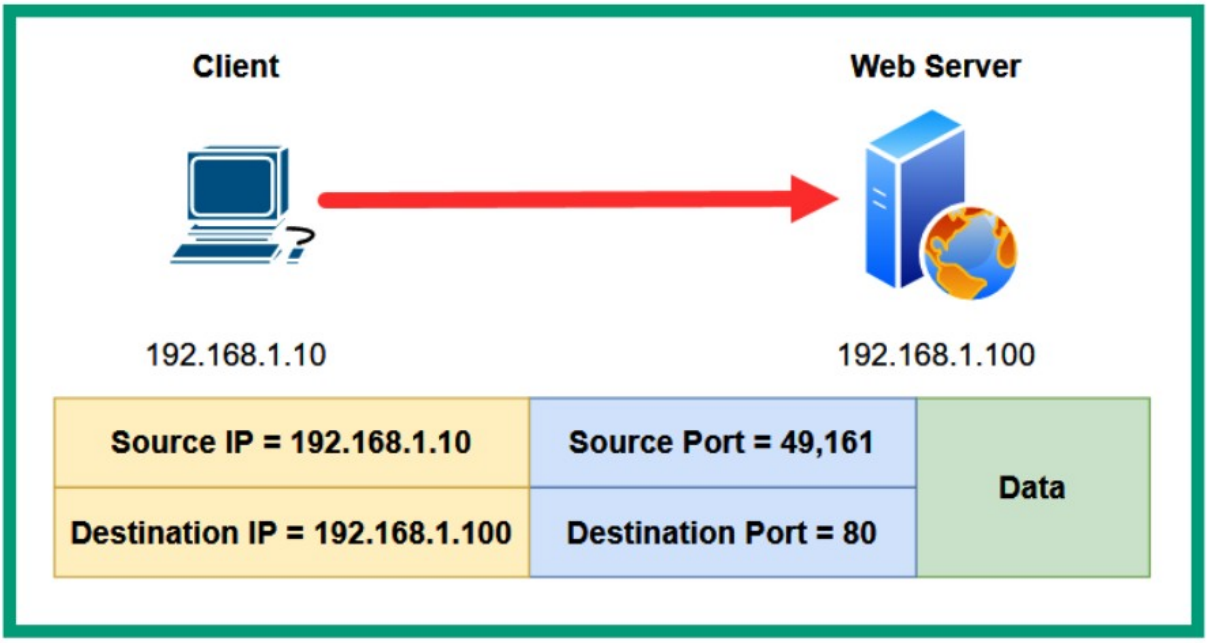



*Figure 5: Packet header*

**Layer 2: Data Link Layer**

- **Function:** Responsible for **physical addressing** (MAC addresses) and preparing the packet for transmission over the physical media. It encapsulates the packet into a **frame** by adding a header and trailer.

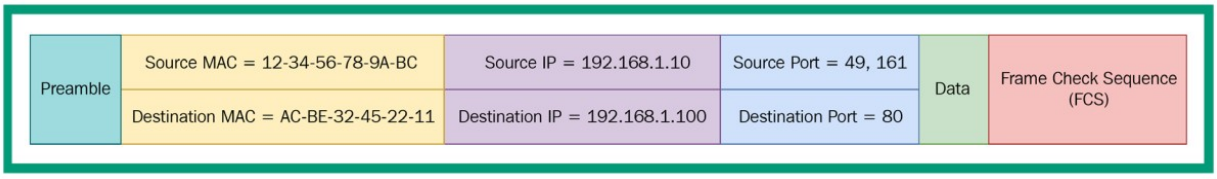



*Figure 6: High-level overview of a Frame*

- **Sublayers:**
  - **LLC (Logical Link Control):** Communicates with the upper layers (software).
  - **MAC (Media Access Control):** Handles data encapsulation, access to the NIC, and physical addressing.
- **MAC Address:** A 48-bit (6-byte) burned-in address (BIA) on a Network Interface Card (NIC). The first 3 bytes are the OUI (Organizationally Unique Identifier), identifying the vendor.
- **Error Detection:** The trailer contains the **Frame Check Sequence (FCS)**, which includes a CRC value to check the integrity of the frame.
- **Frame Size:** Minimum 64 bytes, maximum 1,518 bytes (excluding the preamble).

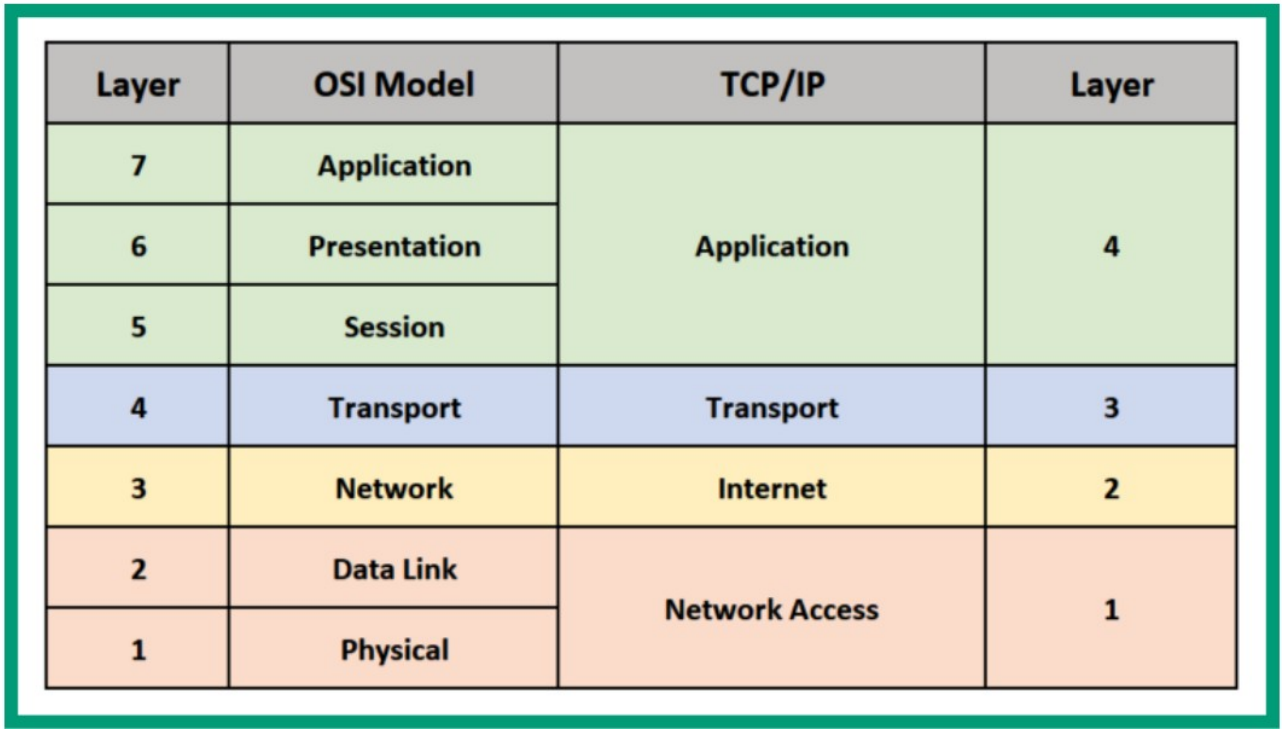
**Layer 1: Physical Layer**

- **Function:** Transmits the raw **bits** (electrical, optical, or radio signals) over the physical medium. It defines the standards for hardware components, cables, connectors, encoding, and signaling.
- **Governed by:** Standards organizations like ISO, TIA/EIA, IEEE, and FCC.

## 3. Understanding TCP/IP

- **Origin:** Developed by the US Department of Defense (DoD) to be a vendor-neutral protocol suite.
- **Status:** It is the **dominant and universal** networking model used on all modern networks, including the internet and private networks. Every device connected to a network implements TCP/IP.
- **Name:** While named after its two core protocols (**T**ransmission **C**ontrol **P**rotocol and **I**nternet **P**rotocol), it is actually a **suite** of many protocols that work together.

The TCP/IP model is a condensed version of the OSI model.



| Layer | OSI Model | TCP/IP | Layer |
|---|---|---|---|
| 7 | Application | Application | 4 |
| 6 | Presentation | | |
| 5 | Session | | |
| 4 | Transport | Transport | 3 |
| 3 | Network | Internet | 2 |
| 2 | Data Link | Network Access | 1 |
| 1 | Physical | | |

*Figure 7: TCP/IP protocol suite*

**Layer 4: Application Layer**

- **Function:** Combines the functionality of the top three OSI layers. It defines how applications create user data and interact with lower-layer protocols to transmit that data.
- **Protocols:** This layer contains **high-level protocols**.
    - **HTTP/HTTPS:** For web browsing.
    - **SMTP:** For sending email.

    - **FTP:** For file transfer.
    - **DNS:** For domain name resolution.
- **PDU:** The data created here is simply called **Data**.

**Layer 3: Transport Layer**

- **Function: Exactly the same** as the OSI Transport Layer. It provides communication sessions between devices and ensures data is delivered reliably.
- **Key Roles:**
    - **Segmentation:** Breaking data into smaller pieces.
    - **Service Port Addressing:** Using port numbers to direct data to the correct application.
    - **Connection Management:** Establishing sessions (TCP 3-way handshake).
- **Protocols:**
    - **TCP:** For connection-oriented, reliable communication.
    - **UDP:** For connectionless, best-effort communication.
- **PDU:** The encapsulated data at this layer is called a **Segment**.

**Layer 2: Internet Layer**

- **Function: Exactly the same** as the OSI Network Layer. It is responsible for addressing, packaging, and routing data across multiple networks.
- **Key Roles:**
    - **Logical Addressing:** Using IP addresses to identify hosts on different networks.
    - **Routing:** Determining the best path for packets to travel through a network.
- **Protocol:** The core protocol is the **Internet Protocol (IP)**. Other supporting protocols include ICMP (for diagnostics) and ARP (for address resolution).
- **PDU:** The encapsulated data at this layer is called a **Packet** (or **Datagram** for UDP).

**Layer 1: Network Access Layer**

- **Function:** Combines the functionality of the OSI Data Link and Physical layers. It defines the procedures for interfacing with the network hardware and transmitting data over the physical medium.
- **Key Roles:**
    - **Physical Addressing:** Using MAC addresses to identify devices on the *same local network*.
    - **Framing:** Encapsulating packets into frames with a header and trailer.
    - **Media Access Control:** Governing how devices share the medium (e.g., CSMA/CD for Ethernet).

- **Signaling & Transmission:** Converting the frame into bits and transmitting them as electrical, optical, or radio signals.

- **PDU:** The encapsulated data at this layer is called a **Frame**. Once placed on the medium, it is simply **Bits**.

## 4. Data encapsulation concepts

- **Definition:** The process of adding headers (and sometimes trailers) to data as it moves **down** the OSI or TCP/IP protocol stack on the sending device.
- **Purpose:** Each layer adds addressing and control information specific to its function, ensuring the data can be successfully transmitted across the network and delivered to the correct application on the destination host.
- **Reverse Process: De-encapsulation** is the process of removing these headers as the data moves **up** the stack on the receiving device.

### 4.1. Ethernet Header (Data Link Layer - Layer 2)

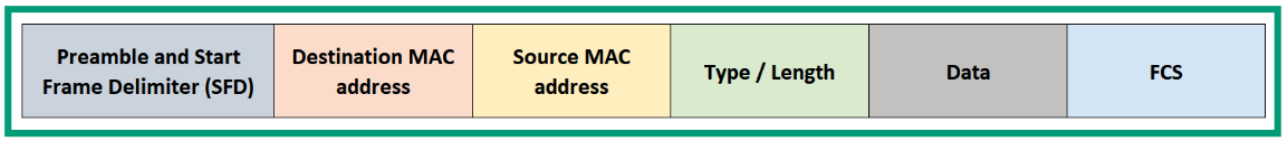



*Figure 8: Ethernet header*

**Key Fields:**

- **Preamble & SFD (Start Frame Delimiter):** 8 bytes total. Synchronizes communication and indicates the start of a frame.
- **Destination MAC Address (6 bytes):** The physical address of the next device to receive the frame.
- **Source MAC Address (6 bytes):** The physical address of the sender's NIC.
- **Type/Length (2 bytes):** Identifies the upper-layer protocol contained in the data (e.g., IPv4, IPv6).
- **Data (46-1500 bytes):** The payload received from the Network Layer (the Packet).
- **FCS (Frame Check Sequence) (4 bytes):** Contains a CRC value for error detection. The receiving device calculates its own CRC; if it doesn't match the FCS, the frame is discarded.

- **Frame Size:** Frames must be between **64 bytes** (to avoid collisions) and **1,518 bytes** (including headers and trailer).

### 4.2. IPv4 Header (Network/Internet Layer - Layer 3)

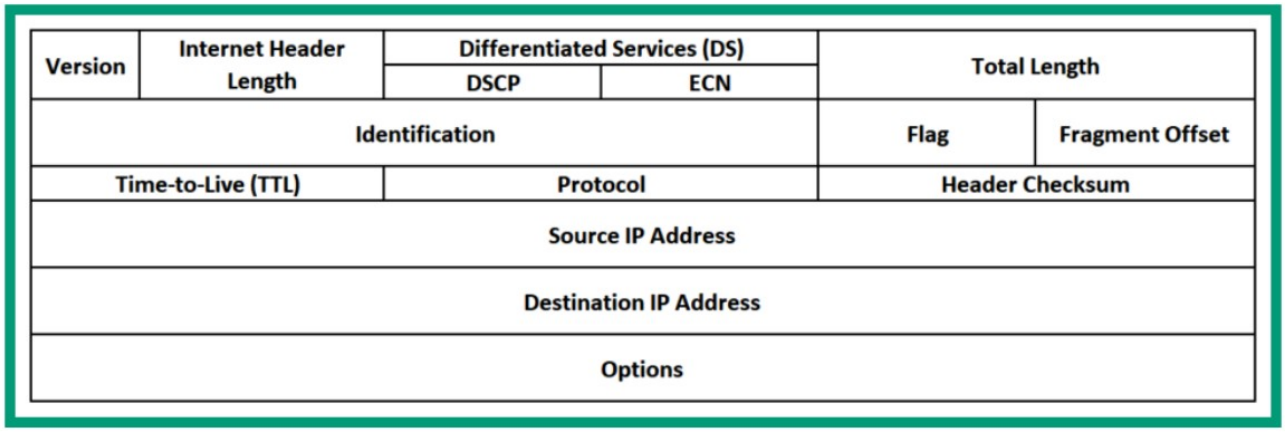



*Figure 9: IPv4 Header*

**Key Fields:**

- **Version (4 bits):** Identifies the IP version (e.g., 4 for Ipv4).
- **Internet Header Length:** This field is made up of 4 bits and is used to indicate where the header section ends and the data section starts.
- **Differentiated Services (DS) (1 byte):** Used for Quality of Service (QoS) to prioritize traffic. Contains:
    - **DSCP (6 bits):** Differentiated Services Code Point for priority.
    - **ECN (2 bits):** Explicit Congestion Notification.
- **Total length:** This field is made up of 16 bits (2 bytes) and is used to indicate the total size of the IPv4 packet.
- **Identification:** This field is made up of 16 bits (2 bytes) and is used to provide identification numbering to each fragmented packet that belongs to an original message.
- **Flags:** This field is made up of 3 bits and is used to indicate whether the packet is to be fragmented or not.
- **Fragment offset:** This field is made up of 13 bits and is used to indicate the sequencing position of a fragmented packet.
- **Time to Live (TTL) (1 byte):** Prevents packets from looping forever. Decremented by 1 at each router. If it reaches 0, the packet is discarded.
- **Protocol (1 byte):** Identifies the upper-layer protocol (e.g., 6 for TCP, 17 for UDP).
- **Source IP Address (4 bytes):** The logical address of the sender.
- **Destination IP Address (4 bytes):** The logical address of the intended recipient.
- **Header Checksum (2 bytes):** Used to check for errors in the IPv4 header only.

### 4.3. IPv6 Header (Network/Internet Layer - Layer 3)

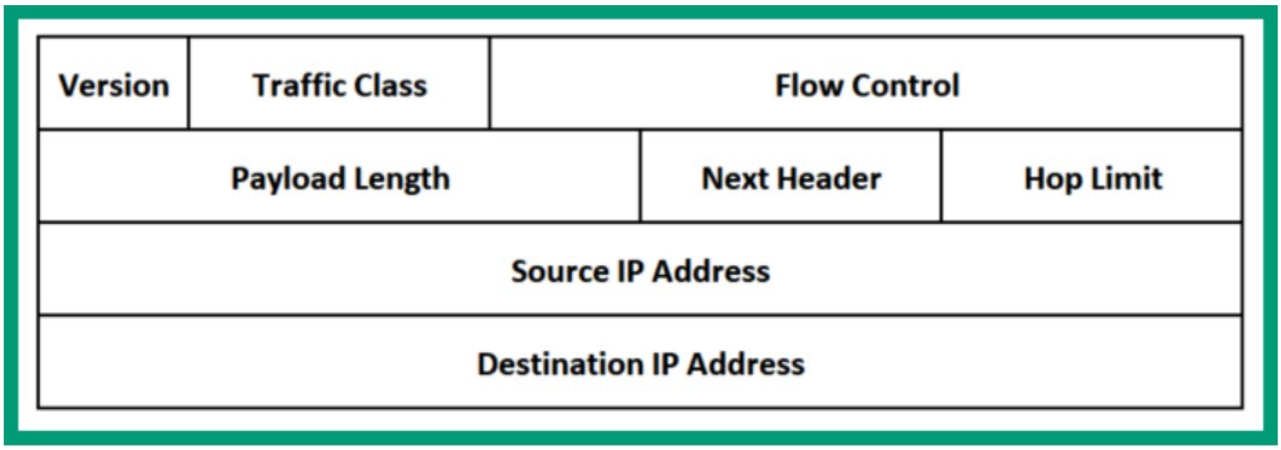



*Figure 10: IPv6 Header*

**Key Fields:**

- **Version (4 bits):** Identifies the IP version (value 6).
- **Traffic Class (8 bits):** Similar to IPv4's DS field; used for QoS prioritization.
- **Flow Label (20 bits):** Used to identify packets belonging to the same flow/communication, allowing for special handling by routers.
- **Payload Length (16 bits):** Indicates the length of the data (**payload**) following the header.
- **Next Header (8 bits):** Replaces the IPv4 Protocol field. Identifies the type of the next header.
- **Hop Limit (8 bits):** Replaces IPv4's TTL. Decremented by 1 by each router; packet is discarded if it reaches 0.
- **Source Address (128 bits):** The IPv6 address of the sender.
- **Destination Address (128 bits):** The IPv6 address of the intended recipient.

### 4.4. TCP Header (Transport Layer - Layer 4)

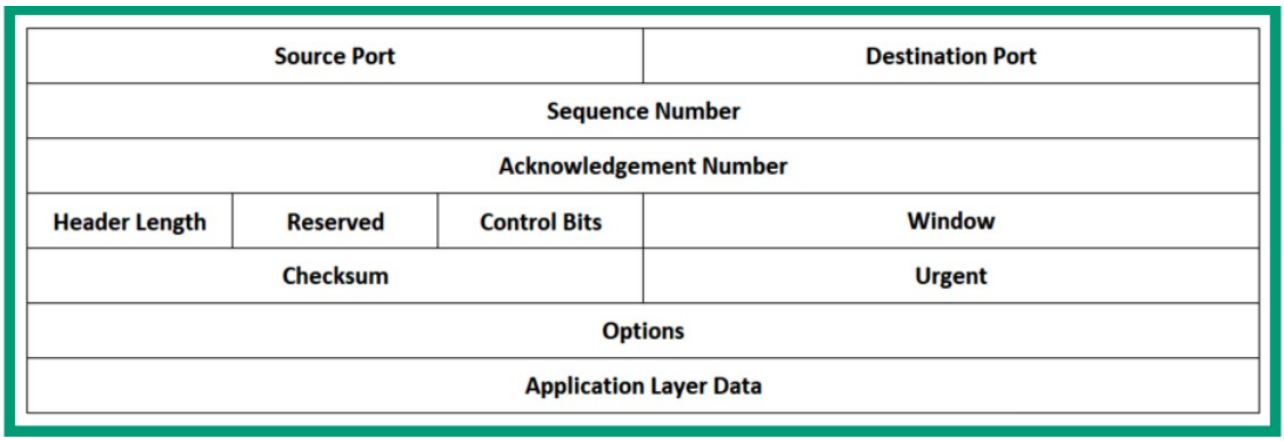


*Figure 11: TCP Header*

**Key Fields:**

- **Source Port (2 bytes):** The port number of the sending application.
- **Destination Port (2 bytes):** The port number of the destination application.
- **Sequence Number (4 bytes):** Used for reassembling data in the correct order.
- **Acknowledgment Number (4 bytes):** Confirms receipt of data; contains the next expected sequence number.
- **Header Length (4 bits):** Indicates the length of the TCP header.
- **Control Bits (Flags) (6 bits):** Control connection states (e.g., **SYN**, **ACK**, **FIN**, **RST**, **PSH**, **URG**).
- **Window Size (2 bytes):** Used for flow control; indicates how much data the receiver can accept.
- **Checksum (2 bytes):** Used for error detection of the TCP header and data.
- **Urgent:** This is a 16-bit (2-byte) field that is used to indicate urgency on the TCP header.

### 4.4. UDP Header (Transport Layer - Layer 4)

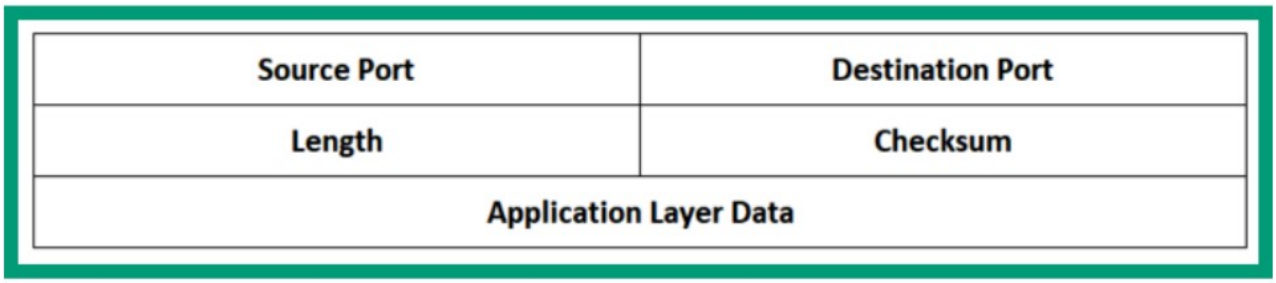


*Figure 12: UDP Header*

**Key Fields:**

- **Source Port (2 bytes):** The port number of the sending application.

- **Destination Port (2 bytes):** The port number of the destination application.
- **Length (2 bytes):** The length of the entire UDP segment (header + data).
- **Checksum (2 bytes): Used for error detection (optional in IPv4).**

## 5. Error Detection Methods (Parity, Checksum, CRC)

### 5.1. Parity Check

- **Concept:** The simplest form of error detection. A single extra bit (the **parity bit**) is added to a binary data unit (often a byte) to make the total number of `1`s either even or odd.
- **Types:**
    - **Even Parity:** The parity bit is set to `1` or `0` to make the total number of `1`s **even**.
    - **Odd Parity:** The parity bit is set to `1` or `0` to make the total number of `1`s **odd**.
- **Process:**
    - The sender counts the number of `1`s in the data.
    - The sender sets the parity bit according to the chosen scheme (even or odd).
    - The receiver counts the number of `1`s in the received data (including the parity bit).
    - If the count matches the expected outcome (even or odd), the data is assumed correct. If not, an error is detected.
- **Limitations:**
    - **Detects only an odd number of bit errors.** If two bits are flipped (an even number of errors), the parity check will **fail to detect the error**.
    - It cannot **correct** errors, only detect them.

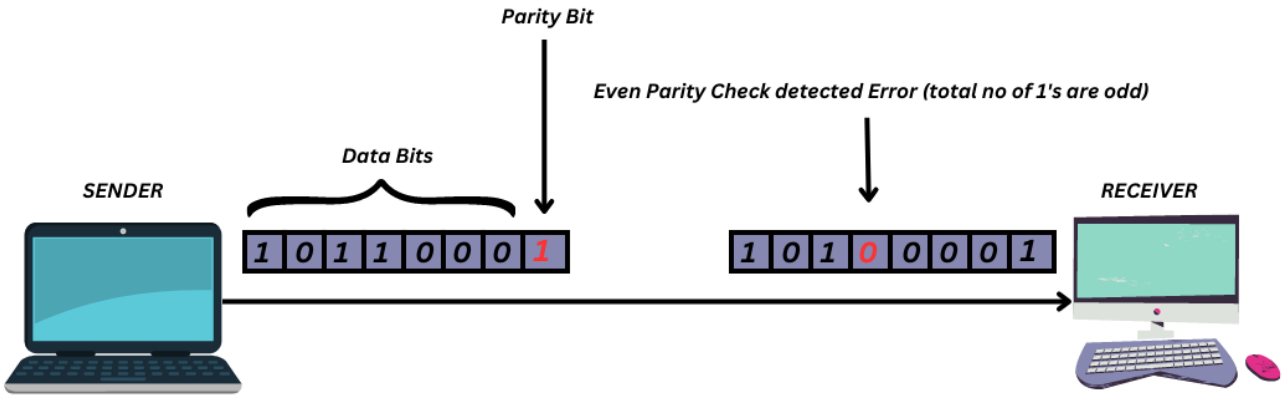



*Figure 13: Even Parity*

### 5.2. Checksum

- **Concept:** A more robust method where a value is calculated based on the contents of a packet or segment. This value is appended to the data before transmission.

- **Process (as applied in TCP/UDP/IP):**
    1. **Sender:**
        - The data to be sent is divided into fixed-size segments (e.g., 16-bit words).
        - All segments are added together using **one's complement arithmetic**.
        - The **one's complement** of this sum is taken, resulting in the **checksum value**.
        - This checksum value is placed in the header (IP, TCP, UDP).
    2. **Receiver:**
        - Performs the same calculation on the received data (including the checksum field).
        - If the result is **all 1s** (which is equivalent to -0 in one's complement), the data is assumed to be error-free. Any other value indicates an error.

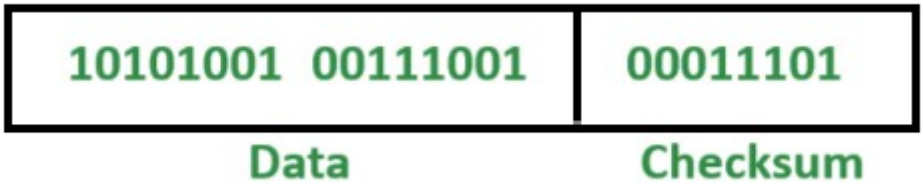



*Figure 14: Checksum*

- **Strength:** Better than parity at detecting common errors, such as a single-bit error or a burst error.
- **Weakness:** Relatively weak compared to CRC. It is possible for multiple errors to cancel each other out, resulting in the same checksum and an **undetected error**.

### 5.3. CRC (Cyclic Redundancy Check)

- **Concept:** The most powerful common error detection method used at the **Data Link Layer**. It treats the data as a large binary number and performs a **polynomial division** to generate a remainder, which becomes the CRC value.
- **Process:**
    1. A standard **generator polynomial** (e.g., CRC-32, CRC-16) is agreed upon by the sender and receiver.
    2. **Sender:**
        - The data (e.g., the Ethernet frame's header and payload) is divided by the generator polynomial.
        - The **remainder** of this division is the **CRC value**.

- This value is placed in the **Frame Check Sequence (FCS)** field of the frame's trailer.

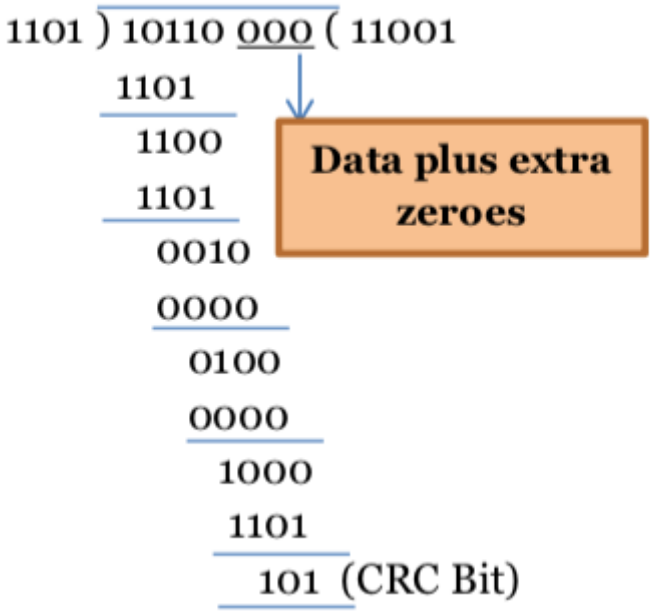



*Figure 15: CRC(Sender)*

3. **Receiver:**
    - Performs the same polynomial division on the received frame (including the FCS field).
    - If the remainder of this division is **zero**, the frame is error-free. Any non-zero remainder indicates an error.

- **Strengths:**
    1. **Excellent at detecting** common types of errors, including:
        - All single-bit errors.
        - All double-bit errors, as long as the generator polynomial has at least three terms.
        - Any odd number of errors.
        - Any burst error shorter than the length of the polynomial.

## 6. Error Correction Methods (Hamming Code)

- **Concept:** It involves adding several redundant **parity bits** to the original data bits. These parity bits are placed in specific positions (powers of 2: 1, 2, 4, 8, etc.) within the code word.

  Each parity bit calculates and checks the parity (even or odd) for a specific subset of the total bits. If a single bit gets flipped, the pattern of which parity checks fail directly tells us the position of the error.

- **Process:**

  **Let's encode a 4-bit message: `1011`.**

**Step 1: Determine the number of parity bits (p) needed.**

The formula is:
**2^p ≥ m + p + 1**
where:

- `m` = number of data bits
- `p` = number of parity bits
- The `+1` is for the scenario of no error.

For `m=4`:

- 2^p ≥ 4 + p + 1
- Try p=3: $2^3=8 \geq 4+3+1=8$. It works.

So, our total code word will be **7 bits long** (4 data bits + 3 parity bits).

**Step 2: Place the parity bits in the correct positions.**

Parity bits (`P`) go in positions that are powers of two: **1, 2, 4**.
Data bits (`D`) go in the other positions: **3, 5, 6, 7**.

Let's write out the positions:

| **Bit Position** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| Type | P1 | P2 | D3 | P4 | D5 | D6 | D7 |
| Value | ? | ? | 1 | ? | 0 | 1 | 1 |

Our data bits `1011` are placed in positions 3, 5, 6, and 7.

**Step 3: Determine the values of the parity bits.**

Each parity bit is calculated based on **even parity**.

- **P1 (Position 1)**: Covers all bits where the **least significant bit** in the position number is `1`.
  - Positions: 1, 3, 5, 7 → P1, D3, D5, D7
  - Values: P1, 1, 0, 1. We want even parity (even number of 1s).
  - So: P1 + 1 + 0 + 1 = even. Therefore, **P1 = 0**. (0+1+0+1=2, which is even )
- **P2 (Position 2)**: Covers all bits where the **second bit** is `1`.
  - Positions: 2, 3, 6, 7 → P2, D3, D6, D7
  - Values: P2, 1, 1, 1. We want even parity.
  - So: P2 + 1 + 1 + 1 = even. Therefore, **P2 = 1**. (1+1+1+1=4, which is even )
- **P4 (Position 4)**: Covers all bits where the **third bit** is `1`.
  - Positions: 4, 5, 6, 7 → P4, D5, D6, D7

- Values: P4, 0, 1, 1. We want even parity.
- So: P4 + 0 + 1 + 1 = even. Therefore, **P4 = 0**. (0+0+1+1=2, which is even )

**Step 4: Final Encoded Message**

Now we fill in the parity values:

| **Position** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| Value | **0** | **1** | 1 | **0** | 0 | 1 | 1 |

The final encoded data we send is: **0110011**

## 5.1. Error Detection and Correction

Now, let's assume this message is transmitted and a **single-bit error** occurs. The receiver gets: **0110111** (Error at position 5).

The receiver must now check the parity to find and correct the error.

**Step 1: Re-calculate the parity checks.**

The receiver recalculates each parity group, assuming even parity, and records if the check is correct (0) or incorrect (1). This creates a **syndrome word**.

- **Check P1 (Positions 1, 3, 5, 7)**: 0, 1, `1`, 1 → 0+1+1+1=3 (odd). **Error! → 1**
- **Check P2 (Positions 2, 3, 6, 7)**: 1, 1, `0`, 1 → 1+1+1+1=4 (even). **Correct → 0**
- **Check P4 (Positions 4, 5, 6, 7)**: 0, `1`, `0`, 1 → 0+1+1+1=3 (odd). **Error! → 1**

**Step 2: Form the syndrome word.**

Write the results of the checks in reverse order: **P4 P2 P1**
The syndrome word is **1 0 1**.

**Step 3: Locate and correct the error.**

Convert the syndrome word `101` from binary to decimal:
(1 × 4) + (0 × 2) + (1 × 1) = **5**.

This tells us the error is in **bit position 5**. The receiver simply flips the bit at position 5 to correct it.

- **Incorrect received word:** 0 1 1 0 `1` 1 1
- **Corrected word:** 0 1 1 0 **0** 1 1