

Department of Computer Science and Engineering

Course with code: Computer Networks - BCS502

Semester:5th

Academic Year 2026-2027

Course Coordinator: Dr. Shivaprasad B J, Mrs. Sushma V, Mrs. Darshini M S, CSE, ATMECE



INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Department of Computer Science and Engineering

Vision of the Department

To develop highly talented individuals in computer science and engineering to deal with real world challenges in industry, education, research and society.

Mission of the Department

- To inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.
- Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research.

Program Educational Objectives (PEO'S):

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in related areas.

Program Specific Outcomes (PSOs)

1. Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.
2. Apply knowledge In the field of computer networks for building network and internet based applications.

MODULE-1: Data Communications & Networks

1. DATA COMMUNICATIONS

Data communication is the process of transferring data from one point to another using a communication system. It involves several essential components and mechanisms to ensure the accurate and timely delivery of data.

1.1. Components

A data communication system includes the following components:

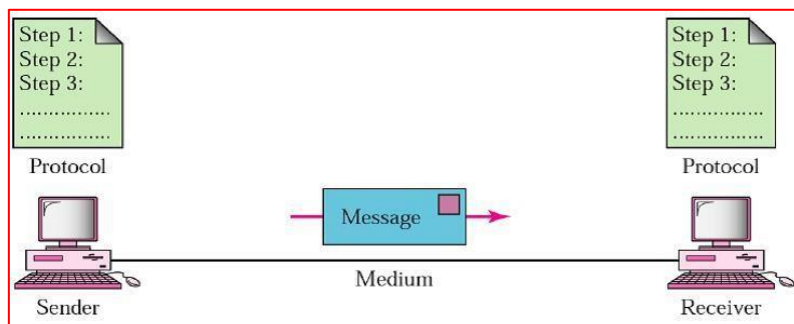


Figure 1.1: A data communications system has five components

1. **Message:** The data or information being communicated (e.g., text, images, audio).
2. **Sender:** The device that sends the message, such as a computer or smartphone.
3. **Receiver:** The device that receives the message, like another computer or a printer.
4. **Transmission Medium:** The physical path through which the data is transmitted, like cables or radio waves.
5. **Protocol:** A set of rules that governs the communication between devices to ensure proper data exchange.

The performance of a data communication system relies on four key characteristics: delivery, accuracy, timeliness, and jitter.

1. **Delivery:** The system must ensure that data reaches the correct destination. Only the intended recipient—whether a device or a user—should receive the data.
2. **Accuracy:** Data must be transmitted without errors. If data is altered during transmission and not corrected, it becomes unusable.
3. **Timeliness:** Data must be delivered promptly. Delayed data, especially in applications like video and audio, lose their value. For real-time transmission, data must be delivered in the same sequence and without significant delays.

4. **Jitter:** Jitter refers to the inconsistency in packet arrival times. Inconsistent delays, such as video packets arriving at varying intervals, can degrade the quality of the audio or video. For instance, if video packets are sent every 30 ms, but some arrive after 40 ms, the video quality will be affected.

1.2. Data Representation

Data can be represented in various forms to suit the type of communication. Common types of data representation include:

- **Text:** Represented as a sequence of bits using encoding systems like ASCII or Unicode.
- **Numbers:** Represented directly in binary form, which allows for efficient computation and transmission.
- **Images:** Represented as a matrix of pixels, where each pixel is assigned a bit pattern based on the image's color or intensity. Color images often use RGB (Red, Green, Blue) or YCM (Yellow, Cyan, Magenta) encoding.
- **Audio:** Audio data is captured as a continuous signal, which can be sampled and digitized for transmission.
- **Video:** A sequence of images or frames is used to convey motion, with each frame represented as individual bit patterns.

1.3. Data Flow

Data flow refers to the manner in which data is transmitted between two devices. It can happen in three modes:

1. **Simplex:** Data flows in one direction only, like a keyboard sending data to a computer (one-way communication).
2. **Half-Duplex:** Both devices can send and receive data, but not at the same time. For example, a walkie-talkie allows communication in both directions, but one at a time.
3. **Full-Duplex:** Both devices can transmit and receive data simultaneously, like in a phone conversation where both parties can speak and listen at the same time.

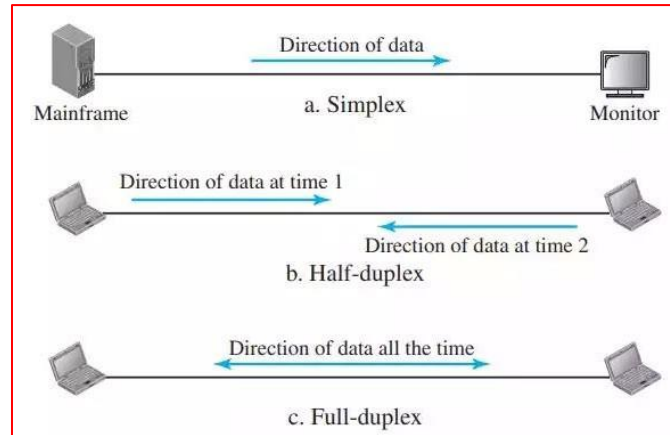


Figure 1.2: Data flow (simplex, half-duplex, and full-duplex)

2. NETWORKS

A network refers to the interconnection of a set of devices capable of communication. These devices can include hosts such as computers, smartphones, and security systems, as well as connecting devices like routers, switches, and modems. Networks are connected through transmission media, either wired (like cables) or wireless (like air).

2.1. Network Criteria

Networks must meet several essential criteria to be effective, namely:

1. **Performance:** This is evaluated by:
 - **Transit time:** Time for a message to travel between two devices.
 - **Response time:** Time between sending a request and receiving a response. Factors influencing performance include the number of users, the transmission medium, hardware capabilities, and software efficiency. Key metrics include:
 - **Throughput:** Amount of data transmitted successfully.
 - **Delay:** Time taken for data to reach its destination. Optimizing both often leads to trade-offs.
2. **Reliability:** This includes:
 - Accuracy in data delivery.
 - Frequency and recovery time from network failures.
 - The network's ability to function during a catastrophe.
3. **Security:** It encompasses protecting data from unauthorized access, damage, or corruption and establishing recovery procedures for security breaches.

2.2. Physical Structures

Networks can be categorized by their **connection types** and **topologies**:

1. Type of Connection:

- **Point-to-Point:** A direct link between two devices, providing the full capacity of the link for communication (e.g., remote control to TV).
- **Multipoint (Multidrop):** Multiple devices share a single link, either spatially (simultaneous use) or temporally (taking turns).

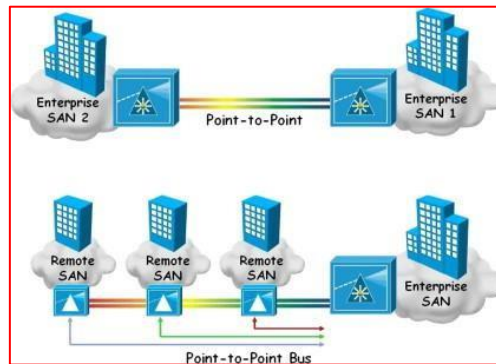


Figure 1.3: Types of connections: point-to-point and multipoint

2. **Physical Topology:** Refers to how devices are physically arranged in a network. Four main topologies include:

- **Mesh Topology:**
 - Every device is connected to every other device, requiring $n(n-1)/2$ links for n devices.
 - Node 1 must be connected to $n - 1$ nodes, node 2 must be connected to $n - 1$ nodes, and finally node n must be connected to $n - 1$ nodes. We need $n(n - 1)$ physical links. However, if each physical link allows communication in both directions (duplex mode), we can divide the number of links by 2. In other words, we can say that in a mesh topology, we need $n(n - 1) / 2$ duplex-mode links.
 - Advantages: Dedicated links, fault isolation, robust, secure.
 - Disadvantages: Expensive, complex installation, excessive cabling.
 - Example: Telephone networks between regional offices.

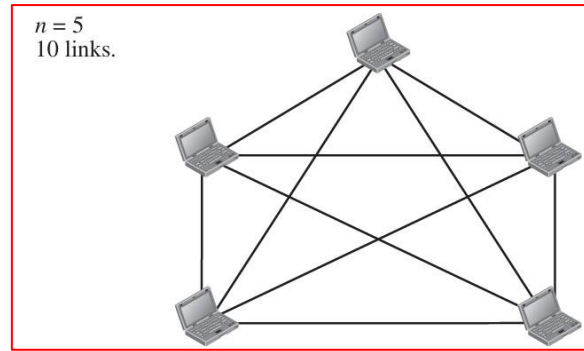


Figure 1.4 A fully connected mesh topology (five devices)

○ **Star Topology:**

- Each device is connected to a central hub, which manages communication.
- Advantages: Easy installation and fault isolation; if a link fails, only that device is affected.
- Disadvantages: Entire system fails if the hub goes down.
- Common in local area networks (LANs).

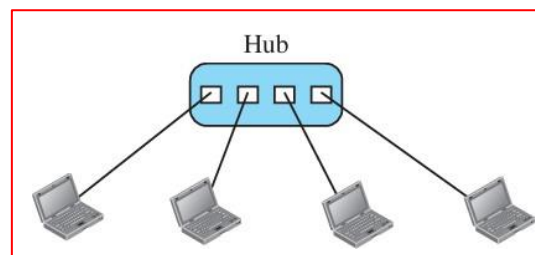


Figure 1.5: A star topology connecting four stations

○ **Bus Topology:**

- All devices are connected to a single backbone cable.
- Advantages: Easy installation, less cabling than mesh.
- Disadvantages: Difficult to add devices, faults in the backbone disrupt the entire network.
- Example: Early Ethernet LANs.

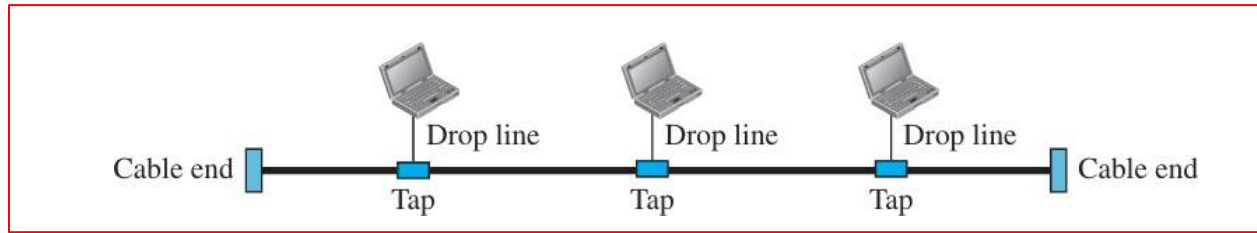


Figure 1.6: A bus topology connecting three stations

○ **Ring Topology:**

- Devices are connected in a loop, with signals traveling in one direction through repeaters.
- Advantages: Easy to install, simple fault detection.
- Disadvantages: A break in the ring can disable the entire network, though dual rings or switches can mitigate this.
- Example: IBM's Token Ring LANs.

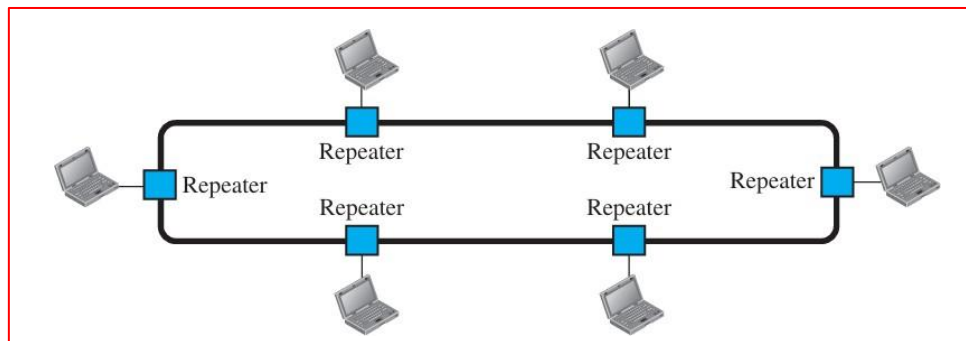


Figure 1.7: A ring topology connecting six stations

3. NETWORK TYPES

Which is used to connect networks to form an internetwork (a network of networks).

3.1. Local Area Network (LAN)

A LAN is a network that connects computers and devices within a small geographical area, such as a home, office, or campus.

Characteristics:

- Covers a limited area.
- High data transfer rates (up to 10 Gbps).
- Typically owned and managed by a single organization.

Examples: Ethernet networks, Wi-Fi networks in homes and small offices.

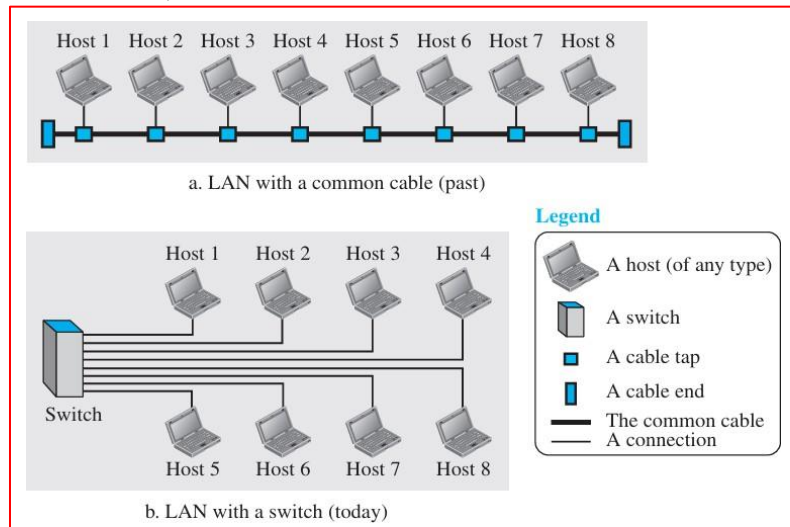


Figure 1.8: An isolated LAN in the past and today

LAN Setup:

- **Common Cable Connection:** In the past, all hosts within a Local Area Network (LAN) were connected via a single common cable.
 - **Packet Transmission:** When one host sent a packet to another, it was broadcast to all hosts on the network.
 - **Packet Filtering:** The intended recipient would accept the packet, while all other hosts would drop it.
 - **Drawback:** This method created significant network congestion, as every packet was visible to all hosts, even when they weren't the intended recipient.

Modern LAN Setup:

- **Switch-Based LAN:** Today, most LANs use **smart switches** to handle packet delivery.
 - **Address Recognition:** A switch is intelligent enough to recognize the destination address of each packet.
 - **Selective Packet Forwarding:** Instead of broadcasting packets to all hosts, the switch sends the packet directly to the destination host.
 - **Benefits:**
 - **Reduced Traffic:** Since packets are only sent to the intended recipient, overall network traffic is significantly reduced.
 - **Simultaneous Communication:** More than one pair of hosts can communicate simultaneously as long as they have different source and destination addresses. This improves network efficiency.

LAN Capacity:

- **Flexibility:** The term "LAN" does not specify any constraints on the number of hosts. A LAN can accommodate a wide range of host numbers, depending on the network's size and structure.

3.2. Wide Area Network (WAN)

A WAN covers a large geographical area, such as a city, country, or even globally. It connects multiple LANs.

Characteristics:

- Slower data transfer rates than LANs.
- Often managed by multiple entities (e.g., ISPs, governments).
- Utilizes routers and public or private communication links.

Examples: The Internet itself is a WAN, Private networks connecting different company branches.

A point-to-point WAN:

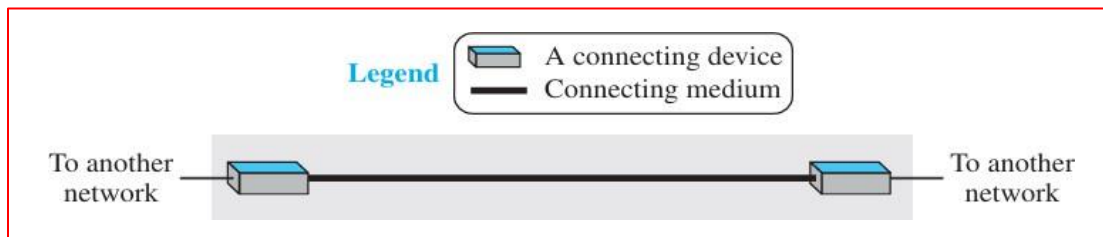


Figure 1.9: A point-to-point WAN

A Point-to-Point WAN is a network that connects two communication devices directly via transmission media, such as cables or wireless links. This type of network allows data transmission between just two endpoints, ensuring a dedicated and private communication link. Used by businesses to link branch offices or remote locations.

Transmission Media:

- **Cable:** Examples include fiber optic, coaxial, or twisted-pair cables.
- **Air:** Wireless links like microwave or satellite connections.

Advantages:

- **Dedicated Line:** Since only two devices share the link, there is minimal congestion and consistent bandwidth.
- **Security:** Limited exposure to external devices reduces the risk of unauthorized access.

A switched WAN:

A switched Wide Area Network (WAN) is a network infrastructure with multiple endpoints that are interconnected. Unlike traditional point-to-point WANs, which only connect two endpoints, a switched WAN allows for dynamic routing between several nodes through switches.

Role in Global Communication: Switched WANs are fundamental in the backbone of modern global communication systems, facilitating data transfer across vast geographical areas.

- A switched WAN is scalable and can connect many endpoints.
- The switches in the network are responsible for choosing the best path for data to travel.
- It is commonly used in large-scale, high-traffic networks like the internet.

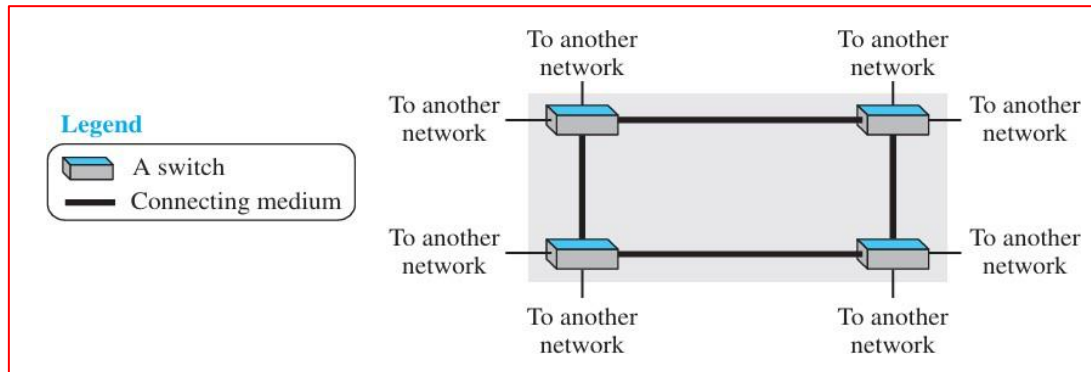


Figure 1.10: A switched WAN

Internetwork

In modern networks, it's uncommon to find a LAN (Local Area Network) or WAN (Wide Area Network) operating independently. Instead, they are often connected. When two or more networks connect, they form an internetwork, or internet.

For example, imagine a company with offices on the east and west coasts. Each office has a LAN where employees can communicate within the office. To allow communication between employees at both locations, the company leases a dedicated WAN connection from a service provider. This WAN links the two LANs, creating an internetwork, or a private internet. Now, employees from both offices can communicate with each other.

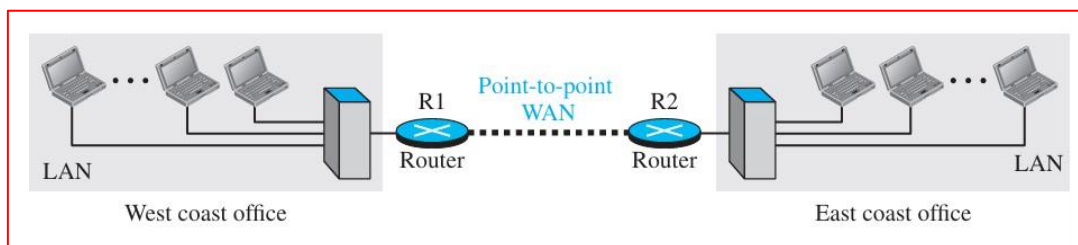


Figure 1.11: An internetwork made of two LANs and one point-to-point WAN

A heterogeneous network

A **heterogeneous network** is a system that connects various types of devices and network architectures. The term "heterogeneous" signifies diversity in the types of networks, which may

include different hardware, operating systems, and communication protocols. When combining WANs and LANs into a heterogeneous network, the WANs provide the backbone for long-distance communication, while the LANs allow for local connectivity.

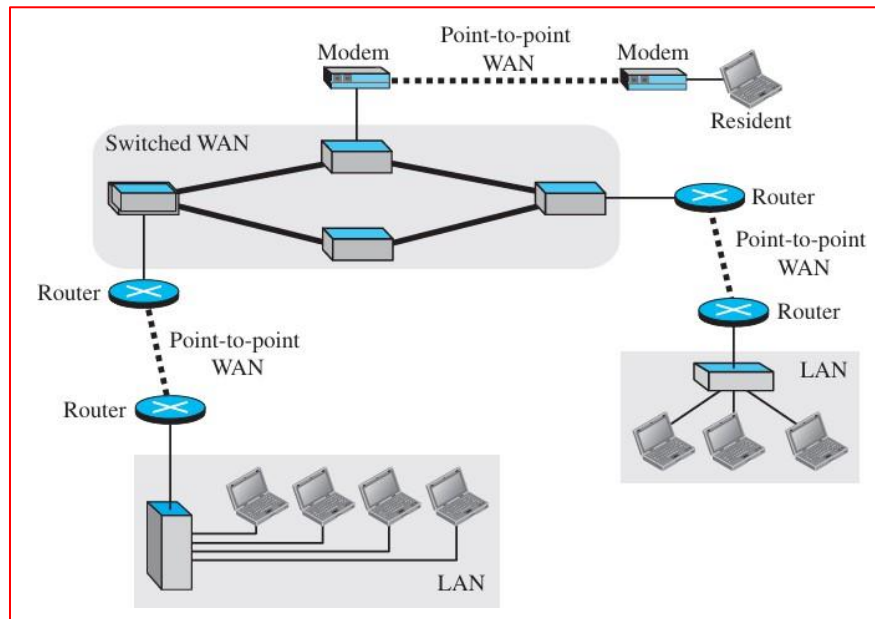


Figure 1.12: A heterogeneous network made of four WANs and three LANs

3.3. Switching

Switching is the process of forwarding data packets in a network from a source to a destination through intermediate devices called switches.

Types of Switching:

1. Circuit Switching:

In a circuit-switched network, a fixed connection (circuit) is always available between two devices, like telephones, and it can be activated or deactivated by the network switch.

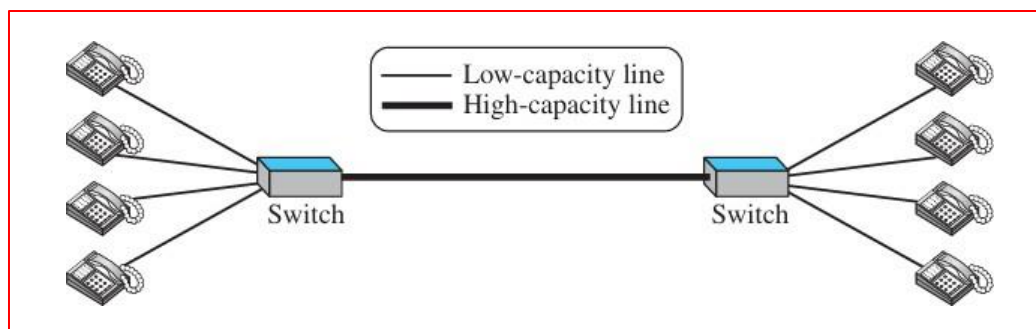


Figure 1.13: A circuit-switched network

Example Figure 1.13

- The network consists of four telephones on each side, connected to a switch.
- When a call is made, the switch connects a phone on one side to a phone on the other side.
- A thick line connects the two switches, with enough capacity to support four simultaneous voice calls. This line's capacity is shared by all phone pairs.

Two Scenarios:

1. **All phones are in use:** Four people on one side are talking to four people on the other side, using the full capacity of the line.
2. **One phone in use:** Only one pair of phones is talking, using only one-quarter of the line's capacity.

Disadvantages: A circuit-switched network is only efficient when operating at full capacity. If fewer devices are active, the unused capacity leads to inefficiency. The thick line needs four times the capacity of a single voice line to prevent call failures when all phones are in use simultaneously.

2. Packet Switching:

In a computer network, communication between devices is carried out in blocks of data called **packets**, rather than as continuous streams, as seen in phone calls. These packets are independent units, which makes it possible for network switches to store and forward them later if needed.

Example Figure 1.14

A packet-switched network where two sets of four computers are connected across two sites. In this setup, a router queues the packets and sends them when the network is ready.

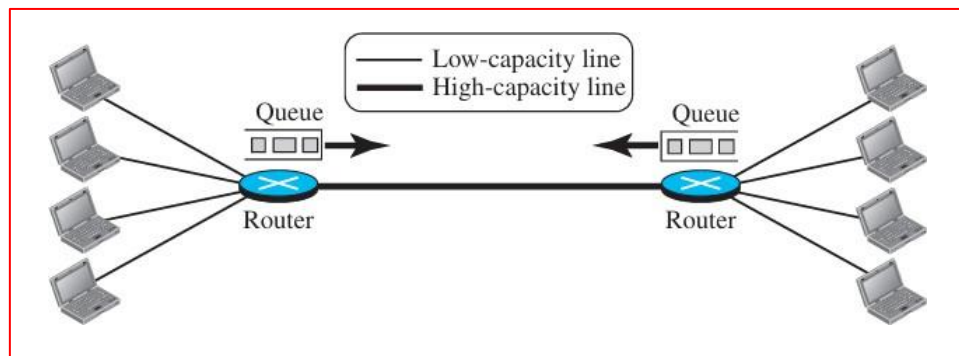


Figure 1.14: A packet-switched network

Consider a scenario where the thick line (the main connection) has twice the capacity of the smaller data lines that connect computers to routers. If only two computers (one from each site) are communicating, the packets move without delay. However, if the thick line is busy and more packets arrive, they are queued and sent in order of arrival.

Disadvantages: This demonstrates the efficiency of packet-switched networks over circuit-switched networks. However, packet-switching can introduce delays when network traffic is heavy.

3. Message Switching:

- Entire messages are stored and forwarded from one switch to another (obsolete in modern networks).

3.4.The Internet

The Internet is a global system of interconnected computer networks that use the TCP/IP protocol to link billions of devices worldwide.

Characteristics:

- Decentralized: No central controlling entity.
- Based on TCP/IP for communication.
- Enables various services such as web browsing, email, file sharing, etc.

Components:

- Clients:** Devices that request services from servers.
- Servers:** Systems that provide resources and services to clients.
- Routers and switches:** Devices that direct data traffic across the network.
- ISPs (Internet Service Providers):** Organizations that provide access to the Internet.

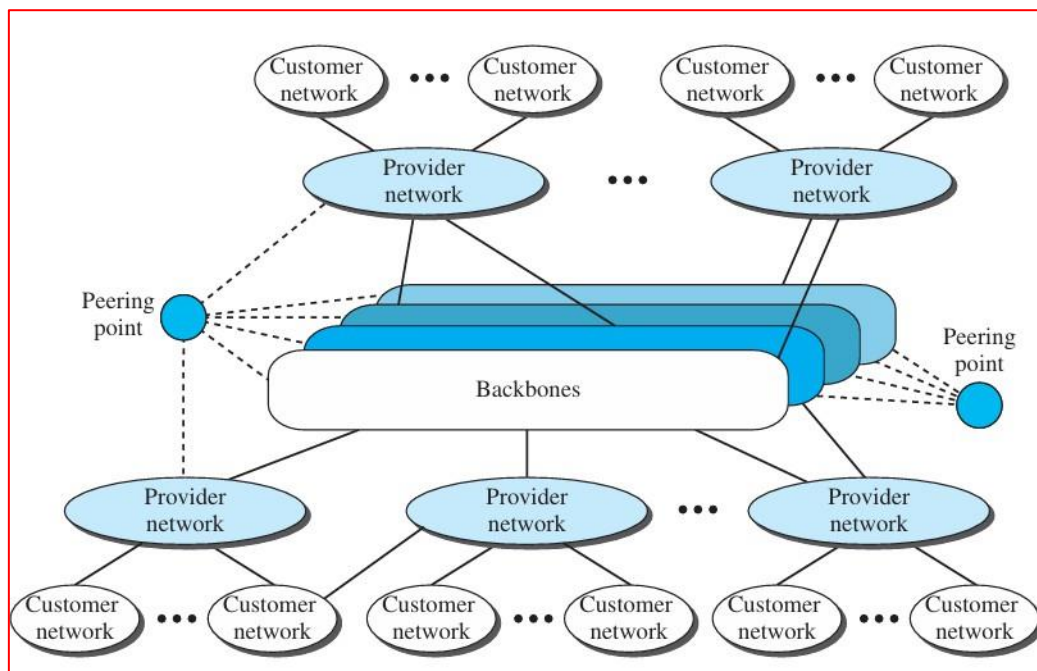


Figure 1.15: The Internet today

3.5. Accessing the Internet

Methods of Access:

1. **Dial-up:**
 - Connects to the Internet using a telephone line.
 - Low-speed, outdated method (up to 56 kbps).
2. **DSL (Digital Subscriber Line):**
 - Uses telephone lines but allows simultaneous voice and data transmission.
 - Faster than dial-up (up to 100 Mbps).
3. **Cable:**
 - Uses cable television lines to provide high-speed Internet (up to 1 Gbps).
4. **Fiber Optic:**
 - Uses light to transmit data through optical fibers.
 - Extremely high-speed (up to 10 Gbps and beyond).
5. **Satellite:**
 - Provides Internet access via satellite communication, useful in remote areas.
 - Slower than fiber or cable but offers wide coverage.
6. **Mobile Broadband:**
 - Wireless Internet access through cellular networks (3G, 4G, 5G).
 - Widely accessible but may have data limits.
7. **Wi-Fi Access:**
 - Wireless local area networking technology that allows devices to access the Internet within the range of a wireless router.
 - Common in homes, offices, and public places.

4. PROTOCOL LAYERING

Protocol layering is a design principle in computer networks where communication tasks are broken down into multiple layers. Each layer performs a specific function, and the layers work together to enable communication between devices over a network.

4.1. Scenarios

First Scenario: Simple Communication in One Layer

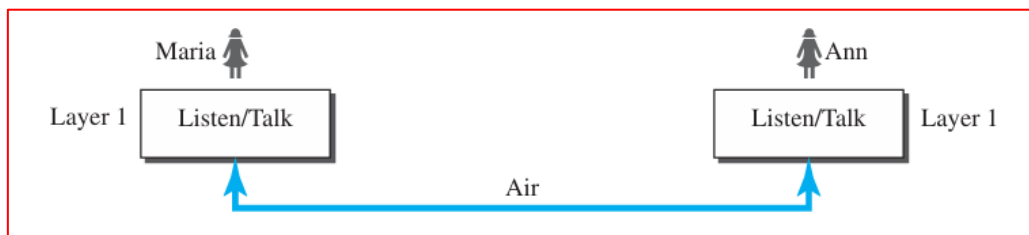


Figure 1.16: A single-layer protocol

In the first scenario, communication is straightforward and occurs in a single layer. Imagine Maria and Ann, who are neighbors and share many common interests. Their interaction happens face-to-face, in the same language, as depicted in Figure 1.16. Despite its simplicity, certain rules govern their communication.

Second Scenario: A three-layer protocol

In this scenario, Ann is offered a promotion requiring her to relocate far from her friend Maria. Despite the distance, they wish to continue their communication to collaborate on an innovative project for their future retirement business. To maintain the confidentiality of their exchange, they decide to use a secure encryption/decryption technique. This technique ensures that their letters remain unreadable to unauthorized individuals.

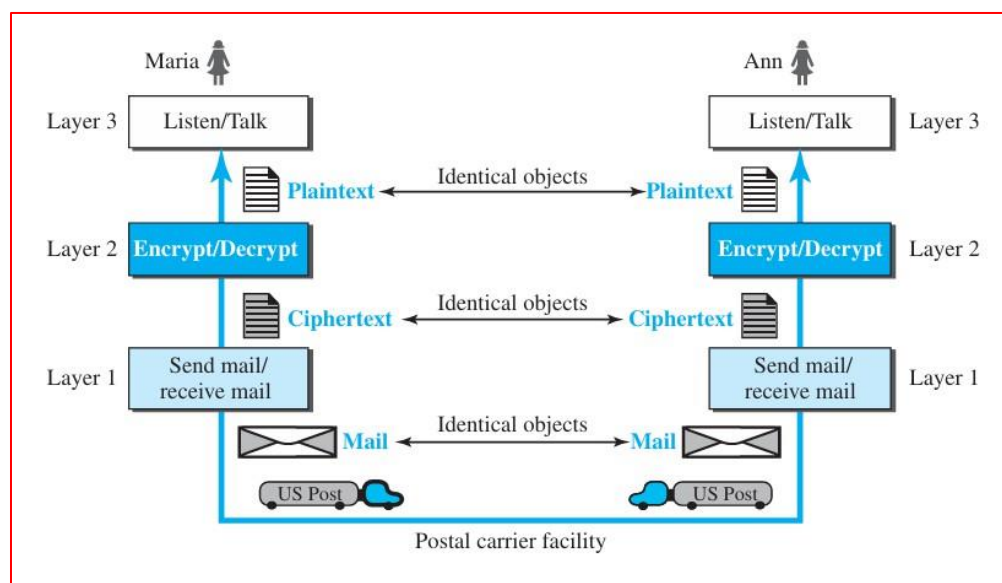


Figure 1.17 A three-layer protocol

The communication between Ann and Maria can be visualized as taking place in three distinct layers:

1. **Third Layer (Plaintext Creation):** Maria speaks to the third-layer machine, which listens and generates the plaintext (the original letter). This plaintext is then handed over to the second-layer machine.
2. **Second Layer (Encryption/Decryption):** The second-layer machine encrypts the plaintext, creating ciphertext (the encrypted letter). This ciphertext is then passed to the first-layer machine.
3. **First Layer (Mailing):** The first-layer machine puts the ciphertext in an envelope, adds the sender's and receiver's addresses, and mails it.

Protocol layering is useful in various scenarios, such as:

- **Data transmission** over the internet where multiple types of data (emails, videos, web pages) are transmitted using different protocols at each layer.
- **Device communication** between a computer and a printer, where layered protocols manage data transfer and error detection.
- **Telecommunication systems**, where layered protocols ensure voice signals are transmitted efficiently and correctly.
- **Multimedia streaming**, where protocols manage data buffering, synchronization, and error correction to deliver seamless audio and video streams.

Advantages of Protocol Layering:

1. **Modularity:** Each layer is independent, allowing for changes in one layer without affecting others. For example, if Ann and Maria decide that the encryption needs to be upgraded, they can replace only the second-layer machine without altering the other layers.
2. **Service Separation:** Protocol layering separates the services from their implementation. Maria could perform the tasks of the first layer herself if needed, as long as she provides the required services.
3. **Intermediate Systems:** In more complex networks, intermediate systems only need to handle specific layers, making the overall system less complex and less expensive.

Disadvantages of Protocol Layering:

1. **Complexity in Layer Integration:** While modularity offers flexibility, it can add complexity. A single machine performing all tasks could simplify the setup, but if there are issues, the entire system might need to be replaced rather than just one layer.

4.2. Principles of Protocol Layering

First Principle: Bidirectional Communication

To achieve bidirectional communication in protocol layering, each layer must be designed to handle two complementary tasks, one for each direction of communication. For example:

Third Layer: Responsible for listening in one direction and transmitting in the other.

Second Layer: Handles encryption in one direction and decryption in the other.

First Layer: Manages sending and receiving mail.

Each layer must be capable of performing its specific function for both incoming and outgoing data.

Second Principle: Identical Objects

In protocol layering, the objects processed by each layer at both communicating sites must be identical. For instance:

Layer 3: The object should be a plaintext letter at both sites.

Layer 2: The object should be a ciphertext letter at both sites.

Layer 1: The object should be a piece of mail at both sites.

Consistency in the objects at each layer ensures proper communication and processing across different layers.

4.3. Logical Connections

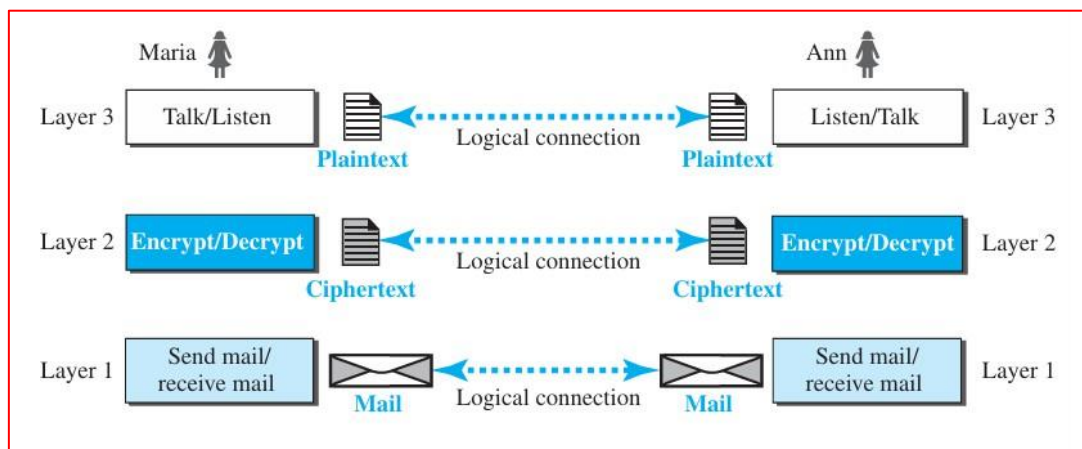


Figure 1.18: Logical connection between peer layers

This means there is communication from one layer to the next. Imagine that Maria and Ann view each layer as having a logical (or imaginary) connection that allows them to transmit objects created at that layer. Understanding these logical connections will enhance your comprehension of layering in data communication and networking.

In a layered model, **logical connections** refer to virtual connections established between the same layers in different devices. For example:

- The application layers of two computers communicate as though they are directly connected, even though physical transmission occurs across multiple lower layers.
- Logical connections provide the abstraction necessary to hide the complexities of the underlying transmission methods, giving the appearance that communication occurs at a single layer.

Logical connections ensure that communication protocols at each layer, such as TCP (Transport Layer) or HTTP (Application Layer), interact correctly without the user needing to understand the intricacies of network transmission.

5. TCP/IP PROTOCOL SUITE

TCP/IP is the protocol suite widely used for communication on the Internet today. It is a collection of protocols organized into different layers, where each layer handles specific tasks, making it easier to manage complex communication processes. TCP/IP is a **hierarchical protocol**, meaning that each higher-level protocol depends on the services provided by one or more lower-level protocols. This layered structure ensures efficient and organized communication across networks. However, today, it is more commonly represented as a **five-layer model**. These layers work together to ensure smooth data transmission over the network.

5.1. Layered Architecture

To understand how the layers in the TCP/IP protocol suite work during communication between two hosts, let's consider a small network composed of three local area networks (LANs), each connected by a link-layer switch. These LANs are also interconnected through a router. Figure 1.20 illustrates this setup.

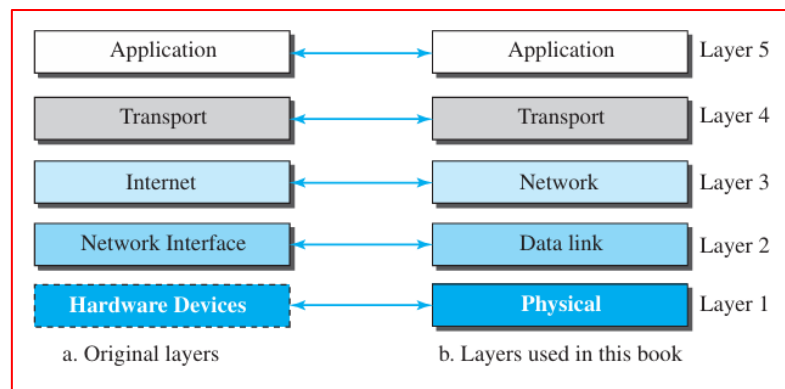


Figure 1.19: Layers in the TCP/IP protocol suite

In this scenario, imagine that **Host A** (the source) communicates with **Host B** (the destination). The communication process involves five devices:

1. Source Host (Computer A)
2. Link-layer switch in LAN 1
3. Router
4. Link-layer switch in LAN 2
5. Destination Host (Computer B)

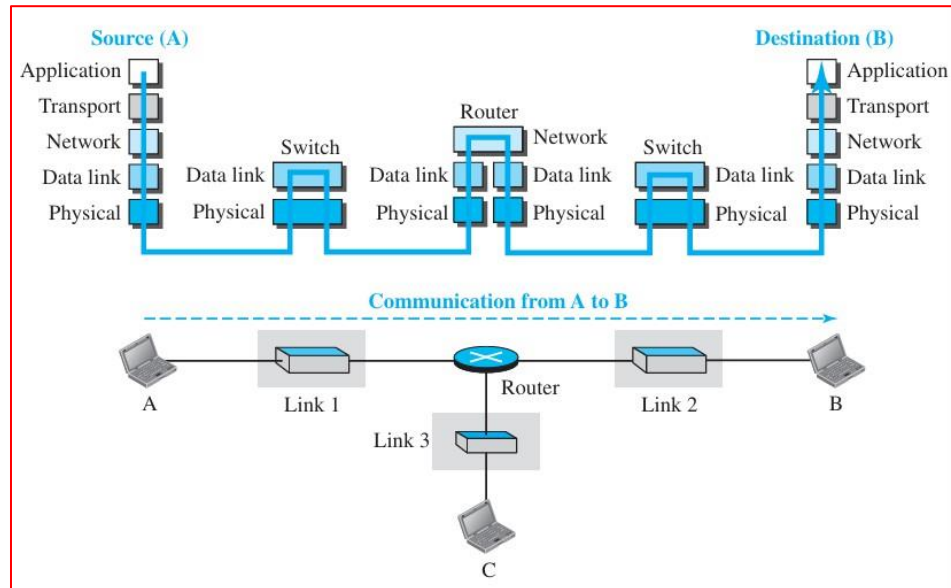


Figure 1.20: Communication through an internet

Each of these devices operates at different layers of the TCP/IP protocol stack, depending on its role in the network:

1. Hosts (Source and Destination)

Both **Host A** and **Host B** are involved in all five layers of the TCP/IP model:

- **Application Layer:** The source host (Host A) creates a message at the application layer and sends it down through the stack.
- **Transport Layer:** The message is passed to the transport layer, which ensures reliable delivery.
- **Network Layer:** At the network layer, the message is encapsulated into packets for transmission across the network.
- **Data Link Layer:** The packets are then prepared for transmission over the physical network in the data-link layer.
- **Physical Layer:** Finally, the message is sent through the physical medium (wires, cables, etc.) to reach the destination host.

At the destination, **Host B** receives the message at the physical layer and passes it up through the layers until it reaches the application layer for processing.

2. Router

A **router** plays a different role and operates at three layers of the TCP/IP model:

- **Network Layer:** The router's primary function is routing packets across networks. It forwards packets based on their destination IP address.
- **Data Link Layer & Physical Layer:** A router is connected to multiple links, and each link may use a different data-link and physical-layer protocol. For instance, if a packet arrives from **LAN 1** (Link 1) using one set of protocols, the router must handle it and forward it to **LAN 2** (Link 2) using another set of protocols.

Importantly, the router does not deal with the transport or application layers, as its role is solely to move packets between networks.

3. Link-Layer Switch

A **link-layer switch** operates only at the data-link and physical layers:

- **Data Link Layer:** The switch processes the data frames and ensures they are forwarded to the correct device within the same LAN.
- **Physical Layer:** The switch forwards the data through the physical medium.

Unlike routers, link-layer switches do not need to handle different sets of protocols for different links. They operate within a single LAN, using a single protocol set for the data-link and physical layers.

5.2. Layers in the TCP/IP Protocol Suite

TCP/IP protocol suite functions and responsibilities of each layer.

Understanding the Logical Connections Between Layers

To grasp the role of each layer, it's helpful to visualize the **logical connections** between them. Figure 1.21 in the book illustrates these connections in a simple internet model.

- **End-to-End vs. Hop-to-Hop Duties:**
 - The **application, transport, and network layers** are responsible for **end-to-end** communication, meaning they manage data from one end device to the other across the network.
 - The **data-link and physical layers**, on the other hand, handle communication on a **hop-to-hop** basis, where each "hop" refers to a host or router.

This distinction is key: the top three layers operate across the entire internet, while the lower two layers manage communication on individual network segments or "links."

Data Units and Layer Responsibilities

Another important way to understand these connections is by considering the **data units** created at each layer.

- In the **top three layers**, the data units (referred to as packets) are **not modified** by routers or link-layer switches.
- In the **bottom two layers**, however, the packet created by the host can be modified by routers but **not** by link-layer switches.

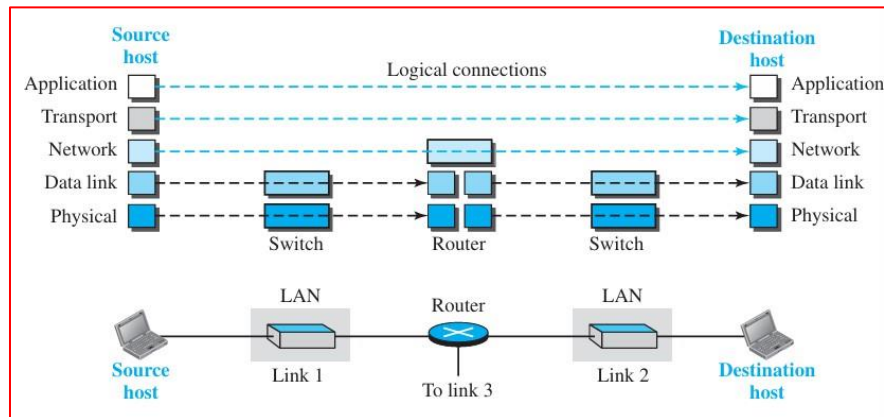


Figure 1.21: Logical connections between layers of the TCP/IP protocol suite

Figure 1.22 shows a second principle of protocol layering: identical objects exist below each layer for connected devices.

- At the **network layer**, even though there's a logical connection between two hosts, a router might fragment packets into smaller units.
- The link between two hops does not alter these packets.

This layering approach allows for a structured, predictable method of managing data as it moves across the network.

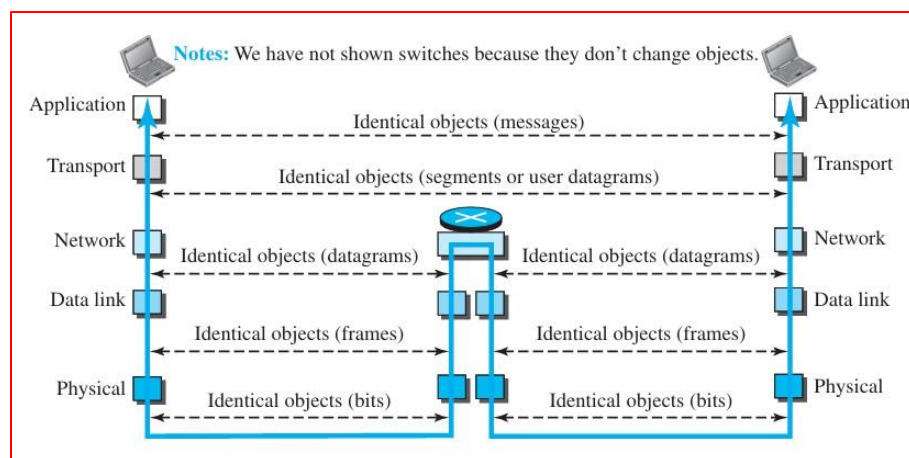


Figure 1.22: Identical objects in the TCP/IP protocol suite

5.3. Description of Each Layer

Physical Layer:

Role of the Physical Layer:

- The physical layer focuses on transmitting the bits from the data-link layer as electrical or optical signals over a physical medium (such as cables or wireless channels).
- It converts bits from the data-link layer into signals that can travel through the medium connecting the devices.

Transmission Media:

- The physical connection between two devices is made through a transmission medium, which can be either **cables** (like twisted-pair, fiber-optic cables) or **air** (wireless communication).
- The **transmission medium** does not directly carry bits; instead, it transmits **signals** (electrical or optical) that represent the bits.

Logical Communication:

- Despite being the lowest layer, the physical layer enables **logical communication** between devices by ensuring that signals are accurately transmitted and interpreted.
- From a logical perspective, the unit of data at this layer is the **bit**, even though the actual signals sent over the medium are different in nature.

Signal Transformation:

- The physical layer converts bits into appropriate forms of signals (either electrical or optical) depending on the medium used.
- Protocols at this layer define how these bits are transformed into signals to be carried over the medium.

Data-link Layer:

In an internet, multiple links (LANs and WANs) are connected by routers. There are often multiple paths that a datagram can take from the host to its destination. Routers are responsible for determining the most efficient route. Once the next link is chosen by a router, the **data-link layer** manages the process of transmitting the datagram across that link.

These links can vary widely and include:

- Wired LANs with link-layer switches
- Wireless LANs
- Wired WANs
- Wireless WANs

Each type of link may use different protocols, but the **data-link layer** has the essential role of ensuring that the packet is successfully transmitted across any type of link.

The **TCP/IP model** does not mandate a specific protocol for the data-link layer. Instead, it accommodates a wide range of both standard and proprietary protocols. Any protocol that can encapsulate a datagram and move it across a link is considered acceptable.

At the data-link layer, the datagram is encapsulated into a packet, referred to as a **frame**.

- Some link-layer protocols offer **both error detection and correction**, ensuring that transmitted data is free from errors.

Network Layer:

The network layer is crucial for establishing communication between the source and destination computers. It manages **host-to-host communication** across multiple devices (routers) in the path. Each router ensures that packets are directed along the most efficient route. The main role of the network layer is to enable communication between hosts and determine the best path for data transmission.

Internet Protocol (IP)

The primary protocol of the network layer is the **Internet Protocol (IP)**, which defines the structure and format of data packets, known as **datagrams** at this level. IP also determines the addressing system, ensuring each packet is sent from its source to its destination. The forwarding of packets from router to router is a fundamental part of IP's function.

- **Connectionless Protocol:** IP is connectionless, meaning it does not manage flow control, error control, or congestion control. These features are handled by the transport layer if needed by an application.
- **Routing Protocols:** While IP is responsible for actual routing, protocols like **unicast** (one-to-one) and **multicast** (one-to-many) create forwarding tables to assist routers with routing decisions.

Auxiliary Protocols Supporting IP

Several additional protocols work alongside IP to assist in the routing and delivery process:

- **ICMP (Internet Control Message Protocol):** Helps IP report issues encountered during packet routing.
- **IGMP (Internet Group Management Protocol):** Supports IP with multicast communication.
- **DHCP (Dynamic Host Configuration Protocol):** Assigns network-layer addresses to hosts.
- **ARP (Address Resolution Protocol):** Converts a network-layer address into a link-layer address for communication with a specific host or router.

These protocols ensure efficient routing and network management, enabling seamless host-to-host communication in complex networks.

Transport Layer:

The **Transport Layer** plays a crucial role in ensuring **end-to-end communication between hosts** in a network. It handles the transfer of data between the application layer on one device and the corresponding application layer on another device, making sure that messages are transmitted reliably and efficiently.

Key Functions:

- **Encapsulation and Transmission:** The transport layer at the source host takes messages from the application layer, encapsulates them into transport layer packets (called *segments* in TCP or *user datagrams* in UDP), and transmits them to the destination transport layer.
- **End-to-End Communication:** Even though the application layer provides an end-to-end connection, the transport layer separates tasks, handling data transmission independently from the application. This separation allows flexibility by providing multiple protocols suited for different needs.

Protocols in the Transport Layer:

1. **Transmission Control Protocol (TCP):**
 - **Connection-Oriented:** TCP establishes a connection between two hosts before transferring data. It creates a virtual "pipe" for sending a continuous stream of bytes.
 - **Flow Control:** Ensures that the data sent matches the receiving capability of the destination, preventing data loss due to overwhelming the receiver.
 - **Error Control:** TCP checks for errors in data transmission and ensures that any corrupted segments are retransmitted.
 - **Congestion Control:** Helps avoid network congestion by adjusting the transmission rate based on the current network conditions.
2. **User Datagram Protocol (UDP):**
 - **Connectionless:** Unlike TCP, UDP does not establish a connection before sending data. Each user datagram is sent independently, without reference to previous or subsequent datagrams.
 - **Simplicity:** UDP is lightweight, with minimal overhead. However, it does not provide flow control, error control, or congestion control.
 - **Use Case:** Best suited for applications that need to send small, independent messages, where speed is critical and retransmission (as in TCP) would be inefficient.
3. **Stream Control Transmission Protocol (SCTP):**
 - **Designed for Modern Applications:** SCTP is tailored for emerging multimedia applications, offering more advanced features suited to high-demand data transmissions.

By providing these different protocols, the transport layer allows application programs to choose the one that best suits their specific requirements, whether they prioritize reliability, simplicity, or speed.

Application Layer:

The application layer in a network facilitates end-to-end communication between two application layers. It appears as though a direct bridge exists between them, but in reality, communication passes through all layers of the networking model.

At this layer, the communication happens between two processes (programs running on separate machines). One process sends a request, and the other process responds. This process-to-process communication is the core responsibility of the application layer. While many predefined protocols exist at this level, users can also create their own custom processes to communicate across hosts.

Key protocols in the application layer include:

- **HTTP (Hypertext Transfer Protocol):** Used for accessing the World Wide Web (WWW).
- **SMTP (Simple Mail Transfer Protocol):** The primary protocol for sending and receiving email.
- **FTP (File Transfer Protocol):** Enables the transfer of files between hosts.
- **TELNET and SSH (Secure Shell):** Facilitate remote access to a site.
- **SNMP (Simple Network Management Protocol):** Allows administrators to manage Internet resources both globally and locally.
- **DNS (Domain Name System):** Helps find the network-layer address (IP address) of a computer for other protocols.
- **IGMP (Internet Group Management Protocol):** Collects information on group membership for multicast communication.

5.4. Encapsulation and Decapsulation

Encapsulation and decapsulation are key concepts in protocol layering within the Internet. These processes occur at both the source and destination hosts, as well as at intermediary devices like routers.

I. Encapsulation at the Source Host

At the source host, the encapsulation process involves the following steps:

1. **Application Layer:** The data generated by the application is called a *message*. This message typically doesn't have a header or trailer, but if it does, the whole unit is still referred to as a message. This message is passed to the transport layer.
2. **Transport Layer:** The transport layer treats the message as a payload. It adds a transport layer header, which includes details like the source and destination application program identifiers and information necessary for tasks such as flow control, error control, or

congestion control. The resulting packet is known as a *segment* (in TCP) or a *user datagram* (in UDP). This transport layer packet is then passed to the network layer.

3. **Network Layer:** At this layer, the transport layer packet is treated as the payload. A network layer header is added, which contains the source and destination IP addresses, as well as additional information for error checking and fragmentation. The final packet is referred to as a *datagram*, which is then passed to the data-link layer.
4. **Data-Link Layer:** The network layer packet is encapsulated in a data-link layer frame. A header is added here, which includes the addresses of the sending host and the next hop (e.g., a router). The frame is then passed to the physical layer for transmission over the network.

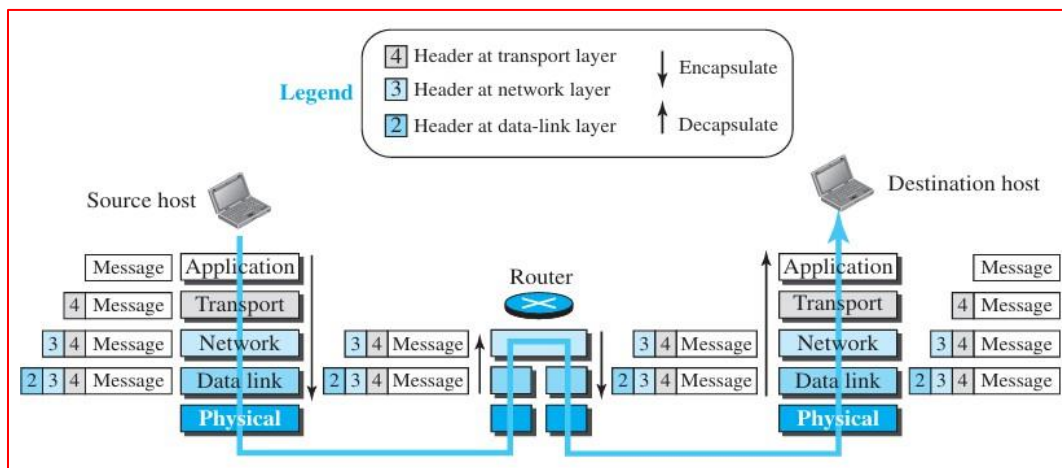


Figure 1.23: Encapsulation/Decapsulation

II. Encapsulation and Decapsulation at the Router

Routers perform both encapsulation and decapsulation, as they are connected to multiple network links. Here's how this works:

1. **Decapsulation at Data-Link Layer:** When the router receives a frame, the data-link layer extracts the *datagram* from the frame and passes it to the network layer.
2. **Network Layer Processing:** The router's network layer examines the source and destination addresses within the datagram header. Using its forwarding table, the router determines the next hop for the datagram. The content of the datagram is not altered unless it requires fragmentation to pass through the next link. Once processed, the datagram is passed to the data-link layer of the next link.
3. **Encapsulation at Data-Link Layer:** The datagram is encapsulated into a new frame suitable for the next link, and the frame is passed to the physical layer for transmission.

III. Decapsulation at the Destination Host

At the destination host, the reverse process of encapsulation—decapsulation—occurs:

1. **Data-Link Layer:** The frame is received, and the data-link layer removes its header to extract the network layer *datagram*.
2. **Network Layer:** The network layer removes its header to extract the transport layer *segment* or *user datagram*.
3. **Transport Layer:** The transport layer removes its header to deliver the original *message* to the application layer.

During decapsulation, error checking is performed at each layer to ensure data integrity. The process continues until the original message reaches the application layer, where it can be used by the application program.

5.5. Addressing in Network

In the context of the Internet and network protocol layering, **addressing** is a key concept that ensures logical communication between pairs of layers. For any communication between two parties, there must be two essential addresses: a **source address** and a **destination address**. Although it might seem like each layer of the protocol model would require a separate pair of addresses, in practice only four are needed, as the **physical layer** does not use addresses. The physical layer deals with bits, and individual bits do not have addresses.

Addressing at Different Layers

Each layer in the protocol stack uses specific types of addresses, which relate to the type of communication and the name of the data unit used at that layer. Here's a breakdown:

1. **Application Layer:**
 - At this layer, addresses are typically **names**, such as a domain name (e.g., `someorg.com`) or an email address (e.g., `somebody@coldmail.com`). These names represent the service or resource being accessed.
 - **Packet name:** *Message*.
2. **Transport Layer:**
 - Addresses here are called **port numbers**. Port numbers identify the specific application-layer programs (e.g., web server-Port 80, email client - Port 25) on the source and destination devices. Each running program has its own port number, ensuring that data reaches the correct application.
 - **Packet name:** *Segment* (for TCP) or *User Datagram* (for UDP).

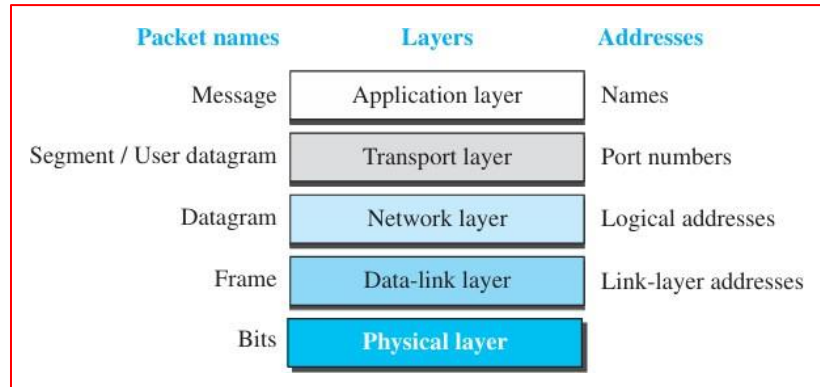


Figure 1.24: Addressing in the TCP/IP protocol suite

3. Network Layer:

- The network layer uses **global addresses**, which are known as **IP addresses (45.113.122.159)**. These uniquely identify a device's connection to the Internet and allow data to be routed across networks.
- **Packet name:** *Datagram*.

4. Link Layer:

- At this layer, **link-layer addresses** (also known as **MAC addresses 78-45-C4-29-17-E8**) are used. These are local addresses that uniquely identify specific devices on a network (such as a LAN or WAN), ensuring that data reaches the correct physical machine.
- **Packet name:** *Frame*.

5.6. Multiplexing and Demultiplexing

In the TCP/IP protocol suite, multiplexing and demultiplexing are essential concepts for managing data flow between different layers of the protocol stack.

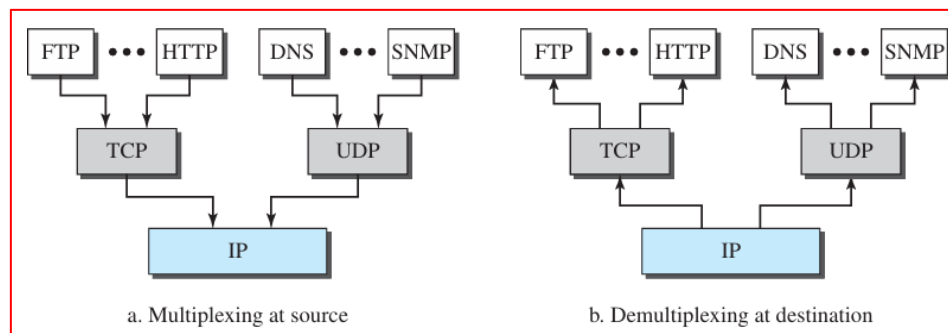


Figure 1.25: Multiplexing and demultiplexing

Multiplexing at the Source: Multiplexing occurs when a protocol at a certain layer can handle multiple types of data from higher layers. For example, at the transport layer, protocols like TCP and UDP can receive messages from various application-layer protocols (such as FTP, HTTP, DNS, and SNMP). The transport layer encapsulates these messages into segments and adds a header field to indicate which application-layer protocol the data belongs to.

Demultiplexing at the Destination: Demultiplexing happens at the destination when a protocol needs to deliver the data to the appropriate higher-layer protocol. At the transport layer, TCP or UDP will examine the header to determine which application-layer protocol should receive the data. Similarly, at the network layer, IP can handle segments from TCP or datagrams from UDP, and it can also process packets from other protocols like ICMP or IGMP. At the data-link layer, a frame may carry payloads from various protocols, such as IP or ARP.

Illustration:

- **At the Transport Layer:** UDP or TCP encapsulates data from application-layer protocols (e.g., FTP, HTTP, DNS, SNMP).
- **At the Network Layer:** IP handles segments from TCP, datagrams from UDP, and packets from other protocols.
- **At the Data-Link Layer:** Frames may include payloads from IP or other protocols like ARP.

6. TRANSMISSION MEDIA

Transmission media are essentially the physical substances through which information travels. They operate at a level below the physical layer of the OSI model, often referred to as "layer zero." Essentially, transmission media can be considered as a fundamental layer that directly interfaces with the physical layer.

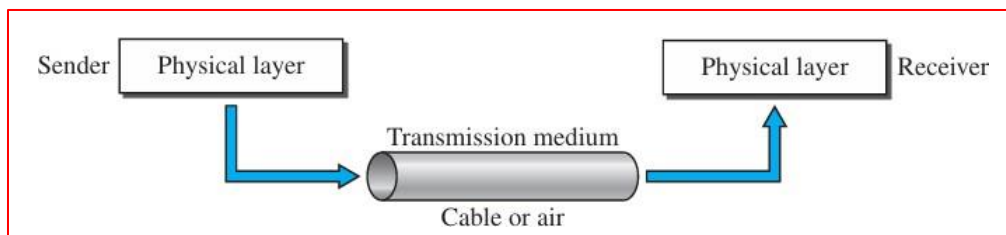


Figure 1.26: Transmission medium and physical layer

Data Communications Examples:

- Transmission media include free space, metallic cables, or fiber-optic cables.
- The information transmitted is usually in the form of signals generated from data.

- **Telegraph (19th Century):** Invented by Morse, it was an early long-distance communication technology using metallic wires, but was slow.
- **Telephone (1869):** Allowed voice transmission over long distances via metallic cables, though communication quality was poor due to inferior wires.
- **Wireless Communication (1895):** Hertz transmitted high-frequency signals, and Marconi later improved this by sending messages across the Atlantic Ocean.

- **Improved Metallic Media:** Development of twisted-pair and coaxial cables has enhanced data transmission.
- **Optical Fibers:** The advent of fiber-optic cables has significantly increased data transmission rates.
- **Efficient Use of Free Space:** Advances in modulation and multiplexing have optimized the use of free space (air, vacuum, water) for communication.

Electromagnetic Spectrum:

- **Signals and Electromagnetic Energy:** Computers and telecommunication devices transmit data as electromagnetic energy, which includes power, radio waves, infrared, visible light, ultraviolet light, and X-rays.
- **Spectrum Usability:** Not all parts of the electromagnetic spectrum are suitable for telecommunications. Only certain portions are used, and the types of media to harness these are limited.

Categories of Transmission Media:

1. **Guided Media:** These include twisted-pair cables, coaxial cables, and fiber-optic cables.
2. **Unguided Media:** This refers to free space, such as air and vacuum.

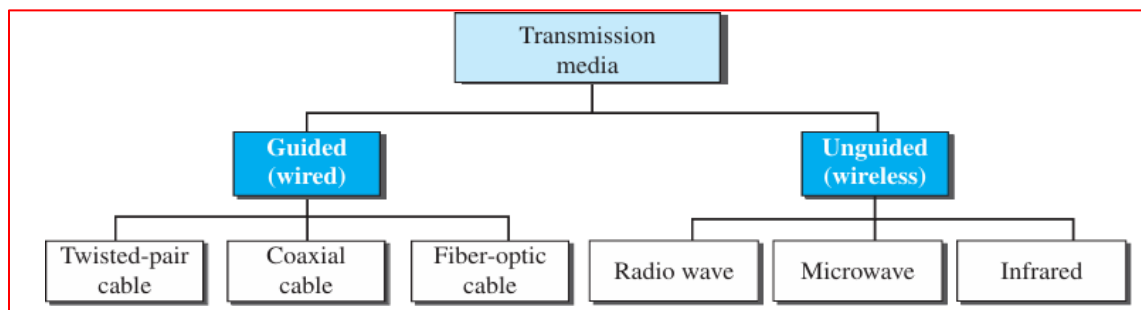


Figure 1.27: Classes of transmission media

6.1. Guided Media

Guided media are types of communication channels that provide a specific path for signals to travel from one device to another. These include:

1. **Twisted-Pair Cable:** This type of cable consists of pairs of insulated copper wires twisted together. The twisting helps reduce electromagnetic interference and maintains signal quality.
2. **Coaxial Cable:** Coaxial cables have a central conductor, an insulating layer, a metallic shield, and an outer insulating layer. This structure helps to protect the signal from interference and allows for high-speed data transmission.
3. **Fiber-Optic Cable:** Unlike twisted-pair and coaxial cables, fiber-optic cables use light signals to transmit data. They consist of thin strands of glass or plastic that carry light pulses over long distances with minimal signal loss.

6.2. Twisted-Pair Cable

A twisted pair cable consists of two insulated copper conductors twisted together. Each wire in the pair serves a different function: one carries the signal to the receiver, and the other acts as a ground reference. The receiver processes the difference between the two wires to retrieve the signal.

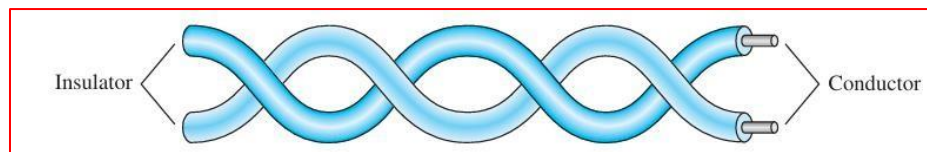


Figure 1.28: Twisted-pair cable

Noise and Interference

Twisted pair cables are designed to minimize the impact of interference (noise) and crosstalk. When the wires are parallel, noise or crosstalk can affect each wire differently due to their varying distances from the sources of interference. By twisting the wires, the cable maintains a balance. In each twist, the relative positions of the wires to the noise source change, helping to ensure that both wires experience similar levels of interference. This twisting reduces the impact of unwanted signals, as the receiver calculates the difference between the wires, canceling out most of the noise.

Shielded vs. Unshielded Twisted-Pair Cables

- **Unshielded Twisted-Pair (UTP):** The most common type used in communications, UTP cables do not have additional shielding. They are less expensive and less bulky but can be more susceptible to interference.
- **Shielded Twisted-Pair (STP):** STP cables have an additional metal foil or braided mesh covering each pair of conductors. This shielding reduces interference and improves signal

quality but makes the cables bulkier and more costly. STP is primarily used by IBM and is less common outside of their applications.

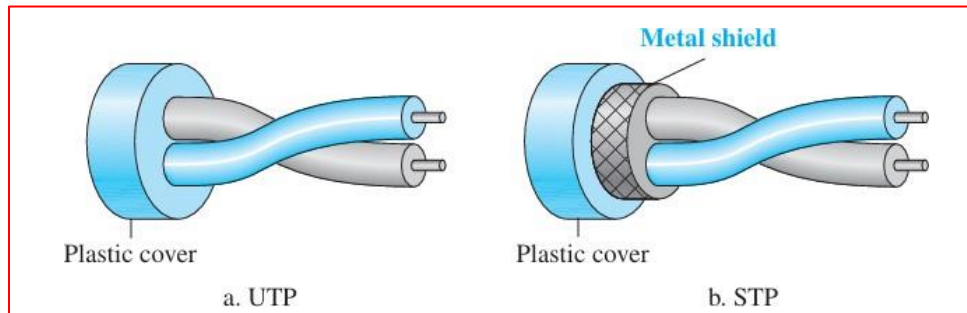


Figure 1.29: UTP and STP cables

Categories of UTP Cables

The Electronic Industries Association (EIA) classifies UTP cables into seven categories, with Category 1 being the lowest quality and Category 7 being the highest. Each category is suitable for specific applications, and the standards help ensure the cable meets certain performance criteria.

Connectors

The RJ45 connector is the most common connector for UTP cables. It is a keyed connector, meaning it can only be inserted in one direction, which ensures a proper connection.

Table 1.1: Categories of unshielded twisted-pair cables

Category	Specification	Data Rate (Mbps)	Use
1	Unshielded twisted-pair used in telephone lines	2	Telephone
2	Unshielded twisted-pair originally used in T1 lines	10	T1 Lines
3	Improved Category 2 used in LANs	20	LANs
4	Improved Category 3 used in Token Ring networks	100	Token Ring Networks
5	Cable wire is normally 24 AWG with a jacket and outside sheath	125	LANs
5E	An extension of Category 5 with additional features to minimize crosstalk and electromagnetic interference	125	LANs
6	New category with matched components from the same manufacturer; cable tested at a 200-Mbps data rate	200	LANs
7	Sometimes called SSTP (Shielded Screen Twisted-Pair); each pair is wrapped in helical metallic foil followed by a metallic foil shield	600	LANs

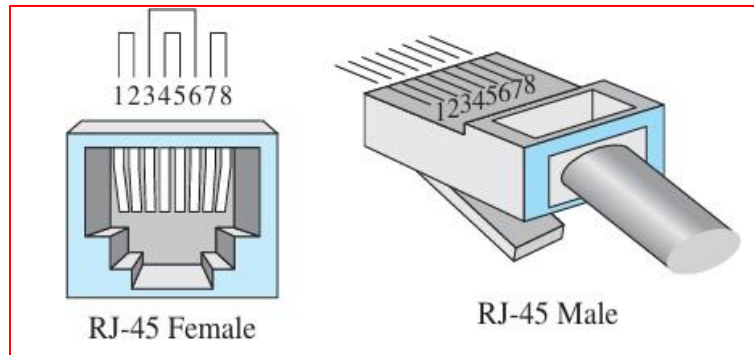


Figure 1.30: UTP connector

Performance

The performance of twisted-pair cables is often assessed by measuring attenuation (signal loss) in relation to frequency and distance. Although twisted-pair cables can handle a broad range of frequencies, attenuation increases significantly at frequencies above 100 kHz. Attenuation is measured in decibels per kilometer (dB/km), and higher frequencies result in greater signal loss.

Applications

Twisted-pair cables are widely used in various applications:

- **Telephone Lines:** Used for voice and data transmission in the local loop connecting subscribers to telephone offices.
- **DSL Lines:** Provide high-data-rate connections by utilizing the high bandwidth of UTP cables.
- **Local-Area Networks (LANs):** Employed in networks such as 10Base-T and 100Base-T for data transmission.

6.3. Coaxial Cable

Coaxial cable, often referred to as coax, is designed to carry high-frequency signals, unlike twisted-pair cables. It consists of the following parts:

- **Central Core Conductor:** A solid or stranded copper wire, responsible for carrying the signal.
- **Insulating Sheath:** Surrounds the core conductor, separating it from the outer conductor.
- **Outer Conductor:** Made of metal foil, braid, or both, which serves two purposes:
 - Acts as a shield against external noise.
 - Functions as the second conductor to complete the circuit.
- **Additional Insulation:** Encases the outer conductor for further protection.
- **Plastic Cover:** Provides external protection for the entire cable.

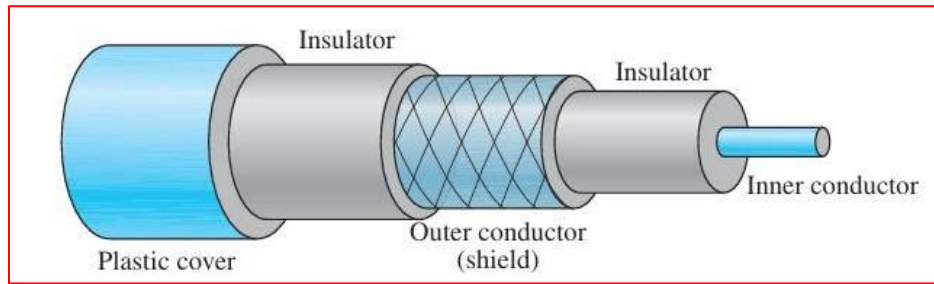


Figure 1.31 Coaxial cable

This design helps coaxial cables transmit signals with less interference and at higher frequencies compared to twisted-pair cables.

Coaxial Cable Standards (RG Ratings)

Coaxial cables are classified based on Radio Government (RG) ratings. Each RG number specifies certain physical characteristics of the cable, such as:

- **Wire Gauge:** Thickness of the inner conductor.
- **Insulation:** Thickness and material of the inner insulator.
- **Shielding:** Design and construction of the outer conductor.
- **Outer Casing:** Size and type of the protective cover.

These RG-rated cables are suited for different specialized applications, as indicated by their specifications.

Table 1.2: RG-rated cables are suited for different specialized applications.

Category	Use
RG-59 75 Ω	Cable TV
RG-58 50 Ω	Thin Ethernet
RG-11 50 Ω	Thin Ethernet

Coaxial Cable Connectors

To connect coaxial cables to devices, special connectors are required. The most common connector type is the **Bayonet Neill-Concelman (BNC)** connector. There are different variations of this connector:

- **BNC Connector:** Used to connect the cable to a device (e.g., a TV).
- **BNC T Connector:** Often used in Ethernet networks to split the signal to multiple devices.
- **BNC Terminator:** Placed at the end of the cable to prevent signal reflection and ensure stable transmission.

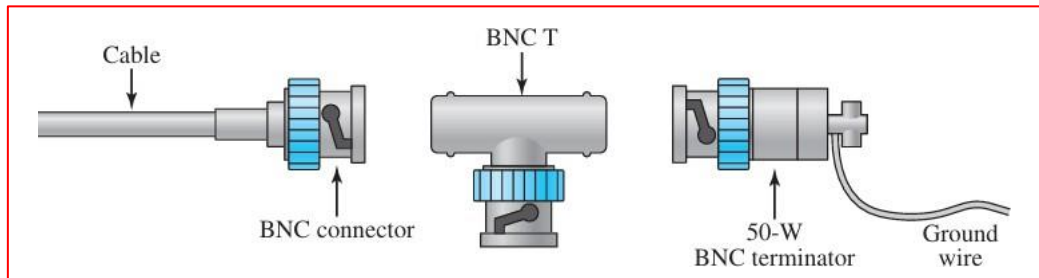


Figure 1.32: BNC connectors

Performance of Coaxial Cables

Coaxial cables provide higher bandwidth compared to twisted-pair cables, allowing them to carry more data. However, their **attenuation** (signal weakening) is greater, meaning the signal degrades over distance and requires frequent use of **repeaters** to maintain signal strength.

Applications

- **Analog Telephone Networks:** Coaxial cables were used to carry large amounts of voice signals, with one cable supporting up to 10,000 voice channels.
- **Digital Telephone Networks:** Coaxial cables were capable of transmitting digital data at speeds up to 600 Mbps.

Today, **fiber optic cables** have largely replaced coaxial cables in telephone networks due to their superior performance.

6.4. Fiber-Optic Cable

A **fiber-optic cable** is a medium made of glass or plastic that transmits signals as light. To understand how it works, it's important to first grasp the nature of light.

Properties of Light

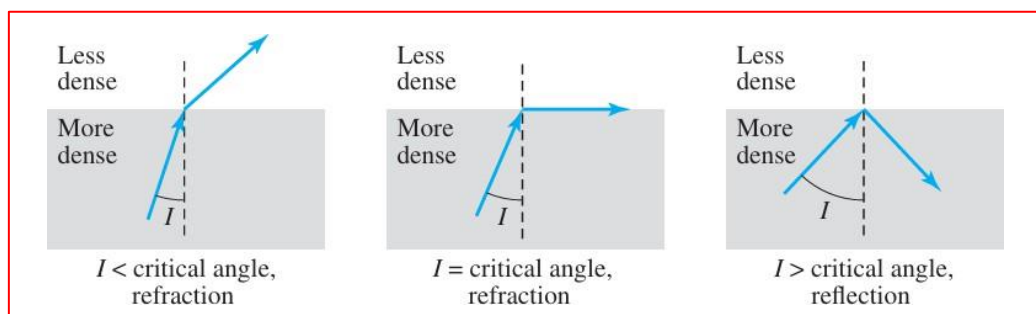


Figure 1.33: Bending of light ray

- **Light travels in a straight line** when moving through a uniform material.
- If light passes from one substance into another with a different density, the direction changes (refraction).
- **Critical Angle:** The angle of incidence at which light bends along the boundary between two materials. If the angle is greater than the critical angle, the light reflects instead of refracting.

Structure of Optical Fibers

Optical fibers guide light through **internal reflection**. The core (made of glass or plastic) is surrounded by a **cladding** with lower density. This ensures that light reflecting off the core-cladding boundary remains in the core.

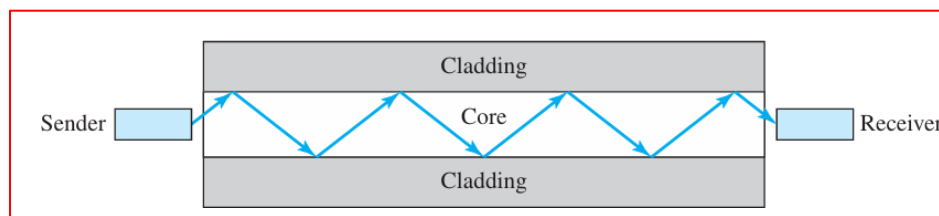


Figure 1.34: Optical fiber

Propagation Modes

Optical fibers use two main modes for light propagation:

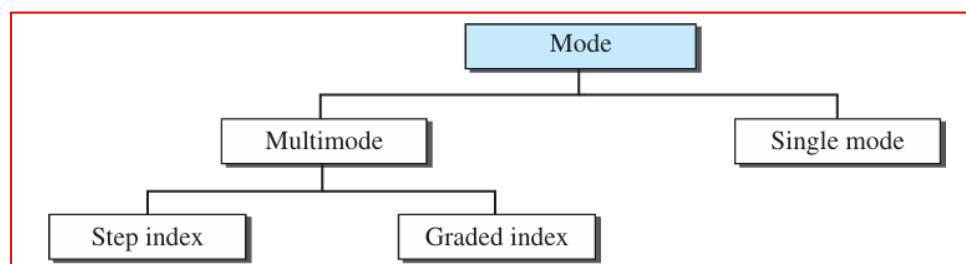


Figure 1.35: Propagation modes

1. **Multimode:** Multiple light beams travel through the core along different paths.
 - **Step-Index Fiber:** The core has uniform density, and light changes direction abruptly at the core-cladding interface.
 - **Graded-Index Fiber:** The core's density gradually decreases from the center, reducing signal distortion.
2. **Single-Mode:** Uses step-index fiber but has a smaller core and tightly focused light. The beams travel nearly parallel, minimizing delay and distortion.

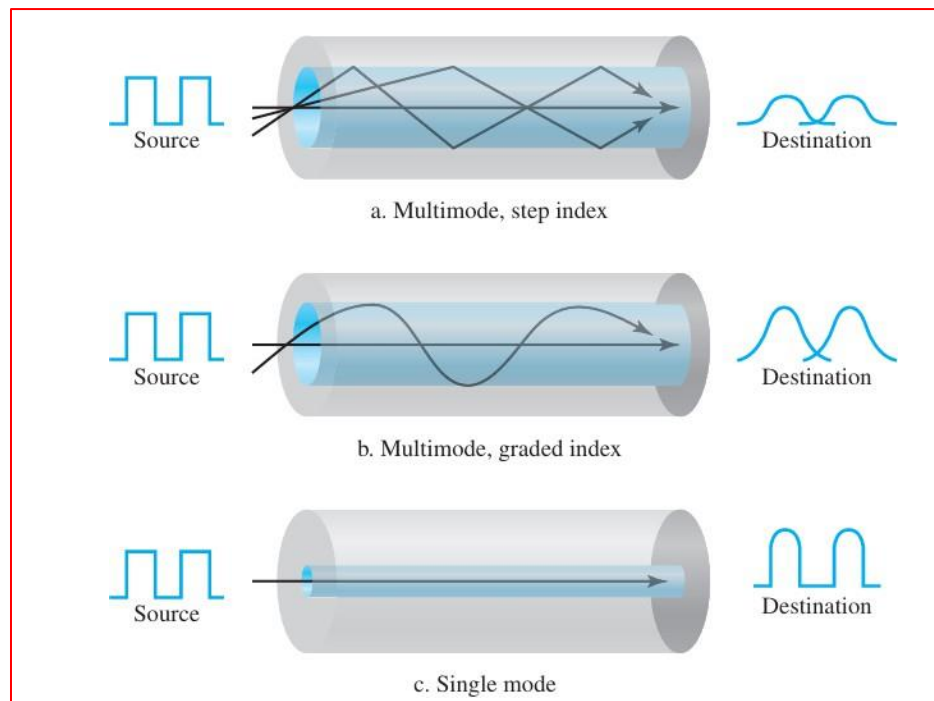


Figure 1.36: Modes

Fiber Sizes

Optical fibers are classified by the ratio of core diameter to cladding diameter (both measured in micrometers). Single-mode fibers typically have a smaller core diameter compared to multimode fibers.

Fiber-Optic Cable Composition

A typical fiber-optic cable consists of:

- **Outer jacket** (PVC or Teflon)
- **Kevlar strands** for strength
- **Plastic coating** for cushioning
- **Core and cladding** for light transmission

Fiber-Optic Connectors

There are three types of connectors:

1. **SC (Subscriber Channel)**: Push/pull locking, used in cable TV.
2. **ST (Straight Tip)**: Bayonet locking, used in networking.
3. **MT-RJ**: Same size as RJ45, used for data networks.

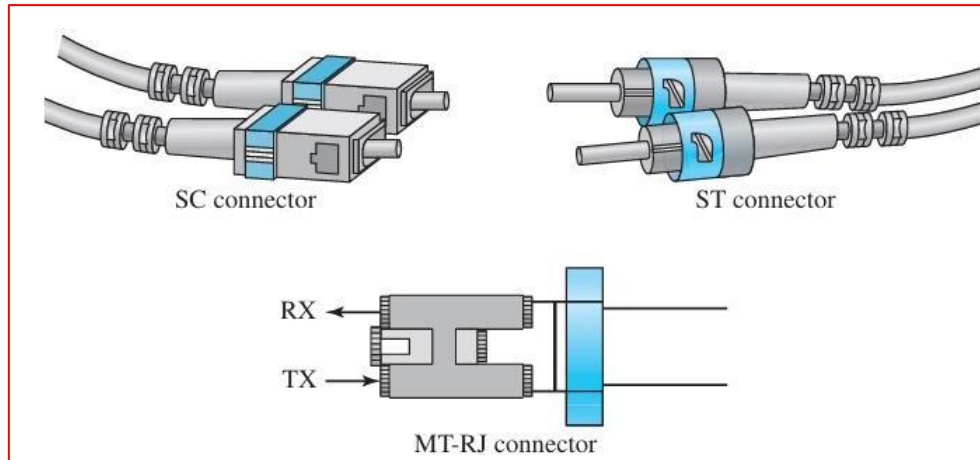


Figure 1.37: Fiber-optic cable connectors

Performance

Fiber-optic cables experience less signal attenuation than twisted-pair or coaxial cables, requiring fewer repeaters. They also support higher data transfer rates, especially with **wavelength-division multiplexing (WDM)**.

Applications

- **Backbone Networks:** Due to their wide bandwidth, fiber-optic cables are used in backbone networks, such as SONET.
- **Hybrid Networks:** Some cable TV companies use a mix of fiber-optic and coaxial cables.
- **Local Area Networks (LANs):** Fiber-optic cables are used in networks like 100Base-FX and 1000Base-X.

Advantages of Optical Fiber

1. **Higher Bandwidth:** Supports higher data rates.
2. **Less Signal Attenuation:** Can transmit over longer distances without needing repeaters.
3. **Immunity to Electromagnetic Interference:** Not affected by electromagnetic noise.
4. **Resistance to Corrosive Materials:** More durable in harsh environments.
5. **Lightweight:** Lighter than copper cables.
6. **Greater Immunity to Tapping:** More secure than copper cables.

Disadvantages of Optical Fiber

1. **Installation and Maintenance:** Requires specialized skills.
2. **Unidirectional Light Propagation:** Requires two fibers for bidirectional communication.
3. **Cost:** More expensive than other cables, especially if bandwidth demand is low.

Fiber-optic cables offer numerous benefits, particularly in high-bandwidth and long-distance applications, but their costs and technical requirements must be carefully considered.

7. UNGUIDED MEDIA: WIRELESS

Unguided media refers to communication channels that use wireless signals to transmit data. These signals travel through the air without any physical conductor. Common types of wireless transmission include radio waves, microwaves, and infrared.

Wireless communication involves transmitting electromagnetic waves without using physical conductors like cables or wires. Instead, signals are broadcast through free space, making them accessible to any device equipped to receive them.

Electromagnetic Spectrum for Wireless Communication

Wireless communication utilizes a portion of the electromagnetic spectrum, ranging from 3 kHz to 900 THz, as shown in Figure 1.38. This spectrum includes a variety of frequencies that allow different methods of signal transmission.

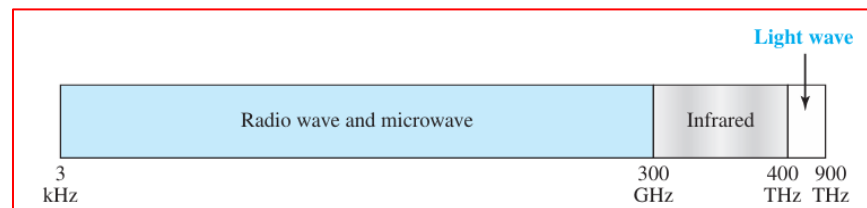


Figure 1.38 Electromagnetic spectrum for wireless communication

Types of Propagation

Unguided signals, such as radio waves, can travel from the source to the destination in three main ways, illustrated in Figure 1.40:

1. Ground Propagation:

- In this method, low-frequency radio waves travel close to the Earth's surface, following the curvature of the planet.
- These signals radiate from the transmitting antenna in all directions, and the distance they cover depends on the power of the signal—the higher the power, the farther the signal can travel.

2. Sky Propagation:

- Higher-frequency radio waves are transmitted upward into the ionosphere, where they are reflected back to Earth.
- This method enables long-distance communication with relatively low power.

3. Line-of-Sight Propagation:

- In this method, very high-frequency signals are transmitted directly between antennas in a straight line.
- The antennas must be properly aligned and either tall enough or close enough to avoid being affected by the curvature of the Earth. This method is more complex because radio waves can't be perfectly focused.

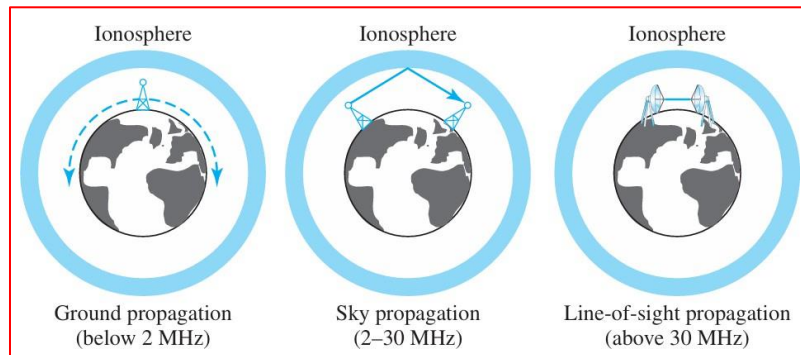


Figure 1.40: Propagation methods

Frequency Bands

The electromagnetic spectrum for wireless communication is divided into eight different ranges or "bands." These bands are classified based on frequency and are regulated by government authorities. The ranges extend from Very Low Frequency (VLF) to Extremely High Frequency (EHF). Each band has specific propagation characteristics and applications, as summarized in Table 1.3.

Table 1.3: Frequency Bands for Applications

Band	Range	Propagation	Application
Very Low Frequency (VLF)	3–30 kHz	Ground	Long-range radio navigation
Low Frequency (LF)	30–300 kHz	Ground	Radio beacons, navigational locators
Middle Frequency (MF)	300 kHz–3 MHz	Sky	AM radio
High Frequency (HF)	3–30 MHz	Sky	Citizens band (CB), ship/aircraft communication
Very High Frequency (VHF)	30–300 MHz	Sky and Line-of-sight	VHF TV, FM radio
Ultra-High Frequency (UHF)	300 MHz–3 GHz	Line-of-sight	UHF TV, cellular phones, paging, satellite
Super High Frequency (SHF)	3–30 GHz	Line-of-sight	Satellite communication
Extremely High Frequency (EHF)	30–300 GHz	Line-of-sight	Radar, satellite communication
Light Wave (Infrared)	300 GHz–900 THz	Line-of-sight	Infrared communication

This breakdown of the spectrum ensures efficient and organized use of frequencies for various communication purposes.

7.1. Radio Waves

Frequency Range: Typically, from 3 kHz to 1 GHz.

Characteristics of Radio Waves:

- **Omnidirectional Propagation:** Radio waves are mostly omnidirectional, meaning they spread out in all directions from the transmitting antenna. The sending and receiving antennas don't need to be aligned for successful communication, as any receiving antenna in range can pick up the signal. However, this characteristic also leads to a disadvantage: **interference**. Multiple antennas transmitting on the same frequency or band can interfere with one another.
- **Long-Distance Travel:** Radio waves, especially those that propagate in the **sky mode**, can travel long distances, making them ideal for applications like **AM radio broadcasting**.
- **Penetration of Walls:** Radio waves, particularly those with **low and medium frequencies**, can penetrate walls. This is useful because devices like AM radios can receive signals indoors. However, it can also be a disadvantage, as signals cannot be restricted to only the inside or outside of a building, leading to potential **signal leakage**.

Limitations:

- The radio wave band is **relatively narrow**, just under 1 GHz. When divided into subbands, the limited width of these subbands results in **low data rates** for digital communications.
- Most of the radio wave spectrum is **regulated** by government authorities, like the **FCC** in the United States and Department of Telecommunications (DoT) & Telecom Regulatory Authority of India (TRAI) in India. Any use of this spectrum requires **official permission**.
- Licensed Frequency Bands: FM band is licensed between **88-108 MHz** for radio broadcasting and Unlicensed Frequency Bands: Wi-Fi: 2.4 GHz, **5 GHz** band.

Omnidirectional Antenna:

Radio waves typically utilize **omnidirectional antennas**, which transmit signals in all directions. These antennas come in various types, depending on factors like **wavelength**, **signal strength**, and the intended purpose of transmission.

Applications of Radio Waves:

Thanks to their omnidirectional nature, radio waves are widely used for **multicasting**, where one sender transmits to many receivers. Common examples include:

- **AM and FM radio**
- **Television broadcasting**

- **Maritime radio**
- **Cordless phones**
- **Paging systems**

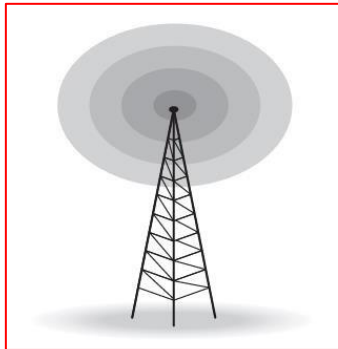


Figure 1.41: Omnidirectional antenna

7.2. Microwaves

Frequency Range: From 1 GHz to 300 GHz.

Characteristics: Microwaves require line-of-sight transmission, meaning the transmitter and receiver must be directly visible to each other. They are less effective in penetrating obstacles like buildings.

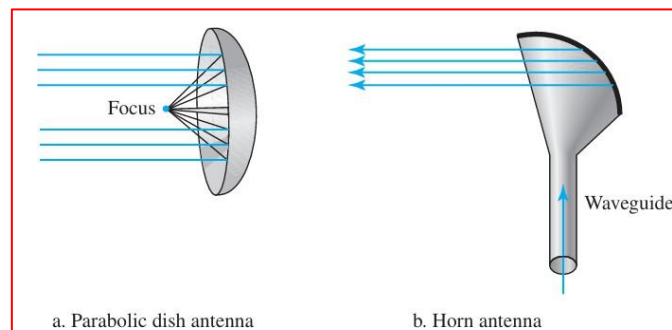


Figure 1.42 Unidirectional antennas

Applications: Satellite communications, radar systems, and microwave ovens. In networking, microwaves are used for point-to-point communication links and cellular networks.

7.3. Infrared

Frequency Range: From 300 GHz to 400 THz.

Characteristics: Infrared signals are used for short-range communication and do not penetrate walls, making them suitable for indoor use. They are highly directional and require line-of-sight transmission.

Applications: Remote controls, short-range data transmission (such as between computers and peripherals), and infrared sensors for detecting heat in security systems or medical devices.

8. PACKET SWITCHING

In data communication, when a message needs to be sent from one end system to another through a **packet-switched network**, it must be divided into smaller units called **packets**. These packets can be of either fixed or variable sizes, depending on the network and the protocol being used.

Key Features of Packet Switching:

1. No Resource Allocation:

- Unlike circuit-switched networks, packet switching does not reserve any specific resources like bandwidth or processing time for the packets.
- Resources are allocated only when needed, and packets are processed on a **first-come, first-served** basis.

2. Possible Delays:

- Since there is no dedicated path or reserved resources, packets might experience delays. For instance, if a switch is busy processing other packets, newly arrived packets must wait their turn, which can increase transmission time.

Types of Packet-Switched Networks:

1. **Datagram Networks:** In these networks, each packet is treated independently, and it may take different routes to reach the destination.
2. **Virtual Circuit Networks:** These networks establish a pre-determined path before any data packets are sent, ensuring all packets follow the same route.

Packet switching is an efficient way to transfer data, especially in systems where multiple users need to share the same network resources.

8.1. Datagram Networks

In a **datagram network**, each packet is handled independently, even if it's part of a larger transmission. The network treats each packet as if it stands alone. These individual packets are known as **datagrams**.

Key Features of Datagram Networks:

1. **Packet Independence:** Each packet in a datagram network can take a different path to its destination, and the network doesn't maintain any connection state between sender and receiver.
2. **Routing:** Packet routing is typically done at the **network layer**, where packets are forwarded based on their destination address. The devices that manage packet routing are called **routers**.

3. **No Fixed Path:** Since packets may travel along different routes, they might reach their destination out of order or with varying delays. Some packets could even be dropped if the network runs out of resources.
4. **Connectionless:** A datagram network is often referred to as a **connectionless network** because it doesn't require a setup phase (like circuit-switched networks). No information about the connection is saved, and each packet is routed independently.

How Datagram Networks Work

In Figure 1.43, four packets are sent from station A to station X using the datagram approach. Here, switches are called routers, and they are depicted with a different symbol. Although these four packets belong to the same message, they might take different paths to reach their destination due to varying link capacities and network congestion. This can lead to packets arriving out of order, with differing delays, or even being lost or dropped due to insufficient resources. Upper-layer protocols typically handle the reordering and retransmission of lost packets before delivering them to the application.

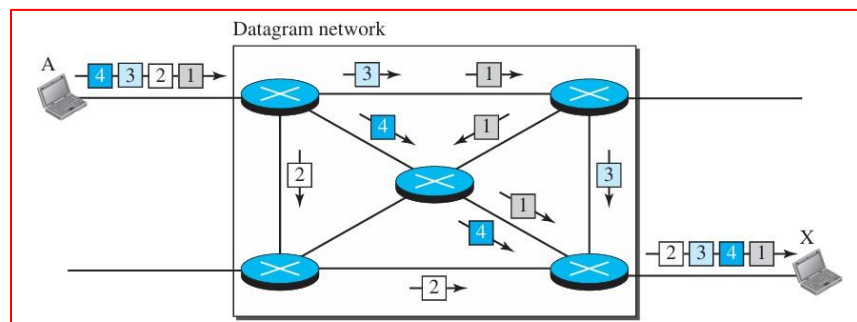


Figure 1.43: A datagram network with four switches (routers)

Routing Table

In a datagram network, each switch uses a routing table based on destination addresses to forward packets. These tables are dynamic and updated regularly.

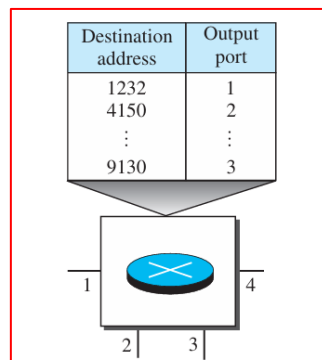


Figure 1.44: Routing table in a datagram network

The routing table records destination addresses and the corresponding output ports. This differs from circuit-switched networks, where entries are created during the setup phase and removed during teardown.

Destination Address

Every packet in a datagram network has a header containing a destination address. Upon receiving a packet, the switch checks this address and uses the routing table to determine the appropriate forwarding port. This destination address remains unchanged throughout the packet's journey.

Delay

Despite their efficiency, datagram networks can experience higher delays compared to virtual-circuit networks. Although there are no setup or teardown phases, each packet may encounter waiting times at switches. Additionally, since packets from the same message may travel through different routes, delays are not uniform. Figure 1.45, illustrates the delay for a packet traveling through two switches, including transmission times ($3T$), propagation delays (3τ), and waiting times ($w1 + w2$). The total delay is given by:

$$Total\ delay = 3T + 3\tau + w1 + w2$$

- **Transmission Time:** The time to send a packet from one point to another.
- **Propagation Delay:** The time it takes for the signal to travel through the medium.
- **Waiting Time:** Time spent at routers before being forwarded.

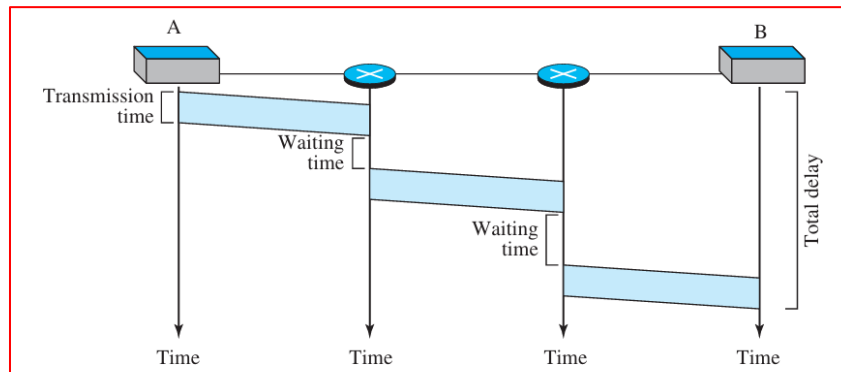


Figure 1.45: Delay in a datagram network

Advantages:

- **Efficiency:** Datagram networks can be more efficient than circuit-switched networks. Resources like bandwidth are allocated only when packets are being transmitted, allowing for better utilization of network resources.

8.2. Virtual-Circuit Networks

A **virtual-circuit network (VCN)** is a hybrid network model that combines features of both **circuit-switched** and **datagram networks**. It provides a balance between connection-oriented and connectionless transmission methods.

Key Features:

1. Connection Phases:

- A VCN has three distinct phases: **setup**, **data transfer**, and **teardown**.
 - **Setup Phase:** A path is established between the sender and receiver before data transmission.
 - **Data Transfer Phase:** Data is sent along the established path in packets.
 - **Teardown Phase:** After data transmission, the connection is terminated, and resources are released.

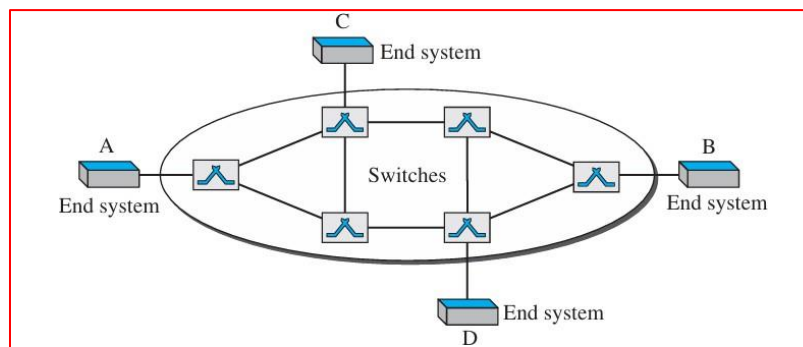


Figure 1.46: Virtual-circuit network

2. Resource Allocation:

- Resources can either be allocated during the setup phase (as in a circuit-switched network) or dynamically during data transmission (similar to a datagram network).

3. Packetized Data with Local Addressing:

- Data is divided into packets, each of which carries an address. However, unlike in a datagram network, the address is not end-to-end but **local**, meaning it only tells the next switch where to send the packet.

4. Consistent Path for Packets:

- Once a connection is established, all packets follow the same predetermined path. This ensures a predictable route for all the packets between the sender and receiver, akin to a circuit-switched network.

5. Layer of Operation:

- Virtual-circuit networks typically operate at the **data-link layer**, while circuit-switched networks operate at the physical layer, and datagram networks at the network layer.

Addressing in Virtual-Circuit Networks:

1. Global Addressing:

- A global address is used to uniquely identify the source and destination during the setup phase. This address is typically unique within the network or globally if the network is part of a larger system.

2. Virtual-Circuit Identifier (VCI):

- During the data transfer phase, a **virtual-circuit identifier (VCI)** is used instead of the global address. The VCI is a small number with local scope, meaning it only identifies the path between two adjacent switches.
- The VCI changes at each switch, as each switch uses its own set of VCIs to manage the connection. This allows efficient and simple management of packet forwarding.

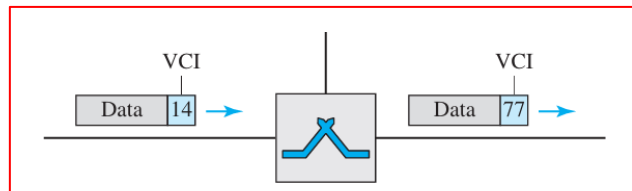


Figure 1.47: Virtual-circuit identifier

Three Phases in a Virtual-Circuit Network

In a virtual-circuit network, the communication between a source and destination involves three phases: **setup**, **data transfer**, and **teardown**. These phases ensure that a reliable path is established and maintained for the communication session.

1. Setup Phase:

- The source and destination use their global addresses to establish a connection. During this phase, switches along the path create table entries to store information about the virtual circuit. This phase ensures that each switch is prepared to route the data properly.

2. Data Transfer Phase:

- After the setup phase, data is transferred between the source and destination. The switches use the table entries created during the setup phase to route the frames.

The switches maintain information like the incoming and outgoing ports and Virtual Circuit Identifiers (VCI). Each frame is processed the same way, with the VCIs changing at each switch to ensure the data follows the correct path. This phase continues until all frames are transferred.

3. Teardown Phase:

- Once the data transfer is complete, the source and destination send signals to the switches to remove the corresponding table entries, effectively ending the virtual circuit.

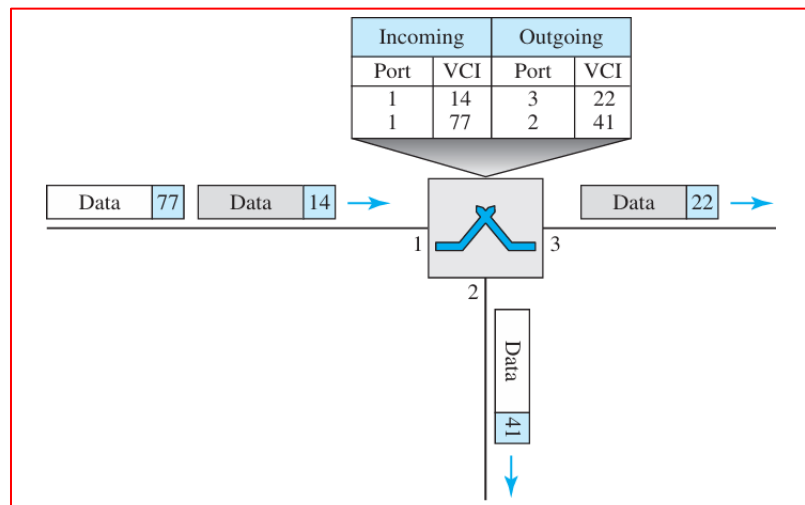


Figure 1.48: Switch and tables in a virtual-circuit network

Data-Transfer Phase

During data transfer, the key action is switching the frames between the source and destination. Each switch in the path must have a table with entries corresponding to the virtual circuit. A table typically consists of four columns: incoming port, incoming VCI, outgoing port, and outgoing VCI.

- When a frame arrives at a switch, the switch looks for the entry that matches the incoming port and VCI.
- After identifying the entry, the switch updates the VCI to the new value and forwards the frame to the next switch via the outgoing port.

For example, if a frame arrives at switch 1 with VCI 14 on port 1, the switch finds this entry in its table, updates the VCI to 22, and forwards the frame through port 3. This process repeats at each switch, ensuring the frame reaches its destination.

Setup Phase

In the setup phase, a virtual circuit is established between the source and destination by creating table entries at each switch. This phase consists of two main steps:

1. Setup Request:

- A setup request frame is sent from the source (A) to the destination (B). As the frame passes through each switch, the switch creates an entry in its table.
- For example, when the setup frame reaches switch 1, it identifies that the outgoing port for the connection is port 3 and assigns an incoming VCI (14) for the frame coming from port 1. At this point, the outgoing VCI remains unknown.

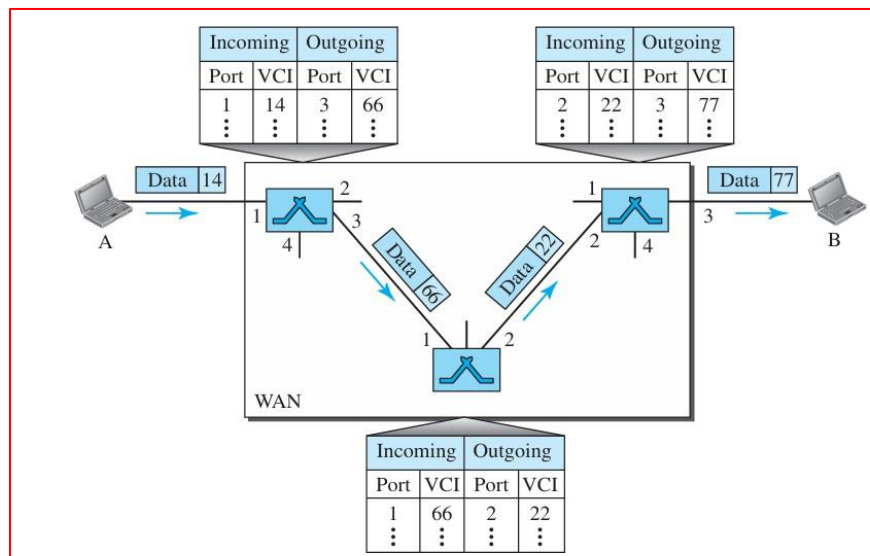


Figure 1.49: Source-to-destination data transfer in a virtual-circuit network

2. Acknowledgment:

- Once the setup request reaches the destination, the destination assigns a VCI (e.g., 77) for incoming frames from the source. The acknowledgment is sent back to the source, and each switch updates its table to complete the missing outgoing VCI information.

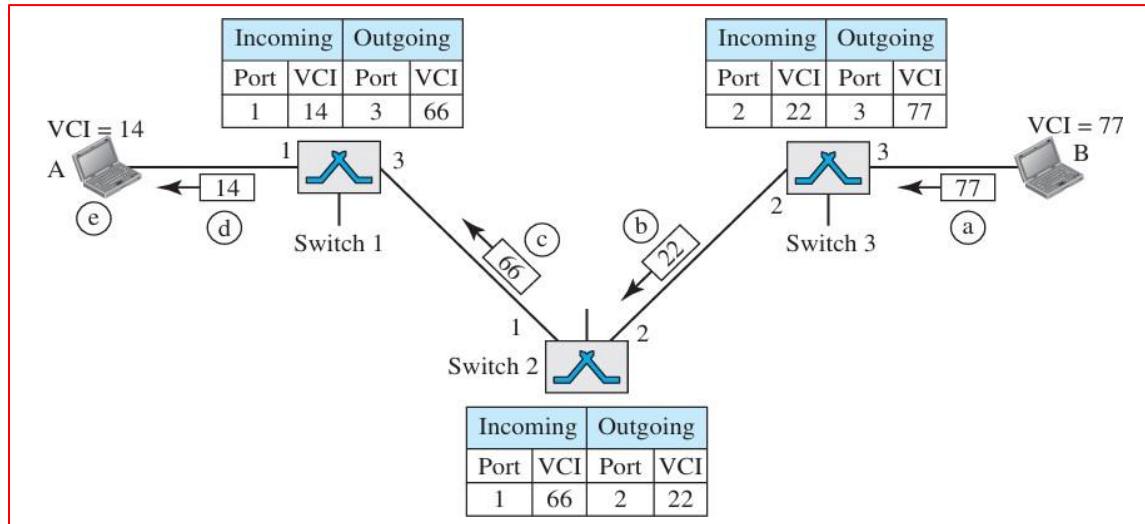


Figure 1.50: Setup acknowledgment in a virtual-circuit network

3. Teardown Phase

When the communication is finished, the source and destination send a signal to the switches to remove the corresponding table entries, thus ending the virtual circuit. This process frees up the resources for future virtual circuits.

Efficiency in Virtual-Circuit Networks

In a virtual-circuit network, resource allocation can happen either during the **setup phase** or on demand during the **data-transfer phase**. When resources are reserved during the setup phase, each packet experiences the same delay. However, if resources are allocated on demand, packet delays may vary.

Even when resource allocation is on demand, a significant advantage of virtual-circuit networks is that the source can check resource availability before data transfer.

In virtual-circuit switching, all packets from the same source to the same destination follow the same path. However, with on-demand resource allocation, packets may arrive with different delays depending on resource availability.

Delay in Virtual-Circuit Networks

In a virtual-circuit network, delays occur during the **setup** and **teardown** phases. These are one-time delays. If resources are allocated during setup, there is no additional waiting time for individual packets during data transfer.

The total delay in such a network includes:

- **Three transmission times (3T):** the time taken for the packet to be transmitted across links.

- **Three propagation times (3τ):** the time taken for the packet to travel across physical distances.
- **Setup delay:** includes transmission and propagation in both directions during the setup phase.
- **Teardown delay:** includes transmission and propagation in one direction during the teardown phase.

For simplicity, processing delays at the switches (routers) are ignored in this calculation.

Thus, the total delay for the packet is:

$$Total\ delay = 3T + 3\tau + setup\ delay + teardown\ delay$$

QUESTIONS

1. Data Communications

1. What are the five key components of a data communication system, and what role does each play in ensuring effective communication?
2. Explain the different forms of data representation used in data communications and provide examples for each.
3. Describe the three modes of data flow and provide real-world examples where each mode is used.

2. Networks

1. What are the key criteria used to evaluate the performance, reliability, and security of a network?
2. Explain the difference between point-to-point and multipoint physical structures in network connections. Provide advantages and disadvantages of each.

3. Network Types

1. Compare and contrast a Local Area Network (LAN) and a Wide Area Network (WAN) in terms of characteristics, speed, and geographical coverage.
2. Define packet switching and explain the key differences between circuit switching and packet switching.
3. Discuss the role of routers and switches in the structure of the Internet. How do they contribute to efficient data transfer?
4. What are the different ways to access the Internet, and how do factors like speed and coverage differ between these methods?

4. Protocol Layering

1. Explain the key principles of protocol layering and discuss how these principles ensure efficient communication between two devices over a network.
2. What are logical connections in the context of protocol layering, and how do they facilitate communication between peer layers on different devices?
3. Explain the layered architecture of the TCP/IP protocol suite. How does each layer contribute to overall network communication?
4. Discuss the main functions of the following layers in the TCP/IP protocol suite:
 - a. Application Layer
 - b. Transport Layer
 - c. Network Layer
 - d. Data Link Layer
 - e. Physical Layer
5. Describe the process of encapsulation and decapsulation in the TCP/IP protocol suite. Why are these processes crucial for data communication?
6. Explain the different types of addresses used at each layer of the TCP/IP protocol stack. Why is addressing critical for network communication?
7. What is multiplexing and demultiplexing in the TCP/IP protocol suite? How do these processes ensure that data is correctly sent and received by the right application?

5. Transmission Media

1. Define guided media and list and explain three types of guided media used in network communications.
2. Explain the difference between unshielded twisted-pair (UTP) and shielded twisted-pair (STP) cables. What are their primary uses?
3. Describe how the twisting of pairs in twisted-pair cables helps to reduce crosstalk.
4. What are the main components of a coaxial cable, and how do they contribute to its performance?
5. Describe the principle of operation of fiber-optic cables and explain why they provide high-speed data transmission.
6. Discuss the advantages and disadvantages of fiber-optic cables compared to coaxial and twisted-pair cables.
7. What are radio waves, and how are they used in wireless communication? Provide an example of a common application.
8. Explain the concept of frequency allocation in radio wave communication and its importance.
9. Discuss the characteristics of microwave transmission and its typical uses in communication systems.
10. Describe how infrared communication works and list two common applications where infrared technology is used.

6. Packet Switching

1. What is a datagram network? Describe its key characteristics and how it handles packet routing and delivery.
2. Compare and contrast datagram networks with virtual-circuit networks in terms of reliability, complexity, and overhead.
3. In a datagram network, how does the destination node determine the correct order of packets? What issues might arise from the unordered delivery of packets?
4. Explain how a datagram network handles packet fragmentation and reassembly. What are the potential impacts on network performance and reliability?
5. Define a virtual-circuit network and describe the main phases involved in setting up, maintaining, and tearing down a virtual circuit.
6. Discuss the advantages and disadvantages of virtual-circuit networks compared to datagram networks. Include aspects such as connection setup, resource allocation, and data transfer efficiency.
7. How does a virtual-circuit network handle data transfer once the connection is established? What mechanisms are in place to ensure data integrity and order?
8. Explain the role of signaling in virtual-circuit networks. How does signaling contribute to the establishment and teardown of virtual circuits?
9. Describe a scenario where virtual-circuit networks would be preferred over datagram networks. Justify your choice based on factors such as performance requirements and network management.

Module 2

Error Detection and Correction

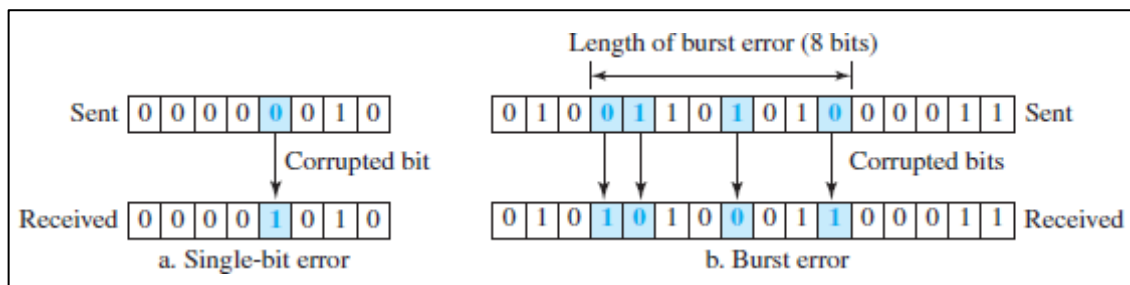
The data link layer transforms the physical layer data into a form suitable for node-to-node (hop-to-hop) communication. Specific responsibilities of the data link layer include **framing, addressing, flow control, error control, and media access control**.

INTRODUCTION

Whenever bits flow from one point to another, they are subject to unpredictable changes because of interference. This interference can change the shape of the signal.

1. Single-Bit Error

The term *single-bit error* means that only 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.



2. Burst Error

The term *burst error* means that 2 or more bits in the data unit have changed from 1 to 0 or from 0 to 1.

Redundancy

- ➔ The central concept in detecting or correcting errors is redundancy.
- ➔ Some extra bits (redundant bits) are added with data to detect and correct errors.
- ➔ These redundant bits are added by the sender and removed by the receiver.

Detection versus Correction

- ➔ The correction of errors is more difficult than the detection.

- ➔ In error detection, we are looking only to see if any error has occurred.
- ➔ In error correction, we need to know the exact number of bits that are corrupted and more importantly, their location in the message.

Coding

Redundancy is achieved through various coding schemes. Coding schemes can be divided into two broad categories: block coding and convolution coding.

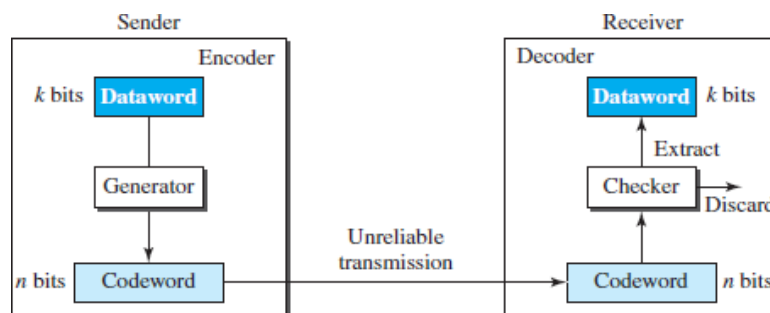
BLOCK CODING

- ➔ In block coding, message is divided into blocks, each of k bits, called **datawords**.
- ➔ Add r redundant bits to each block to make the length $n = k + r$. The resulting n -bit blocks are called **codewords**.
- ➔ With k bits, 2^k datawords can be created; with n bits, 2^n codewords can be created.
- ➔ Since $n > k$, the number of possible codewords is larger than the number of possible datawords.

Error Detection

If the following two conditions are met, the receiver can detect a change in the original codeword.

1. The receiver has (or can find) a list of valid codewords.
2. The original codeword has changed to an invalid one.



Hamming Distance

- ➔ The Hamming distance between two words (of the same size) is the number of differences between the corresponding bits.

- ➔ The Hamming distance between two words x and y is shown as $d(x, y)$.
- ➔ The Hamming distance can easily be found by applying the XOR operation on the two words and count the number of 1s in the result.

Ex The Hamming distance $d(10101, 11110)$ is 3 because

Minimum Hamming Distance

- ➔ It is the measurement that is used for designing a code is the minimum Hamming distance of two.
- ➔ It is the smallest Hamming distance ($d_{\min}$) between all possible pairs.

$d(00000, 01011) = 3$	$d(00000, 10101) = 3$	$d(00000, 11110) = 4$
$d(01011, 10101) = 4$	$d(01011, 11110) = 3$	$d(10101, 11110) = 3$

$d_{\min}$ in this case is 3

Hamming Distance and Error

When a codeword is corrupted during transmission, the Hamming distance between the sent and received codewords is the number of bits affected by the error.

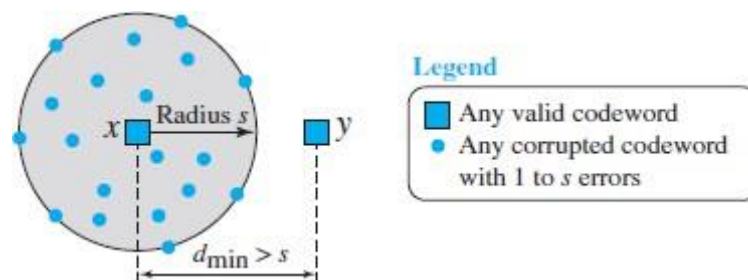
For example, if the codeword 00000 is sent and 01101 is received, 3 bits are in error and the Hamming distance between the two is $d(00000, 01101) = 3$.

Minimum Distance for Error Detection

If s errors occur during transmission, the Hamming distance between the sent codeword and received codeword is s . If the code is to detect up to s errors, the minimum distance between the valid codes must be $s + 1$, so that the received codeword does not match a valid codeword.

We can look at this geometrically.

- ➔ Let us assume that the sent codeword x is at the center of a circle with radius s . All other received codewords that are created by 1 to s are points inside the circle or on the perimeter of the circle
- ➔ All other valid codewords must be outside the circle, as shown in Figure



LINEAR BLOCK CODES

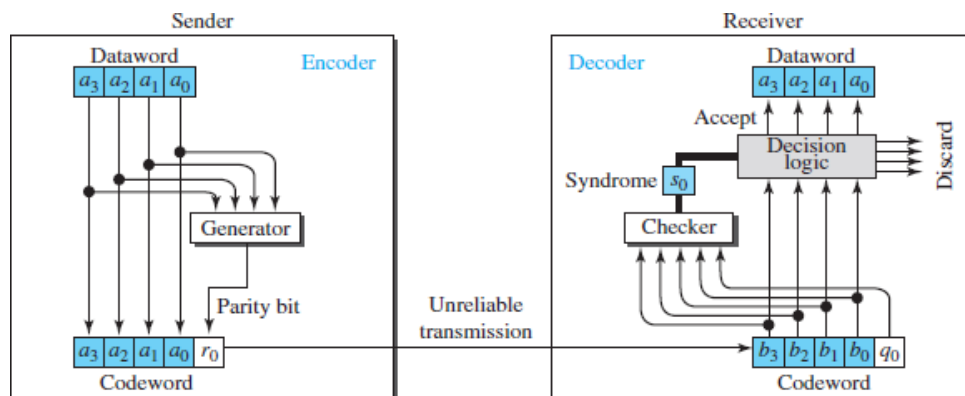
Minimum Distance for Linear Block Codes

It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is the number of 1s in the nonzero valid codeword with the smallest number of 1s

Some Linear Block Codes

1. Simple Parity-Check Code(Explain the structure of encoder and decoder S P Code)

- ➔ In this code, a k -bit dataword is changed to an n -bit codeword where $n = k + 1$.
- ➔ The extra bit, called the parity bit, is selected to make the total number of 1s in the codeword even.
- ➔ The minimum Hamming distance for this category is $d_{\min}=2$, which means that the code is a single-bit error-detecting code; it cannot correct any error.



1. The encoder uses a generator that takes a copy of a 4-bit dataword (a_0, a_1, a_2 and a_3) and generates a parity bit r_0 . The dataword bits and the parity bit create the 5-bit codeword. The parity bit that is added makes the number of 1's in the codeword even.
2. The sender sends the codeword which may be corrupted during transmission. The receiver receives a 5-bit word.
3. The checker at the receiver does the same thing as the generator in the sender with one exception: The addition is done over all 5 bits. The result, which is called the syndrome, is just 1 bit. The

4. The syndrome is passed to the decision logic analyzer. If the syndrome is 0, there is no error in the received codeword; the data portion of the received codeword is accepted as the dataword;
5. If the syndrome is 1, the data portion of the received codeword is discarded. The dataword is not created.

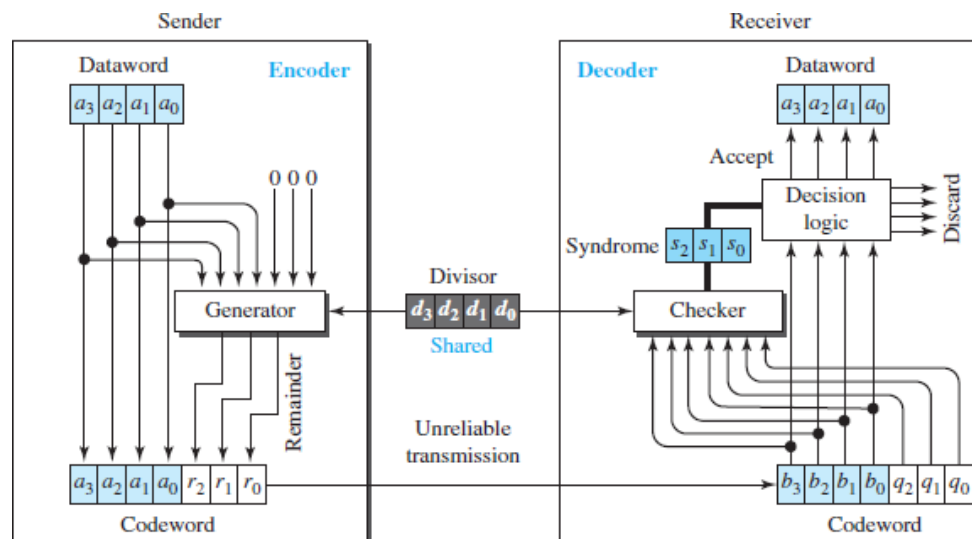
CYCLIC CODES

- ➔ Cyclic codes are special linear block codes with one extra property.
- ➔ In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.
- ➔ In this case, if we call the bits in the first word a_0 to a_6 and the bits in the second word b_0 to b_6 , we can shift the bits by using the following:

$$b_1 = a_0 \quad b_2 = a_1 \quad b_3 = a_2 \quad b_4 = a_3 \quad b_5 = a_4 \quad b_6 = a_5 \quad b_0 = a_6$$

Cyclic Redundancy Check

It is a subset of cyclic codes called the cyclic redundancy check (CRC), which is used in networks such as LANs and WANs.

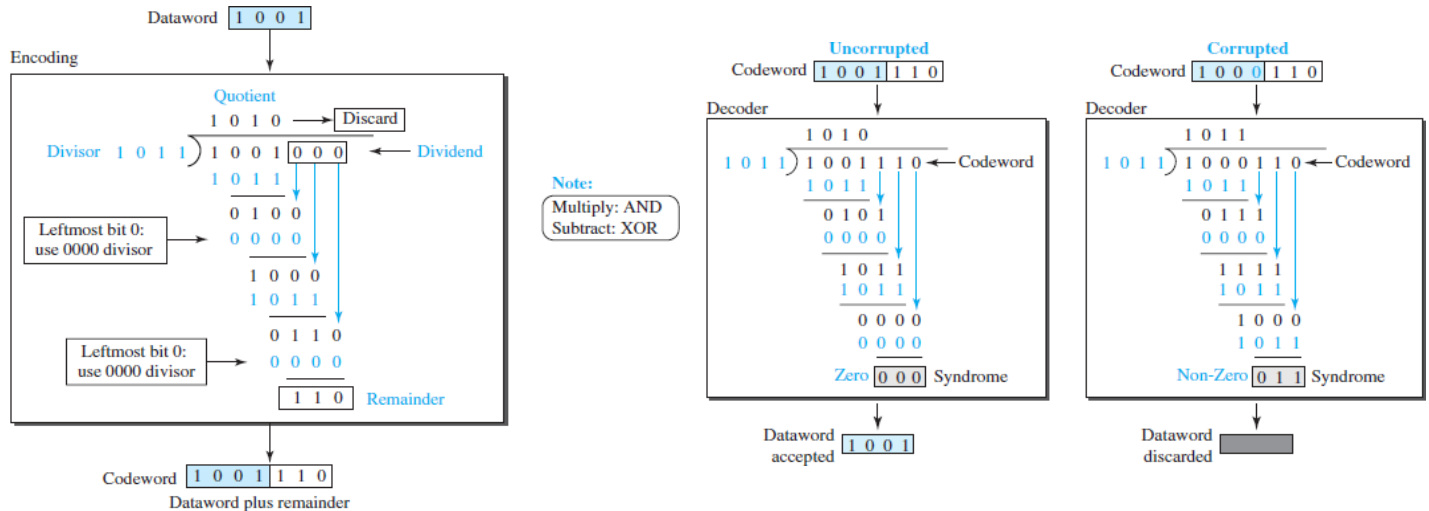


In the **encoder**,

1. The dataword has k bits (4 here); the codeword has n bits (7 here).
2. The size of the dataword is augmented by adding $n - k$ (3 here) 0s to the right-hand side of the word. The n -bit result is fed into the generator.
3. The generator uses a divisor of size $n - k + 1$ (4 here), predefined and agreed upon. The generator divides the augmented dataword by the divisor (modulo-2 division).
4. The quotient of the division is discarded; the remainder ($r_2r_1r_0$) is appended to the dataword to create the codeword.

The decoder,

1. Receives the possibly corrupted codeword.
2. A copy of all n bits is fed to the checker which is a replica of the generator.
3. The remainder produced by the checker is a syndrome of $n - k$ (3 here) bits, which is fed to the decision logic analyzer.
4. The analyzer has a simple function. If the syndrome bits are all as, the 4 leftmost bits of the codeword are accepted as the dataword (interpreted as no error); otherwise, the 4 bits are discarded (error).



Polynomials

A pattern of 0s and 1s can be represented as a **polynomial** with coefficients of 0 and 1. The power of each term shows the position of the bit; the coefficient shows the value of the bit.

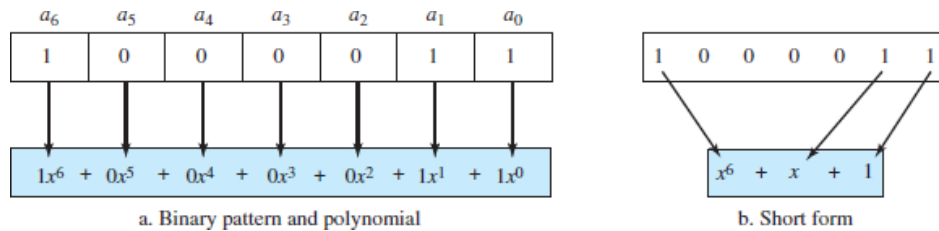


Figure shows one immediate benefit; a 7-bit pattern can be replaced by three terms. The benefit is even more conspicuous when we have a polynomial such as $x^{23} + x^3 + 1$. Here the bit pattern is 24 bits in length (three 1s and twenty-one 0s) while the polynomial is just three terms.

Degree of a Polynomial

The degree of a polynomial is the highest power in the polynomial. For example, the degree of the polynomial $x^6 + x + 1$ is 6. Note that the degree of a polynomial is 1 less than the number of bits in the pattern. The bit pattern in this case has 7 bits.

Adding and Subtracting Polynomials

Adding and subtracting polynomials in mathematics are done by adding or subtracting the coefficients of terms with the same power. For example, adding $x^5 + x^4 + x^2$ and $x^6 + x^4 + x^2$ gives just $x^6 + x^5$. The terms x^4 and x^2 are deleted.

Multiplying or Dividing Terms

In this arithmetic, multiplying a term by another term is very simple; we just add the powers. For example, $x^3 \times x^4$ is x^7 ,

For dividing, we just subtract the power of the second term from the power of the first. For example, $x^5 \div x^2$ is x^3 .

Multiplying Two Polynomials

Multiplying a polynomial by another is done term by term. Each term of the first polynomial must be multiplied by all terms of the second. The result, of course, is then simplified, and pairs of equal terms are deleted. The following is an example:

$$\begin{aligned}
 &(x^5 + x^3 + x^2 + x)(x^2 + x + 1) \\
 &= x^7 + x^6 + x^5 + x^5 + x^4 + x^3 + x^4 + x^3 + x^2 + x^3 + x^2 + x \\
 &= x^7 + x^6 + x^3 + x
 \end{aligned}$$

Dividing One Polynomial by Another

- ➔ We divide the first term of the dividend by the first term of the divisor to get the first term of the quotient.
- ➔ We multiply the term in the quotient by the divisor and subtract the result from the dividend.
- ➔ We repeat the process until the dividend degree is less than the divisor degree.

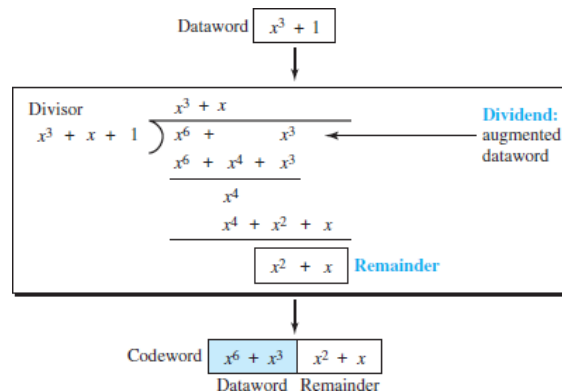
Shifting

Shifting to the left is accomplished by multiplying each term of the polynomial by x^m , where m is the number of shifted bits; shifting to the right is accomplished by dividing each term of the polynomial by x^m .

$$\begin{array}{ll}
 \text{Shifting left 3 bits:} & 10011 \text{ becomes } 10011000 \\
 \text{Shifting right 3 bits:} & 10011 \text{ becomes } 10
 \end{array}
 \quad
 \begin{array}{ll}
 x^4 + x + 1 \text{ becomes } & x^7 + x^4 + x^3 \\
 x^4 + x + 1 \text{ becomes } & x
 \end{array}$$

Cyclic Code Encoder Using Polynomials

- ➔ The dataword 1001 is represented as $x^3 + 1$.
- ➔ The divisor 1011 is represented as $x^3 + x + 1$.
- ➔ Find the augmented dataword, we have left-shifted the dataword 3 bits (multiplying by x)
The result is $x^6 + x^3$.



Cyclic Code Analysis

We can analyze a cyclic code to find its capabilities by using polynomials. We define the following, where $f(x)$ is a polynomial with binary coefficients.

Dataword: $d(x)$	Codeword: $c(x)$	Generator: $g(x)$
Syndrome: $s(x)$	Error: $e(x)$	

.In a cyclic code,

1. If $s(x) \neq 0$, one or more bits is corrupted.

2. If $s(x) = 0$, either

a. No bit is corrupted. or

b. Some bits are corrupted, but the decoder failed to detect them.

In our analysis we want to find the criteria that must be imposed on the generator, $g(x)$ to detect the type of error we especially want to be detected. Let us first find the relationship among the sent codeword, error, received codeword, and the generator.

We can say

$$\text{Received codeword} = c(x) + e(x)$$

In other words, the received codeword is the sum of the sent codeword and the error. The receiver divides the received codeword by $g(x)$ to get the syndrome. We can write this as

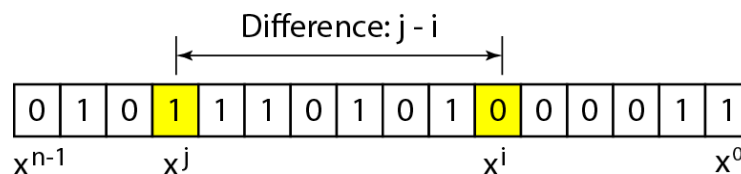
$$\frac{\text{Received codeword}}{g(x)} = \frac{c(x)}{g(x)} + \frac{e(x)}{g(x)}$$

In a cyclic code, those $e(x)$ errors that are divisible by $g(x)$ are not caught.

Single-Bit Error

If the generator has more than one term and the coefficient of x^0 is 1, all single errors can be caught.

Two Isolated Single-Bit Errors



If a generator cannot divide $x^{t+1} + 1$ (t between 0 and $n - 1$), then all isolated double errors can be detected.

Odd Numbers of Errors

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

Burst Errors

1. All burst errors with $L \leq r$ will be detected.
2. All burst errors with $L = r + 1$ will be detected with probability $1 - (1/2)^{r-1}$
3. All burst errors with $L > r + 1$ will be detected with probability $1 - (1/2)^r$

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide $x^{t+1} + 1$, for t between 2 and $n - 1$.
4. It should have the factor $x + 1$.

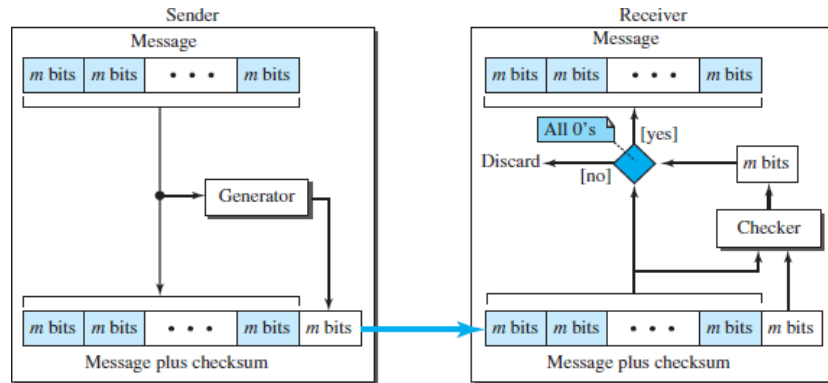
Advantages of Cyclic Codes

1. Cyclic codes have a very good performance in detecting single-bit errors, double errors, an odd number of errors, and burst errors.
2. They can easily be implemented in hardware and software. They are especially fast when implemented in hardware.

CHECKSUM

Checksum is an error-detecting technique that can be applied to a message of any length. In the Internet, the checksum technique is mostly used at the network and transport layer rather than the data-link layer.

At the source, the message is first divided into m -bit units. The generator then creates an extra m -bit unit called the **checksum**, which is sent with the message. At the destination, the checker creates a new checksum from the combination of the message and sent checksum. If the new checksum is all 0s, the message is accepted; otherwise, the message is discarded.



Concept

The concept of the checksum is not difficult.

- ➔ Suppose our data is a list of five 4-bit numbers that we want to send to a destination.
- ➔ In addition to sending these numbers, we send the sum of the numbers.
- ➔ For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers.
- ➔ The receiver adds the five numbers and compares the result with the sum.
- ➔ If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.
- ➔ We can make the job of the receiver easier if we send the negative (complement) of the sum, called the **checksum**.
- ➔ In this case, we send (7, 11, 12, 0, 6, -36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

One's Complement

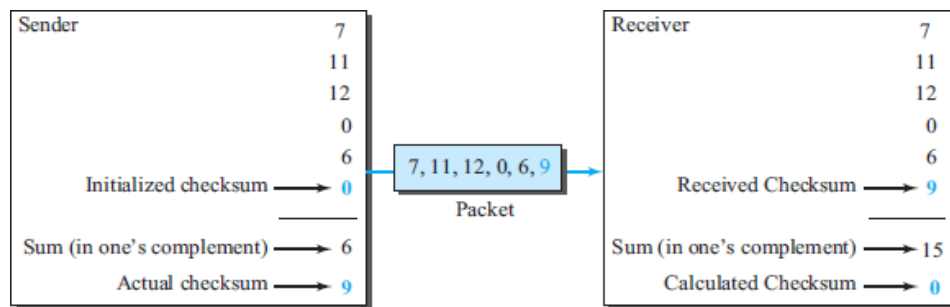
- ➔ The previous example has one major drawback. All of our data can be written as a 4-bit word (they are less than 15) except for the checksum.
- ➔ One solution is to use one's complement arithmetic.
- ➔ If the number has more than n bits, the extra leftmost bits need to be added to the n rightmost bits (wrapping).
- ➔ In one's complement arithmetic, a negative number can be represented by inverting all bits (changing a 0 to a 1 and a 1 to a 0).

Sender

1. The sender initializes the checksum to 0 and adds all data items and the checksum.
2. The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6.
3. The sum is then complemented, resulting in the checksum value 9 ($15 - 6 = 9$). The sender now sends six data items to the receiver including the checksum 9.

Receiver

1. The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45.
2. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted.
3. The receiver drops the checksum and keeps the other data items.
4. If the checksum is not zero, the entire packet is dropped



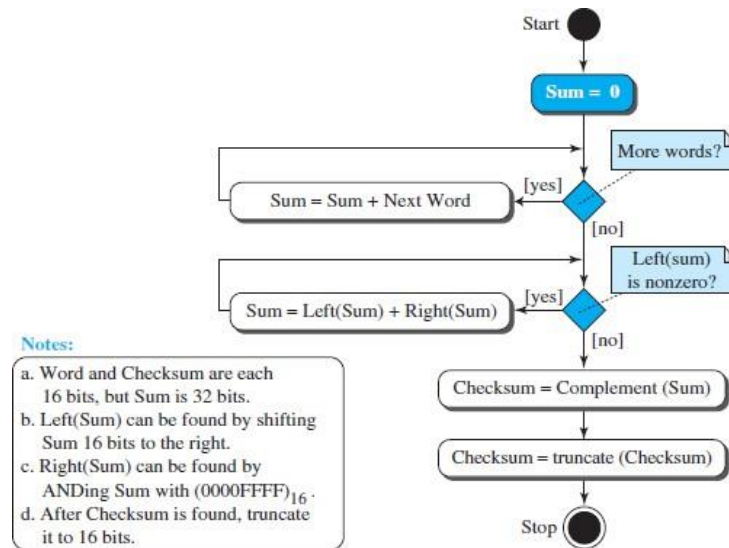
Internet Checksum

Traditionally, the Internet has been using a 16-bit checksum. The sender calculates the checksum by following these steps.

<i>Sender</i>	<i>Receiver</i>
1. The message is divided into 16-bit words.	1. The message and the checksum are received.
2. The value of the checksum word is initially set to zero.	2. The message is divided into 16-bit words.
3. All words including the checksum are added using one's complement addition.	3. All words are added using one's complement addition.
4. The sum is complemented and becomes the checksum.	4. The sum is complemented and becomes the new checksum.
5. The checksum is sent with the data.	5. If the value of the checksum is 0, the message is accepted; otherwise, it is rejected.

Algorithm

We can use the flow diagram of Figure below to show the algorithm for calculation of the checksum. A program in any language can easily be written based on the algorithm. Note that the first loop just calculates the sum of the data units in two's complement; the second loop wraps the extra bits created from the two's complement calculation to simulate the calculations in one's complement.



Other Approaches to the Checksum

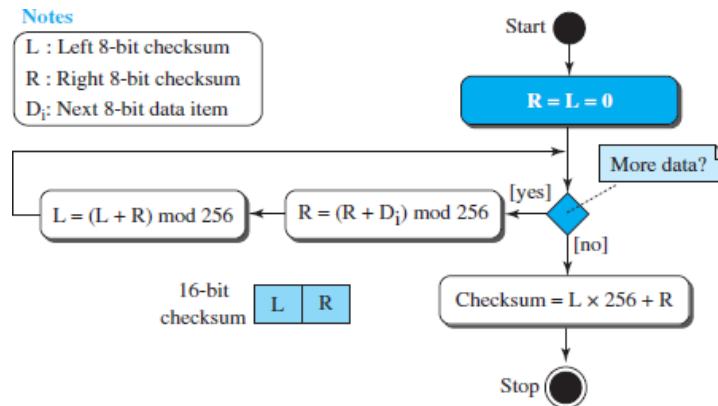
As mentioned before, there is one major problem with the traditional checksum calculation.

If two 16-bit items are transposed in transmission, the checksum cannot catch this error.

This can be overcome by: Fletcher and Adler.

Fletcher Checksum

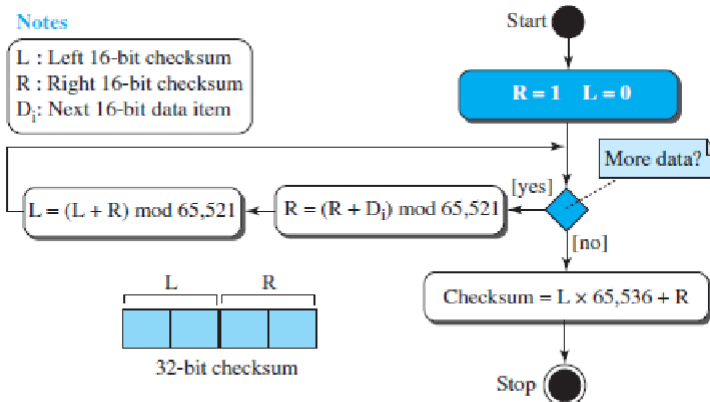
- ➔ The Fletcher checksum was devised to weight each data item according to its position. Fletcher has proposed two algorithms: 8-bit and 16-bit.
- ➔ The 8-bit Fletcher is calculated over data octets (bytes) and creates a 16-bit checksum. The calculation is done modulo 256 (28), which means the intermediate results are divided by 256 and the remainder is kept.
- ➔ The algorithm uses two accumulators, L and R. The first simply adds data items together; the second adds a weight to the calculation.
- ➔ The 16-bit Fletcher checksum is similar to the 8-bit Fletcher checksum, but it is calculated over 16-bit data items and creates a 32-bit checksum. The calculation is done modulo 65,536.



Adler Checksum

➔ The Adler checksum is a 32-bit checksum. Figure shows a simple algorithm in flowchart form. It is similar to the 16-bit Fletcher with three differences.

1. Calculation is done on single bytes instead of 2 bytes at a time.
2. The modulus is a prime number (65,521) instead of 65,536.
3. L is initialized to 1 instead of 0. It has been proved that a prime modulo has a better detecting capability in some combinations of data.



Data Link Control

DLC SERVICES

The two main functions of the data link layer are data link control and media access control. The first, data link control, deals with the design and procedures for communication between two adjacent nodes: node-to-node communication.

Data link control functions include framing, flow and error control, and software implemented Protocols, that provides smooth and reliable transmission of frames between nodes.

Framing

- ➔ Data transmission in the physical layer means moving bits in the form of a signal from the source to the destination.
- ➔ The data link layer needs to pack bits into frames, so that each frame is distinguishable from another.
- ➔ Framing adds a sender address and a destination address. The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt.
- ➔ Frames can be of fixed or variable size.

Fixed-Size Framing

In fixed-size framing, the size itself can be used as a delimiter.

Variable-Size Framing(differentiate b/n Character-oriented & Bit-oriented approach)

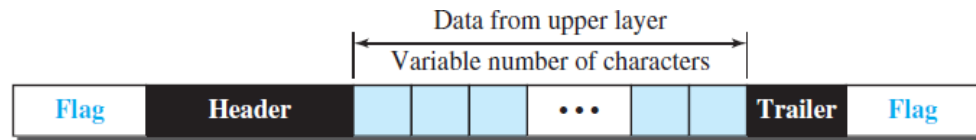
In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Historically, two approaches were used for this purpose:

1. Character-oriented approach.
2. Bit-oriented approach

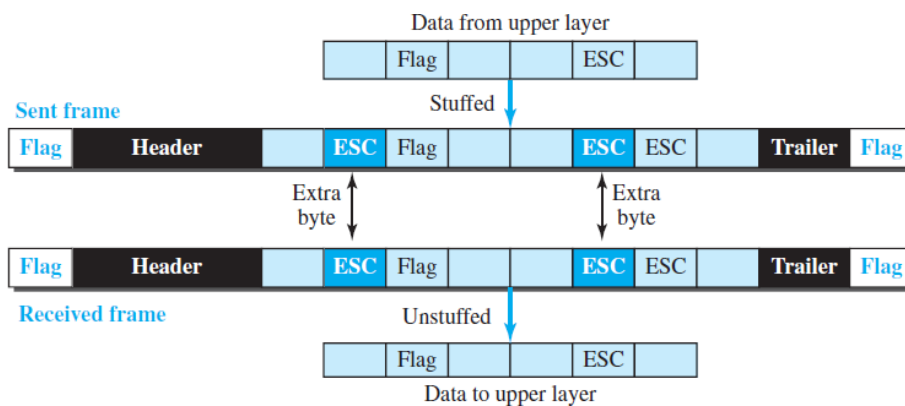
Character-Oriented Protocols

- ➔ In a character-oriented protocol, data to be carried are 8-bit characters I .
- ➔ The header, carries the source and destination addresses and other control information
- ➔ The trailer, carries error detection or error correction redundant bits.

- ➔ To separate one frame from the next, an 8-bit (1-byte) **flag** is added at the beginning and the end of a frame.
- ➔ The flag, composed of protocol-dependent special characters, signals the start or end of a frame.

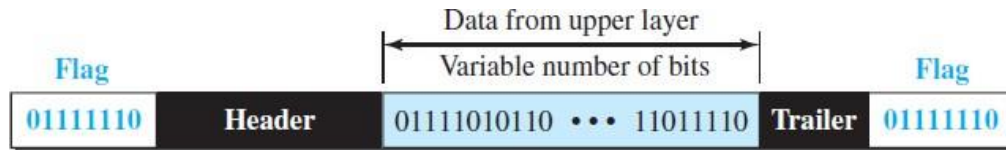


- ➔ If the receiver encounters the flag pattern in the middle of the data, thinks it has reached the end of the frame.
- ➔ To fix this problem, a **byte-stuffing** strategy was added to character-oriented framing.
- ➔ The data section is stuffed with an extra byte if it has data similar to flag.. This byte is usually called the escape character (ESC), which has a predefined bit pattern.
- ➔ The receiver removes ESC from the data section and treats the next character as data, not a delimiting flag.
- ➔ If the escape character is part of the text, an extra one is added to show that the second one is part of the text.

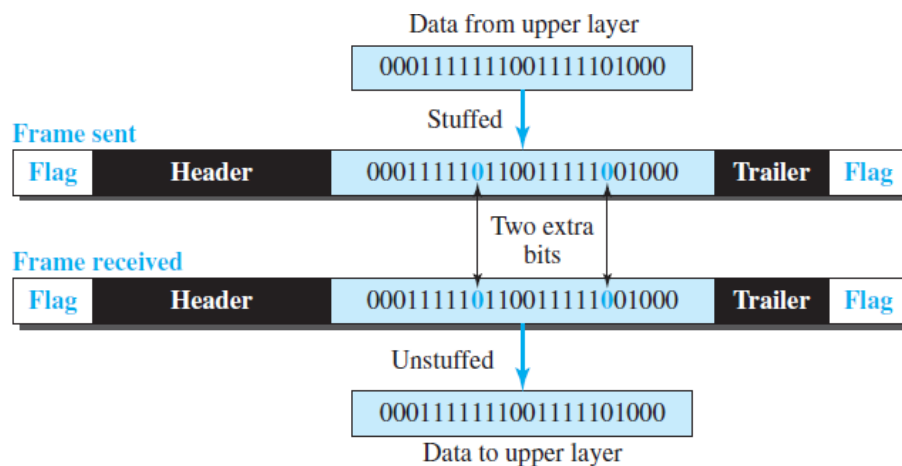


Bit-Oriented Protocols

- ➔ Most protocols use a special 8-bit pattern flag 01111110 as the delimiter to define the beginning and the end of the frame.



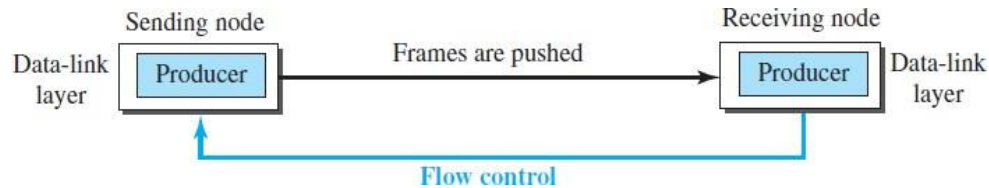
- ➔ If the flag pattern appears in the data, a single 1bit is stuffed to prevent the pattern from looking like a flag. The strategy is called bit stuffing.
- ➔ In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added which is removed from the data by the receiver.



FLOW AND ERROR CONTROL

Flow Control

- ➔ Flow control coordinates the amount of data that can be sent before receiving an acknowledgment.
- ➔ In communication at the data-link layer, we are dealing with four entities: network and data-link layers at the sending node and network and data-link layers at the receiving node.



- ➔ The data-link layer at the sending node tries to push frames toward the data-link layer at the receiving node.
- ➔ If the receiving node cannot process and deliver the packet to its network at the same rate that the frames arrive, it becomes overwhelmed with frames.
- ➔ Flow control in this case can be feedback from the receiving node to the sending node to stop or slow down pushing frames.

Buffers

- ➔ A buffer is a set of memory locations that can hold packets at the sender and receiver. It is usually used to implement flow control.
- ➔ When the buffer of the receiving data-link layer is full, it informs the sending data-link layer to stop pushing frames.

Error Control

Error control at the data-link layer is implemented using one of the following two methods.

- ➔ If the frame is corrupted, it is silently discarded; if it is not corrupted, the packet is delivered to the network layer.
- ➔ if the frame is corrupted, it is silently discarded; if it is not corrupted, an acknowledgment is sent (for the purpose of both flow and error control) to the sender.

Combination of Flow and Error Control

Flow and error control can be combined. In a simple situation, the acknowledgment that is sent for flow control can also be used for error control to tell the sender the packet has arrived uncorrupted.

Connectionless and Connection-Oriented

A DLC protocol can be either connectionless or connection-oriented.

Connectionless Protocol

In a connectionless protocol, frames are sent from one node to the next without any relationship between the frames; each frame is independent. The frames are not numbered and there is no sense of ordering.

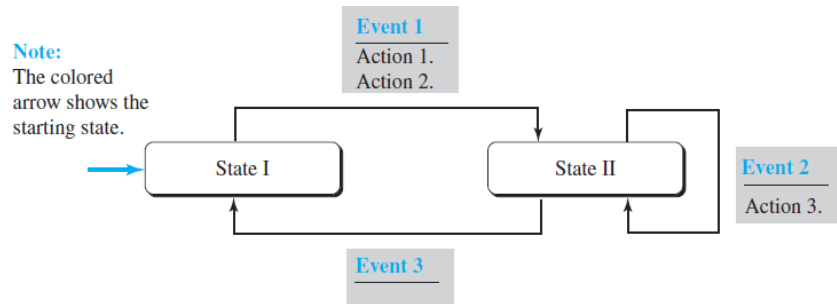
Connection-Oriented Protocol

In a connection-oriented protocol, a logical connection should first be established between the two nodes (setup phase). After all frames that are somehow related to each other are transmitted (transfer phase), the logical connection is terminated (teardown phase). In this type of communication, the frames are numbered and sent in order

DATA-LINK LAYER PROTOCOLS

Traditionally four protocols have been defined for the data-link layer to deal with flow and error control: Simple, Stop-and-Wait, Go-Back-N, and Selective-Repeat. We discuss the first two.

- ➔ The behavior of a data-link-layer protocol can be better shown as a **finite state machine (FSM)**.
- ➔ An FSM is thought of as a machine with a finite number of states. The machine is always in one of the states until an *event* occurs.
- ➔ Each event is associated with two reactions: defining the list (possibly empty) of actions to be performed and determining the next state (which can be the same as the current state).
- ➔ One of the states must be defined as the initial state, the state in which the machine starts when it turns on.
- ➔ Rounded-corner rectangles to show states, colored text to show events, and regular black text to show actions.
- ➔ A horizontal line is used to separate the event from the actions, although later we replace the horizontal line with a slash.
- ➔ The arrow shows the movement to the next state.

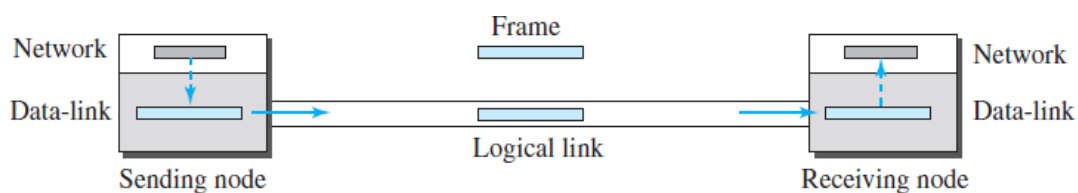


The figure shows a machine with three states. There are only three possible events and three possible actions.

1. The machine starts in state I. If event 1 occurs, the machine performs actions 1 and 2 and moves to state II.
2. When the machine is in state II, two events may occur. If event 1 occurs, the machine performs action 3 and remains in the same state, state II.
3. If event 3 occurs, the machine performs no action, but move to state I.

Simple Protocol

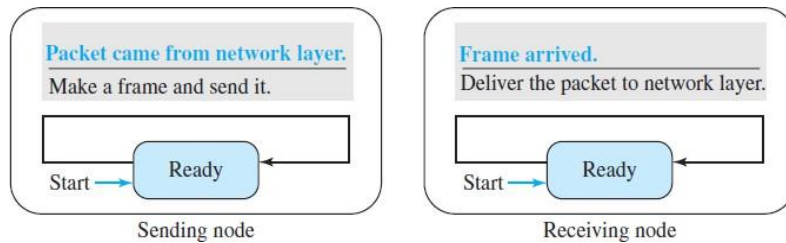
- ➔ It has no flow or error control. It is a unidirectional protocol in which data frames are traveling in only one direction—from the sender to receiver.
- ➔ We assume that the receiver can immediately handle any frame it receives.



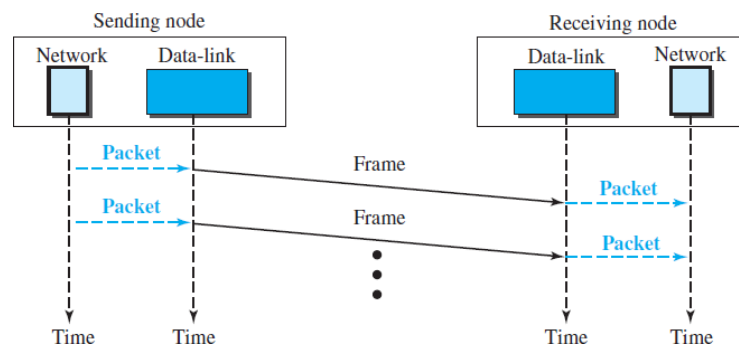
FSMs

- ➔ Each FSM has only one state, the *ready state*. The sending machine remains in the ready state until a request comes from the process in the network layer.
- ➔ When this event occurs, the sending machine encapsulates the message in a frame and sends it to the receiving machine.

- ➔ The receiving machine remains in the ready state until a frame arrives from the sending machine.
- ➔ When this event occurs, the receiving machine decapsulates the message out of the frame and delivers it to the process at the network layer.



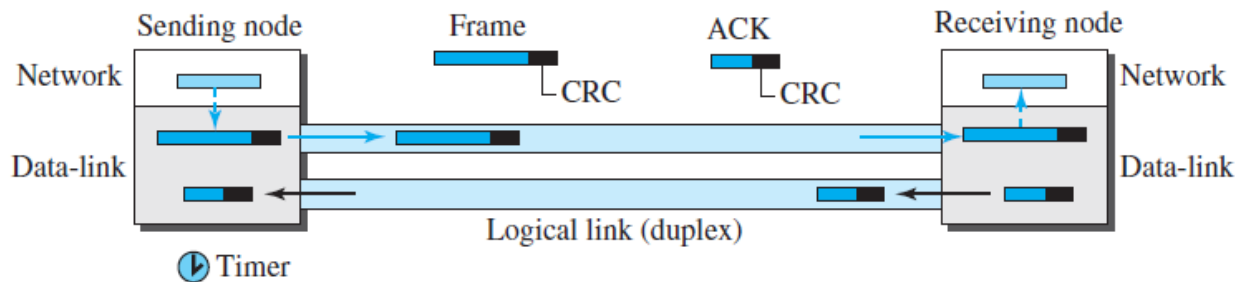
An example of communication using this protocol is below. It is very simple. The sender sends frames one after another without even thinking about the receiver.



Stop-and-Wait Protocol

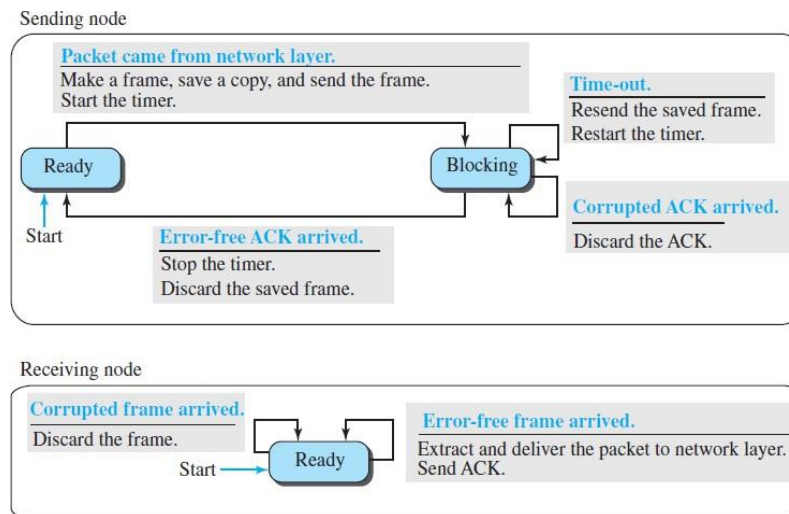
- ➔ Use both flow and error control. The sender sends one frame at a time and waits for an acknowledgment, before sending the next one.
- ➔ To detect corrupted frames, we need to add a CRC to each data frame. When a frame arrives at the receiver site, it is checked. If its CRC is incorrect, the frame is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a frame was either corrupted or lost.
- ➔ Every time the sender sends a frame, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame.

- ➔ If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted. This means that the sender needs to keep a copy of the frame



FSMs

Figure shows the FSMs for our primitive Stop-and-Wait protocol.



Sender States

The sender is initially in the ready state, but it can move between the ready and blocking state **Ready State**. When the sender is in this state, it is only waiting for a packet from the network layer. If a packet comes from the network layer, the sender creates a frame, saves a copy of the frame, starts the only timer and sends the frame. The sender then moves to the blocking state. **Blocking State**. When the sender is in this state, three events can occur:

- If a time-out occurs, the sender resends the saved copy of the frame and restarts the timer.
- If a corrupted ACK arrives, it is discarded.
- If an error-free ACK arrives, the sender stops the timer and discards the saved copy of the frame. It then moves to the ready state.

Receiver

The receiver is always in the *ready* state. Two events may occur:

- If an error-free frame arrives, the message in the frame is delivered to the network layer and an ACK is sent.
- If a corrupted

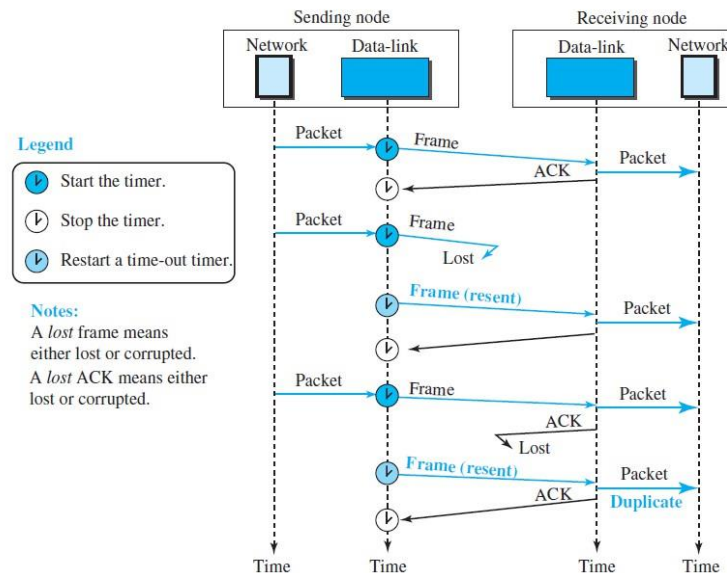
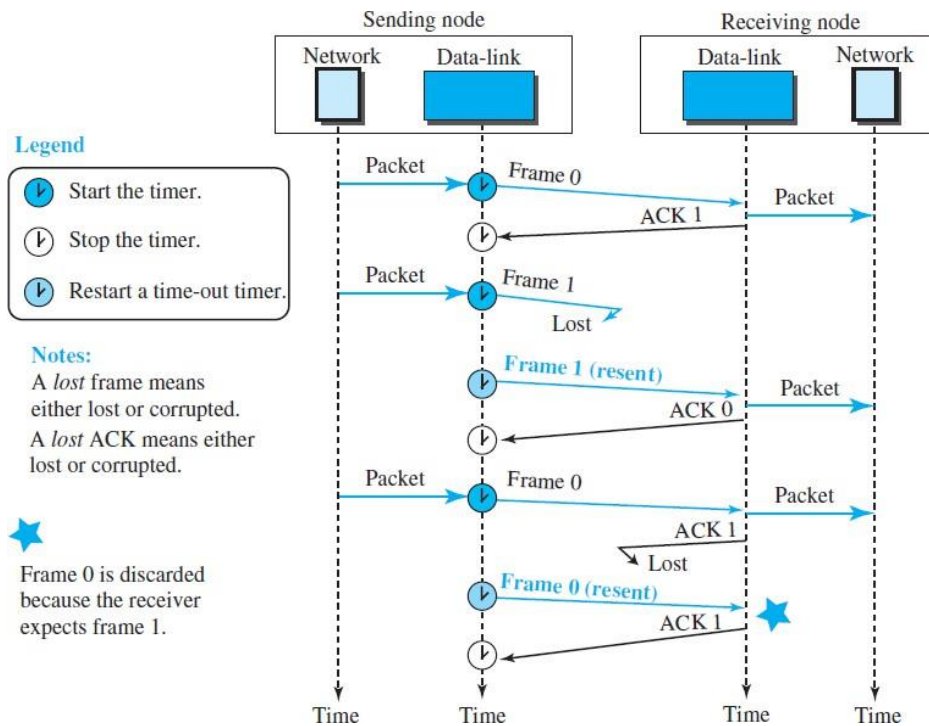


Figure shows an example. The first frame is sent and acknowledged. The second frame is sent, but lost. After time-out, it is resent. The third frame is sent and acknowledged, but the acknowledgment is lost. The frame is resent. However, there is a problem with this scheme. The Network layer at the receiver site receives two copies of the third packet, which is not right.

Sequence and Acknowledgment Numbers

- ➔ Duplicate packets, as much as corrupted packets, need to be avoided. As an example, assume we are ordering some item online.
- ➔ If each packet defines the specification of an item to be ordered, duplicate packets mean ordering an item more than once.
- ➔ To correct the problem in Example, we need to add **sequence numbers** to the data frames and **acknowledgment numbers** to the ACK frames.
- ➔ Numbering in this case is very simple. Sequence numbers are 0, 1, 0, 1, 0, 1, . . . ; the acknowledgment numbers can also be 1, 0, 1, 0, 1, 0, . . .
- ➔ An acknowledgment number always defines the sequence number of the next frame to receive.



Piggybacking

A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.

HDLC

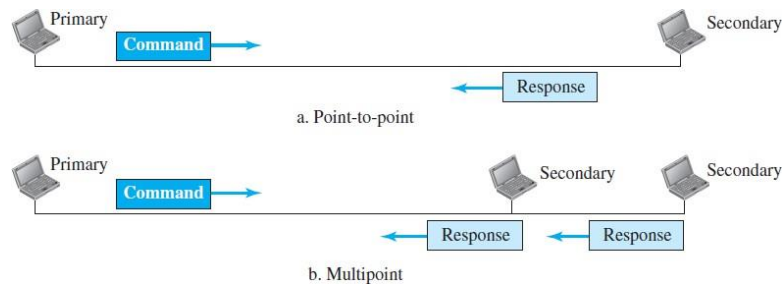
High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the Stop-and-Wait protocol.

Configurations and Transfer Modes

HDLC provides two common transfer modes that can be used in different configurations:

Normal Response Mode

In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond. The NRM is used for both point-to-point and multiple-point links, as shown in Figure.



Asynchronous Balanced Mode

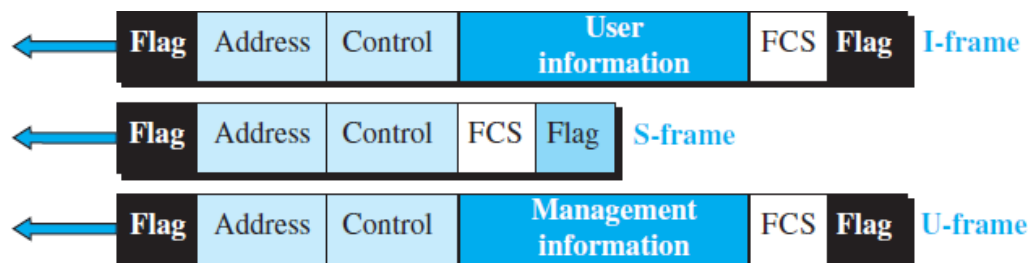
In asynchronous balanced mode (ABM), the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure. This is the common mode today.



Framing

To provide the flexibility necessary to support all the options possible in the modes and configurations just described, HDLC defines three types of frames:

1. I-frames (information frames) are used to transport user data and control information relating to user data (piggybacking).
2. S-frames (supervisory frames) are used only to transport control information. V-frames are reserved for system management.
3. U frames (unnumbered frames) Information carried by U-frames is intended for managing the link itself.



Flag field-The flag field of an HDLC frame is an 8-bit sequence with the bit pattern 01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.

Address field- The second field of an HDLC frame contains the address of the secondary station. If a primary station created the frame, it contains a *to* address. If a secondary creates the frame, it contains a *from* address. An address field can be 1 byte or several bytes long, depending on the needs of the network.

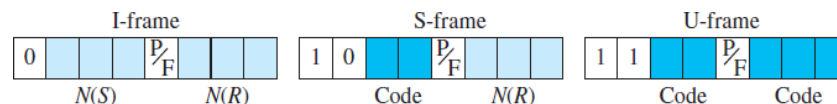
Control field -The control field is a 1- or 2-byte segment of the frame used for flow and error control. The interpretation of bits in this field depends on the frame type.

Information field-The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.

FCS field-The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte ITU-T CRC.

Control Field

The control field determines the type of frame and defines its functionality.



Control Field for I-Frames

I-frames are designed to carry user data from the network layer. In addition, they can include flow and error control information (piggybacking). The subfields in the control field are used to define these functions.

- The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame.
- The next 3 bits, called $N(S)$, define the sequence number of the frame.
- The last 3 bits, called $N(R)$, correspond to the acknowledgment number when piggybacking is used.
- The P/F field is a single bit with a dual purpose. It has meaning only when it is set (bit = 1) and can mean poll or final. It means *poll* when the frame is sent by a primary station to a secondary. It means *final* when the frame is sent by a secondary to a primary.

Control Field for S-Frames

Supervisory frames are used for flow and error control whenever piggybacking is either impossible or inappropriate. S-frames do not have information fields. If the first 2 bits of the control field is 10, this means the frame is an S-frame. The last 3 bits, called $N(R)$, corresponds to the acknowledgment number (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame. The 2 bits called code is used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames, as described below:

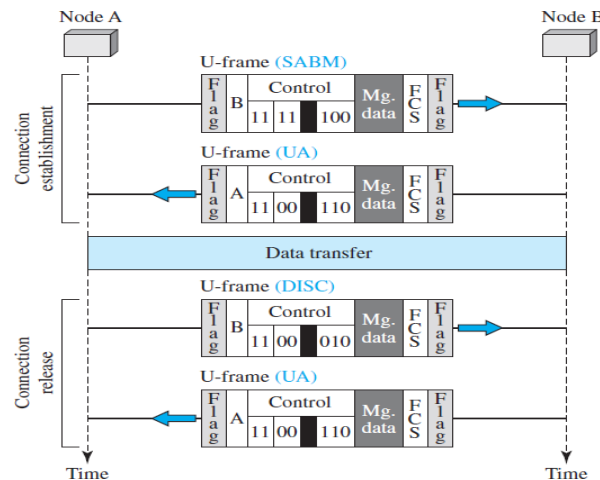
1. Receive ready (RR). If the value of the code subfield is 00, it is an RR S-frame. Receive not ready (RNR). If the value of the code subfield is 10, it is an RNR S-frame. The value of NCR is the acknowledgment number.
2. Reject (REJ). If the value of the code subfield is 01, it is a REJ S-frame. The value of NCR is the negative acknowledgment number.
3. Selective reject (SREJ). If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. The value of $N(R)$ is the negative acknowledgment number.

Control Field for U-Frames

Unnumbered frames are used to exchange session management and control information between connected devices.

U-frame codes are divided into two sections: a 2-bit prefix before the P/F bit and a 3-bit suffix after the P/F bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames.

Figure shows how U-frames can be used for connection establishment and connection release. Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes (not shown in the figure). After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment).



POINT-TO-POINT PROTOCOL

Although HDLC is a general protocol that can be used for both point-to-point and multipoint configurations, the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP). PPP is by far the most common. PPP provides several services:

1. PPP defines the format of the frame to be exchanged between devices. PPP defines how two devices can negotiate the establishment of the link and the exchange of data.
2. PPP defines how network layer data are encapsulated in the data link frame.
3. PPP defines how two devices can authenticate each other.
4. PPP provides multiple network layer services supporting a variety of network layer protocols.
5. PPP provides connections over multiple links.
6. PPP provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.

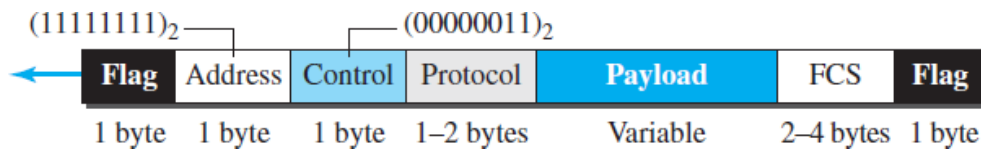
On the other hand, to keep PPP simple, several services are missing:

1. PPP does not provide flow control. A sender can send several frames one after another with no concern about overwhelming the receiver.
2. PPP has a very simple mechanism for error control.
3. PPP does not provide a sophisticated addressing mechanism to handle frames in a multipoint configuration.

Framing

PPP is a byte-oriented protocol. Framing is done according to byte-oriented protocols.

Frame Format



Flag. A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110.

Address. The address field in this protocol is a constant value and set to 11111111 (broadcast address). During negotiation (discussed later), the two parties may agree to omit this byte.

Control. This field is set to the constant value 11000000.

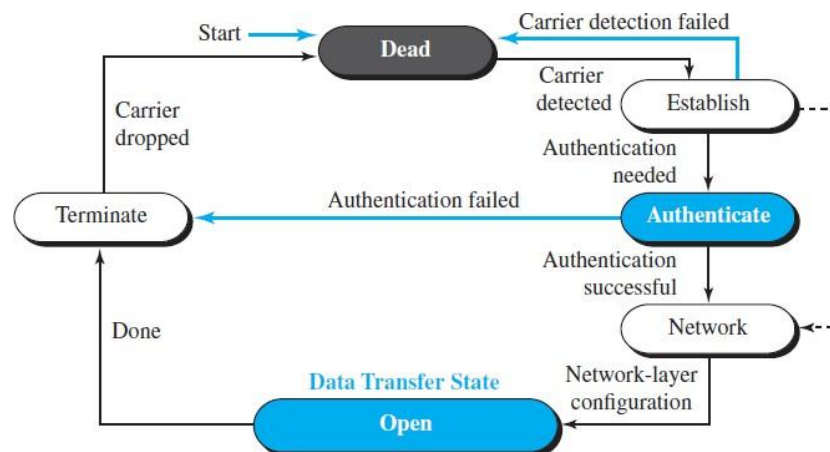
Protocol. The protocol field defines what is being carried in the data field: either user data or other information. This field is by default 2 bytes long, but the two parties can agree to use only 1 byte.

Payload field. This field carries either the user data or other information. The data field is a sequence of bytes with the default of a maximum of 1500 bytes; but this can be changed during negotiation.

FCS. The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC.

Transition Phases

A PPP connection goes through phases which can be shown in a transition phase diagram.



Dead- In the dead phase the link is not being used. There is no active carrier and the line is quiet.

Establish- When one of the nodes starts the communication, the connection goes into this phase. In this phase, options are negotiated between the two parties..

Authenticate- The authentication phase is optional; the two nodes may decide, during the establishment phase, not to skip this phase.

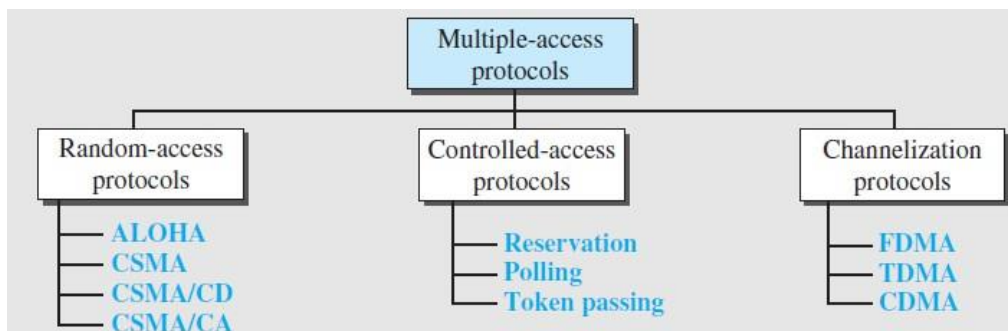
Network- In the network phase, negotiation for the network layer protocols takes place.

Open. In the open phase, data transfer takes place.

Terminate. In the termination phase the connection is terminated.

Media Access control

When nodes or stations are connected and use a common link, called a *multipoint* or *broadcast link*, we need a multiple-access protocol to coordinate access to the link. Many protocols have been devised to handle access to a shared link. All of these protocols belong to a sublayer in the data-link layer called *media access control (MAC)*. We categorize them into three groups.



RANDOM ACCESS

In random access or contention methods, no station is superior to another station. A station that has data to send uses a procedure defined by the protocol to make a decision on whether or not to send.

Two features give this method its name.

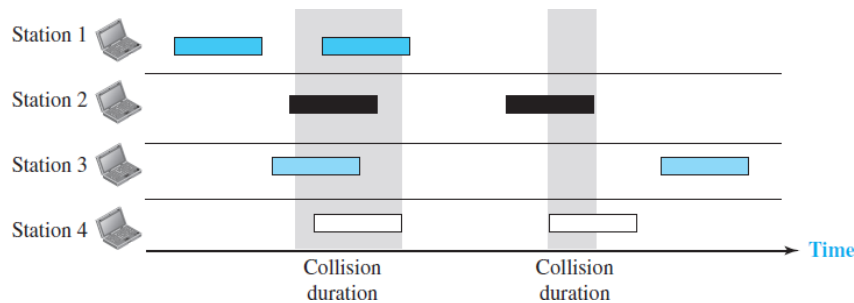
1. There is no scheduled time for a station to transmit. Transmission is random and hence called *random access*.
2. No rules specify which station should send next. Stations compete with one another to access the medium and hence called *contention* methods. If more than one station tries to send, there is an access conflict-collision-and the frames will be either destroyed or modified.

1. ALOHA

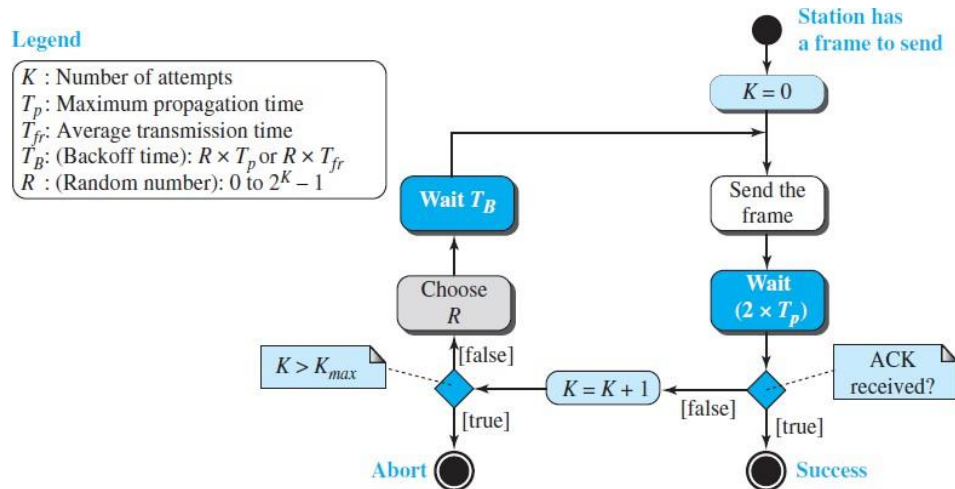
ALOHA, the earliest random access method was developed at the University of Hawaii in early 1970. The medium is shared between the stations and hence has potential arrangements.

Pure ALOHA

- ➔ The original ALOHA protocol is called pure ALOHA. The idea is that each station sends a frame whenever it has a frame to send.
- ➔ Since there is only one channel to share, there is the possibility of collision between frames from different stations.
- ➔ There are four stations (unrealistic assumption) that contend with one another for access to the shared channel. Only two frames survive: frame 1.1 from station 1 and frame 3.2 from station 3.

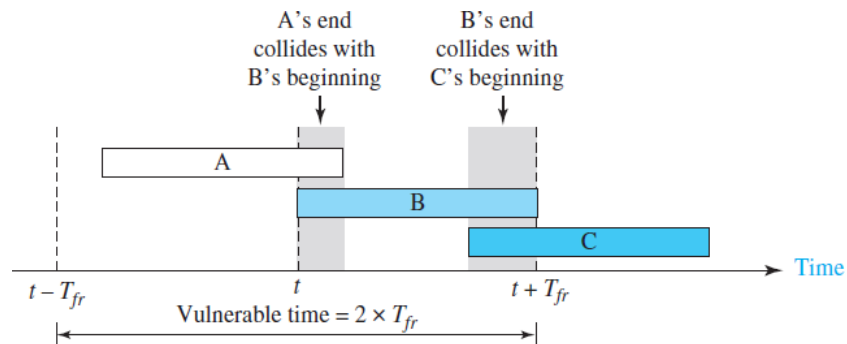


- ➔ The pure ALOHA protocol relies on acknowledgments from the receiver. If the acknowledgment does not arrive after a time-out period, the station resends the frame.
- ➔ If all these stations try to resend their frames after the time-out, the frames will collide again. Each station waits for a random time before resending, called the back-off time T_B .
- ➔ In the second method after a maximum number of retransmission attempts K_{max} a station must give up and try later.



Vulnerable time- Station A sends a frame at time t . Now imagine station B has already sent a frame between $t - T_{fr}$ and t . This leads to a collision between the frames from station A and station B. The end of B's frame collides with the beginning of A's frame. On the other hand, suppose that station C sends a frame between t and $t + T_{fr}$. Here, there is a collision between frames from station A and station C. The beginning of C's frame collides with the end of A's frame.

$$\text{Pure ALOHA vulnerable time} = 2 \times T_{fr}$$

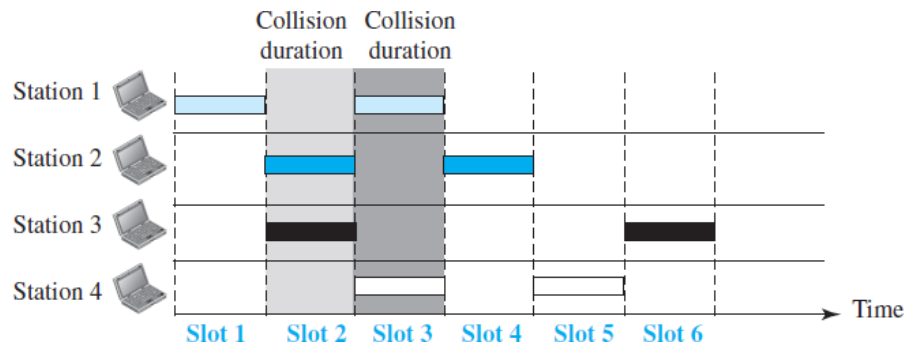


Throughput Let us call G the average number of frames generated by the system during one frame transmission time. Then it can be proved that the average number of successful transmissions for pure ALOHA is $S = G \times e^{-2G}$. The maximum throughput

$$S_{\max} = 0.184, \text{ for } G = 1.$$

2. Slotted ALOHA

In slotted ALOHA we divide the time into slots of T_{fr} s and force the station to send only at the beginning of the time slot.

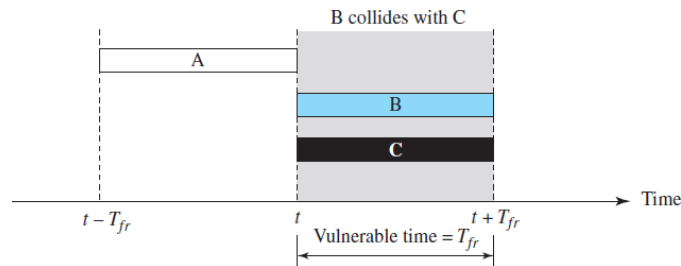


The vulnerable time for slotted ALOHA is one-half that of pure ALOHA.

$$\text{Slotted ALOHA vulnerable time} = T_{fr}$$

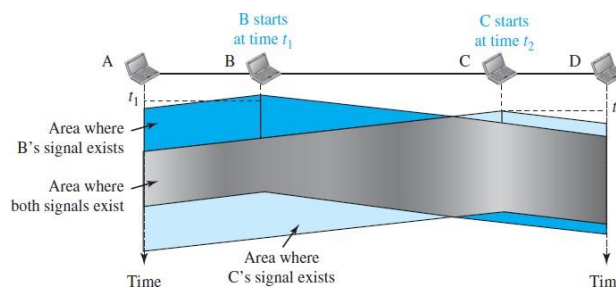
Throughput It can be proved that the average number of successful transmissions for ALOHA is

$S = G \times e^{-G}$. The maximum throughput S_{max} is 0.368, when $G = 1$.



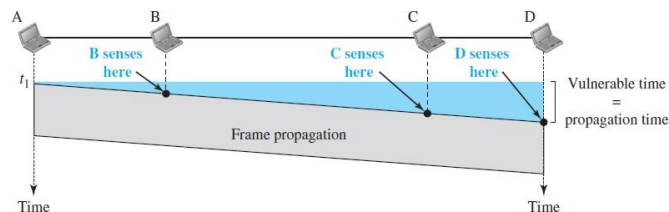
Carrier Sense Multiple Access (CSMA)

- ➔ Carrier sense multiple access (CSMA) requires that each station first listen to the medium before sending. The principle "sense before transmit" or "listen before talk."
- ➔ CSMA can reduce the possibility of collision, but it cannot eliminate it. Stations are connected to a shared channel.
- ➔ The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time for the first bit to reach every station and for every station to sense it.



Vulnerable Time

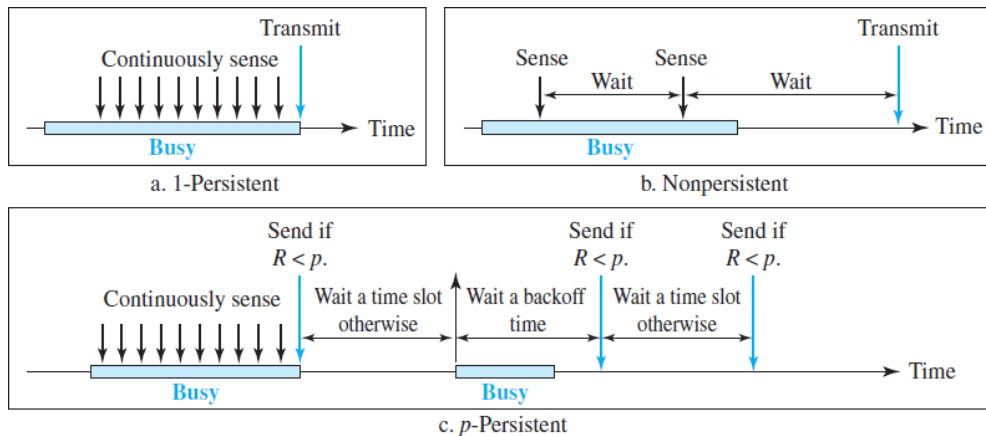
The vulnerable time for CSMA is the propagation time T_p . This is the time needed for a signal to propagate from one end of the medium to the other.

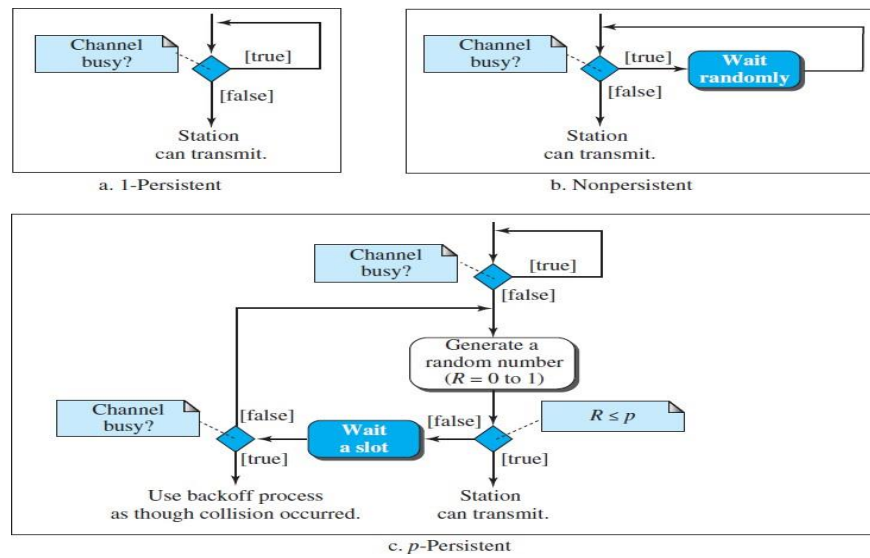


Persistence Methods

This method tells, What should a station do if the channel is busy? What should a station do if the channel is idle? Three methods have been devised

1. **I-persistent** - In this method, after the station finds the line idle, it sends its frame immediately.
2. **Nonpersistent**- In this method, a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits a random amount of time and then senses the line again.
3. **p-Persistent** - This method is used if the channel has time slots with a slot duration equal to or greater than the maximum propagation time. In this method, after the station finds the line idle it follows these steps:
 1. With probability p , the station sends its frame.
 2. With probability $q = 1 - p$, the station waits for the beginning of the next time slot and checks the line again.
 - a. If the line is idle, it goes to step 1.
 - b. If the line is busy, it acts as though a collision has occurred and uses the backoff procedure

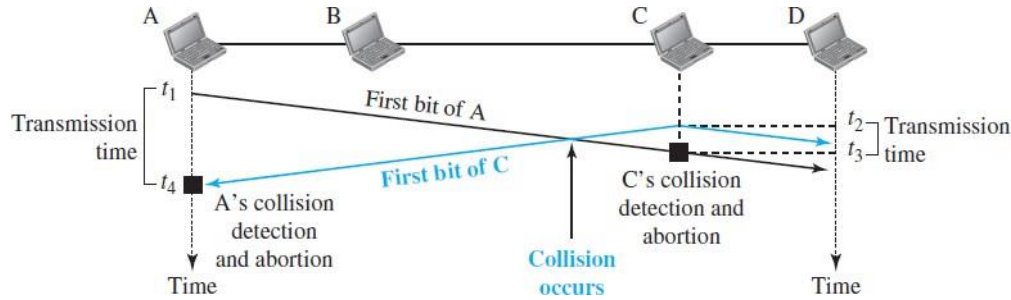




4. Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

In this method, a station monitors the medium after it sends a frame to see if the transmission was successful. If so, the station is finished. If, however, there is a collision, the frame is sent again.

1. At time t_1 , station A has executed its persistence procedure and starts sending the bits of its frame.
2. At time t_2 , station C has not yet sensed the first bit sent by A. Station C executes its persistence procedure and starts sending the bits in its frame, which propagate both to the left and to the right.
3. The collision occurs sometime after time t_2 . Station C detects a collision at time t_3 when it receives the first bit of A's frame.
4. Station C immediately aborts transmission.
5. Station A detects collision at time t_4 when it receives the first bit of C's frame; it also immediately aborts transmission.



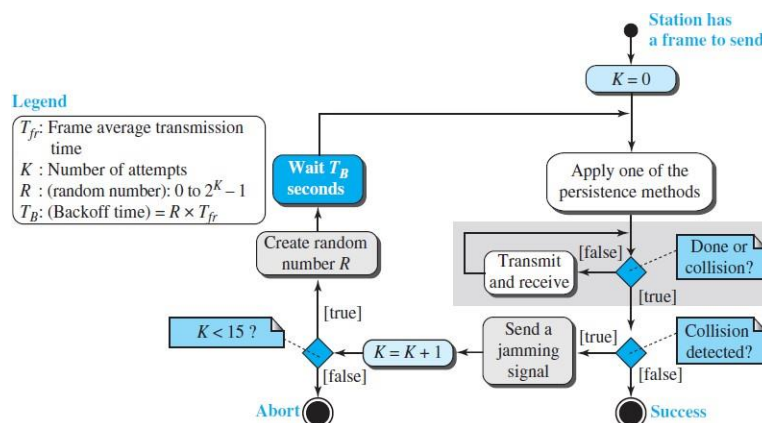
Minimum Frame Size

- ➔ For CSMA/CD to work, a restriction on the frame size is required.
- ➔ Before sending the last bit of the frame, the sending station must detect a collision, if any, because it does not keep a copy of the frame.
- ➔ Therefore, the frame transmission time T_{fr} must be at least two times the maximum propagation time T_p .

Procedure

It is similar to the one for the ALOHA protocol, but there are differences.

1. Addition of the persistence process.
2. In ALOHA, we first transmit the entire frame and then wait for an acknowledgment. In CSMA/CD, transmission and collision detection is a continuous process.
3. A short jamming signal that enforces the collision in case other stations have not yet sensed the collision.

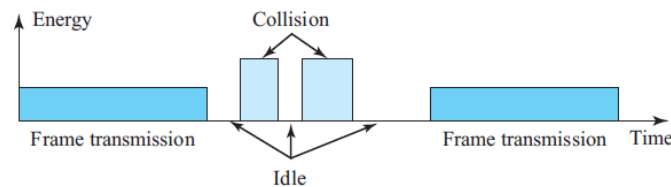


Energy Level

The level of energy in a channel can have three values: zero, normal, and abnormal.

- ➔ At the zero level, the channel is idle.
- ➔ At the normal level, a station has successfully captured the channel and is sending its frame.
- ➔ At the abnormal level, there is a collision and the level of the energy is twice the normal level.

A station that has a frame to send or is sending a frame needs to monitor the energy level to determine if the channel is idle, busy, or in collision mode.



Throughput

The throughput of CSMA/CD is greater than that of pure or slotted ALOHA. The maximum throughput occurs at a different value of G and is based on the persistence method and the value of p in the p-persistent approach.

1. For I-persistent method the maximum throughput is around 50 percent when $G = 1$.
2. For nonpersistent method, the maximum throughput can go up to 90 percent when G is between 3 and 8

5. Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA)

The basic idea behind CSMA/CD is that a station needs to be able to receive while transmitting to detect a collision.

- ➔ When there is no collision, the station receives one signal: its own signal. When there is a collision, the station receives two signals: its own signal and the signal transmitted by a second station.
- ➔ To distinguish between these two cases, the received signals in these two cases must be significantly different.

- ➔ In a wired network, the received signal has almost the same energy. In case of collision, the detected energy almost doubles.
- ➔ In a wireless network, much of the sent energy is lost in transmission. The received signal has very little energy. Collision may add very little additional energy. This is not useful for effective collision detection.
- ➔ Collision must be avoided on wireless network, using the following strategies

1. Interframe Space (IFS)

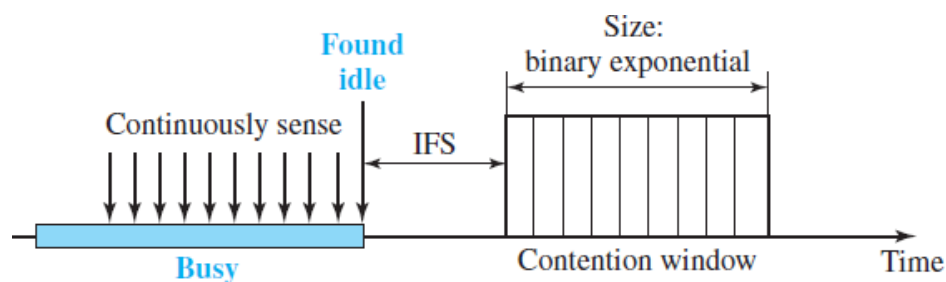
Collisions are avoided by deferring transmission even if the channel is found idle. When an idle channel is found, the station does not send immediately. It waits for a period of time called the interframe space or IFS.

2. Contention Window

The contention window is an amount of time divided into slots. A station that is ready to send chooses a random number of slots as its wait time. The number of slots in the window changes according to the binary exponential back-off strategy (set to one slot the first time and then doubles each time the station cannot detect an idle channel after the IFS time).

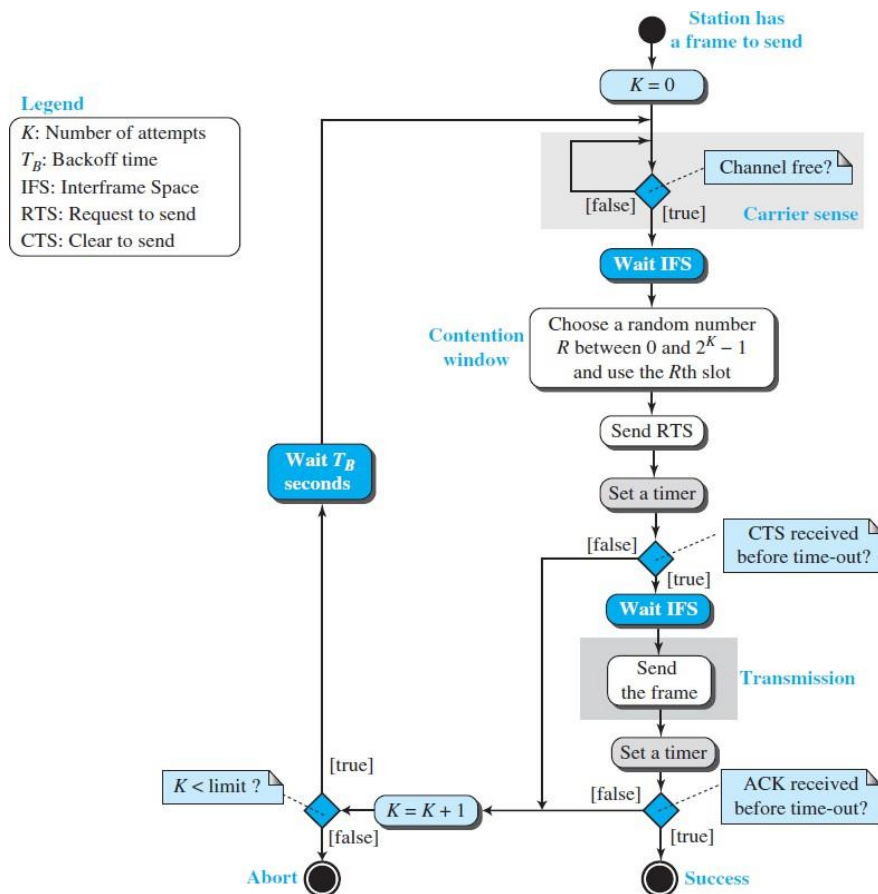
3. Acknowledgment

With all these precautions, there still may be a collision resulting in destroyed data. In addition, the data may be corrupted during the transmission. The positive acknowledgment and the time-out timer can help guarantee that the receiver has received the frame.



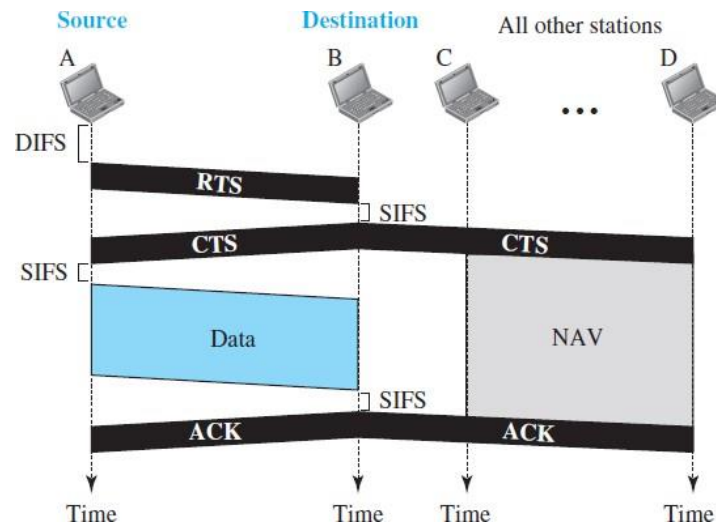
Procedure

- ➔ The channel needs to be sensed before and after the IFS. The channel also needs to be sensed during the contention time.
- ➔ For each time slot of the contention window, the channel is sensed.
- ➔ If it is found idle, the timer continues; if the channel is found busy, the timer is stopped and continues after the timer becomes idle again.



Frame Exchange Time Line

Figure shows the exchange of data and control frames in time.



1. Before sending a frame, the source station senses the medium by checking the energy level at the carrier frequency.
 - a. The channel uses a persistence strategy with backoff until the channel is idle.
 - b. After the station is found to be idle, the station waits for a period of time called the *DCF interframe space (DIFS)*; then the station sends a control frame called the *request to send (RTS)*.
2. After receiving the RTS and waiting a period of time called the *short interframe space (SIFS)*, the destination station sends a control frame, called the *clear to send (CTS)*, to the source station. This control frame indicates that the destination station is ready to receive data.
3. The source station sends data after waiting an amount of time equal to SIFS.
4. The destination station, after waiting an amount of time equal to SIFS, sends an acknowledgment to show that the frame has been received.

Network Allocation Vector

Collision avoidance aspect of this protocol is accomplished by a feature called NAV.

- ➔ When a station sends an RTS frame, it includes the duration of time that it needs to occupy the channel.
- ➔ The stations that are affected by this transmission create a timer called a **network allocation vector (NAV)** that shows how much time must pass before these stations are allowed to check the channel for idleness.
- ➔ Each station, before sensing the physical medium to see if it is idle, first checks its NAV to see if it has expired.

Collision During Handshaking

Two or more stations may try to send RTS frames at the same time. These control frames may collide. However, because there is no mechanism for collision detection, the sender assumes there has been a collision if it has not received a CTS frame from the receiver. The back off strategy is employed, and the sender tries again.

CONTROLLED ACCESS

In controlled access, the stations consult one another to find which station has the right to send. A station cannot send unless it has been authorized by other stations.

1. Reservation

- ➔ In the reservation method, a station needs to make a reservation before sending data.
- ➔ Time is divided into intervals.
- ➔ In each interval, a reservation frame precedes the data frames sent in that interval.

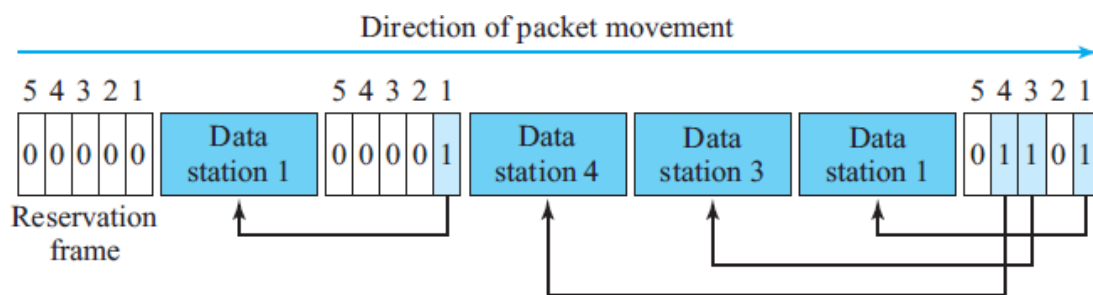
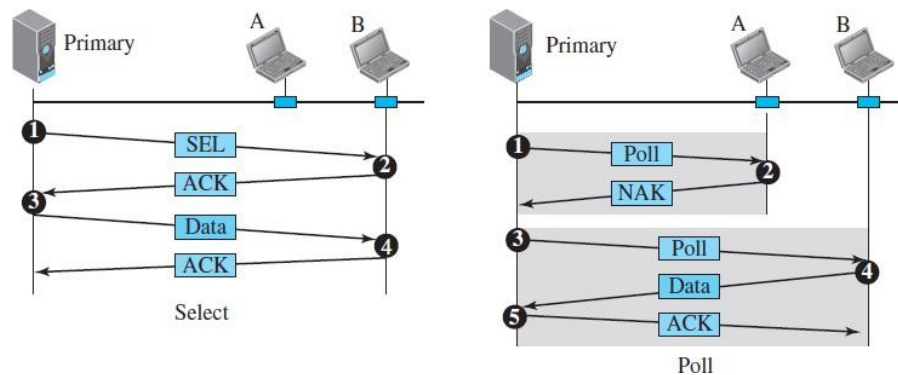


Figure shows a situation with five stations and a five-minislot reservation frame. In the first interval, only stations 1, 3, and 4 have made reservations. In the second interval, only station 1 has made a reservation.

2. Polling

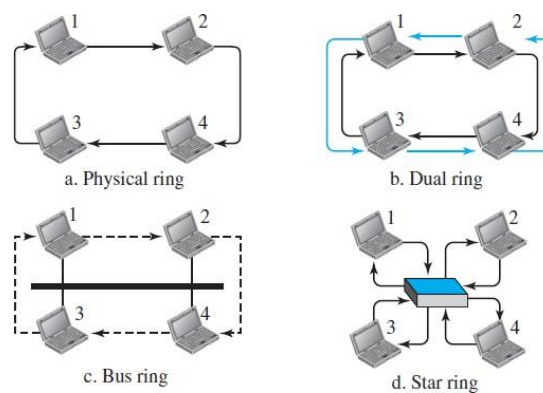
- ➔ Polling works with topologies in which one device is designated as a primary station and the other devices are secondary stations.
- ➔ The primary device controls the link; the secondary devices follow its instructions.
- ➔ The primary device, therefore, is always the initiator of a session
- ➔ If the primary wants to receive data, it asks the secondary's if they have anything to send; this is called **poll** function. Secondary responds with DATA if any, else negative ack (NAK) if it has nothing to send.
- ➔ If the primary wants to send data, it tells the secondary to get ready to receive; this is called **select** function. It transmits SEL and waits for acknowledgement from secondary before sending data.



3. Token Passing

- ➔ In the token-passing method, the stations in a network are organized in a logical ring.
- ➔ A special packet called a token circulates through the ring.
- ➔ The possession of the token gives the station the right to access the channel and send its data.

- ➔ When a station has some data to send, it holds the token and sends its data and releases the token once data is sent.
- ➔ When a station receives the token and has no data to send, it just passes the data to the next station.
- ➔ Stations must be limited in the time they can have possession of the token.
- ➔ The token must be monitored to ensure it has not been lost or destroyed.
- ➔ Stations do not have to be physically connected in a ring; the ring can be a logical one.



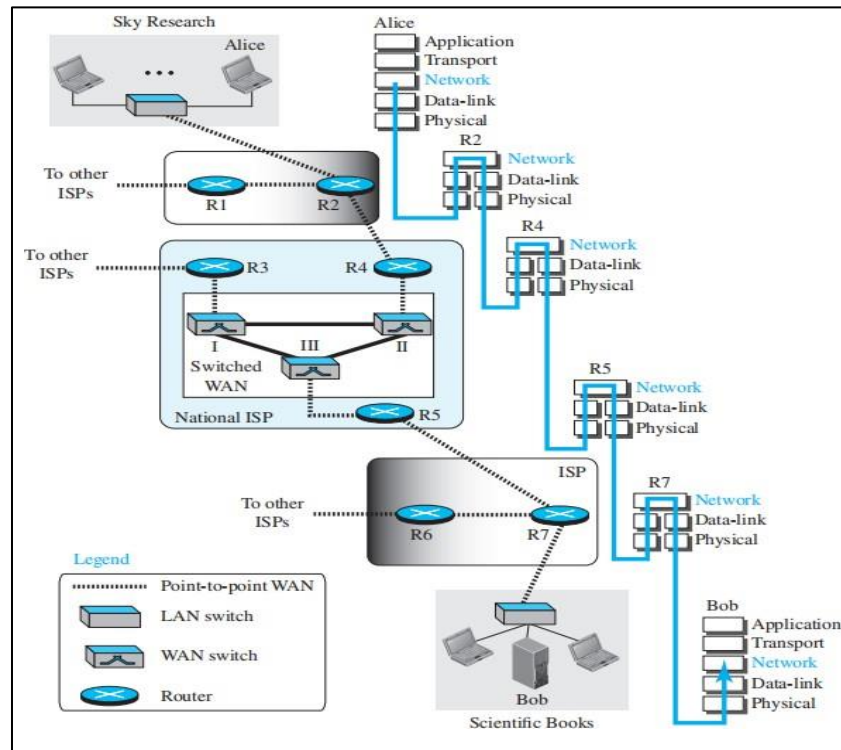
- ➔ The problem with this topology is that if one of the links-the medium between two adjacent stations fails, the whole system fails.
- ➔ The dual ring topology uses a second (auxiliary) ring which operates in the reverse direction compared with the main ring.

Module 3

Network Layer

Introduction to network layer

The network layer in the TCP/IP protocol suite is responsible for the host-to-host delivery of datagrams. It provides services to the transport layer and receives services from the data-link layer.



The following are the services provided by the network layer

1. Packetizing

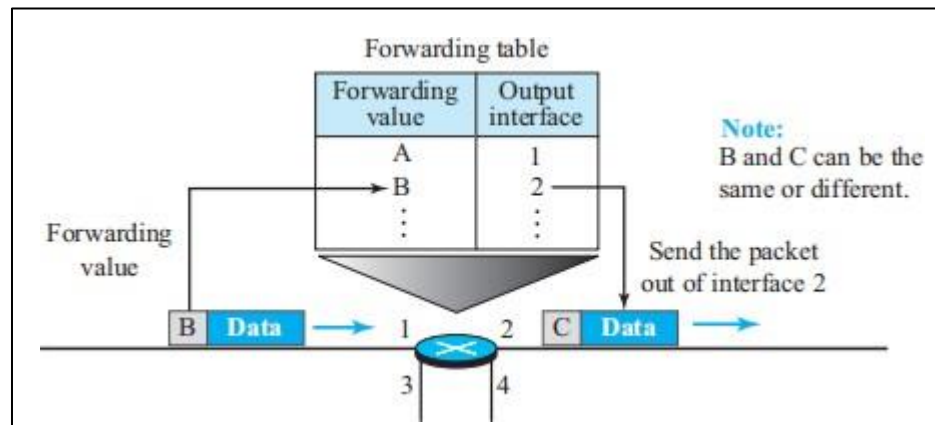
The first duty of the network layer is definitely **packetizing**: encapsulating the payload (data received from upper layer) in a network-layer packet at the source and decapsulating the payload from the network-layer packet at the destination.

2. Routing

The network layer is responsible for routing the packet from its source to the destination. There will be more than one route from the source to the destination. The network layer is responsible for finding the best one among these possible routes.

3. Forwarding

Forwarding can be defined as the action applied by each router when a packet arrives at one of its interfaces. The decision-making table a router normally uses for applying this action is sometimes called the *forwarding table* and sometimes the *routing table*.



Other Services

1. **Error Control**
2. **Flow Control**
3. **Congestion Control**

Congestion in the network layer is a situation in which too many datagrams are present in an area of the Internet. Congestion may occur if the number of datagrams sent by source computers is beyond the capacity of the network or routers.

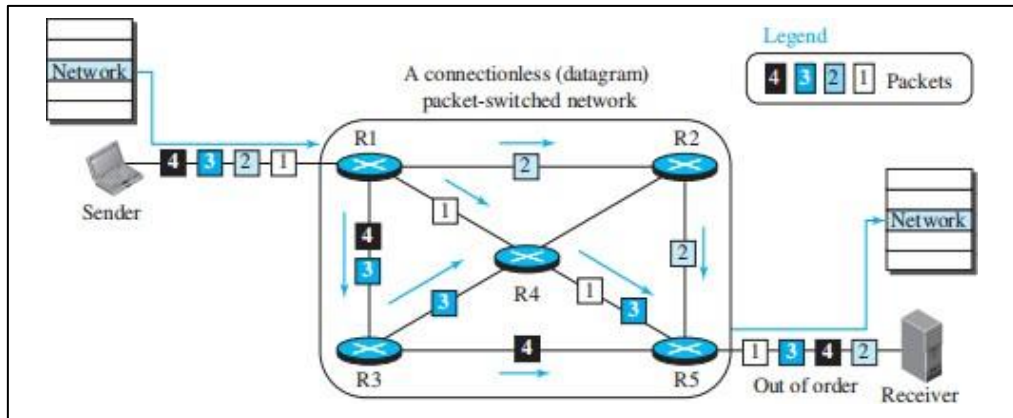
4. **Quality of Service**

As the Internet has allowed new applications such as multimedia communication (in particular real-time communication of audio and video), the quality of service (QoS) of the communication has become more and more important. The Internet has thrived by providing better quality of service to support these applications.

5. **Security**

PACKET SWITCHING

When the Internet was first designed, the network layer provided a connectionless service. Each packet is treated independently, with no relationship to other packets. The network layer's role is to deliver packets from the source to the destination, and packets may take different paths to get there.

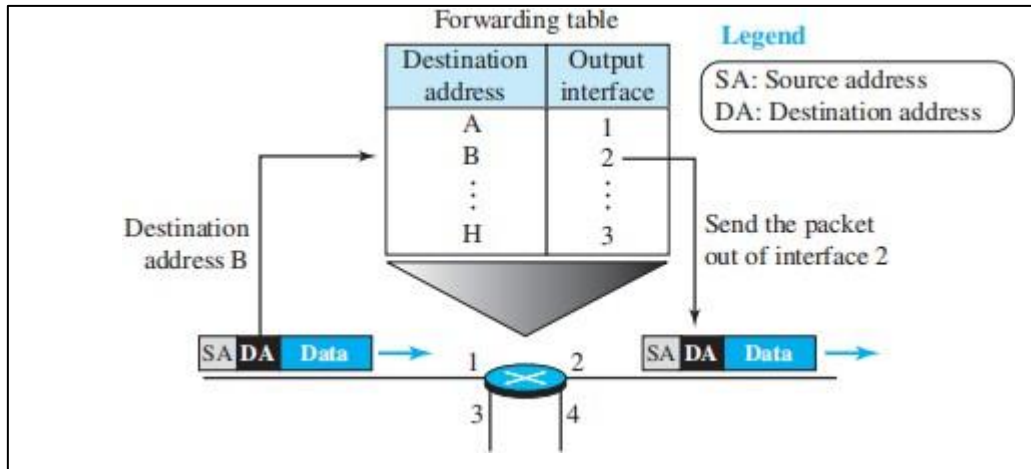


At the network layer, routers act like switches, connecting an input port to one or more output ports, similar to how an electrical switch works. In data communication, there are two main switching types: circuit switching and packet switching. Circuit switching is mostly used at the physical layer, but at the network layer, only packet switching is used because data is sent as packets.

When a message is sent from one device to another, it is divided into packets. The source sends these packets one by one, and the destination waits for all packets to arrive before reconstructing the message. To send the packets, the network uses two routing methods: the **datagram approach** and the **virtual circuit approach**.

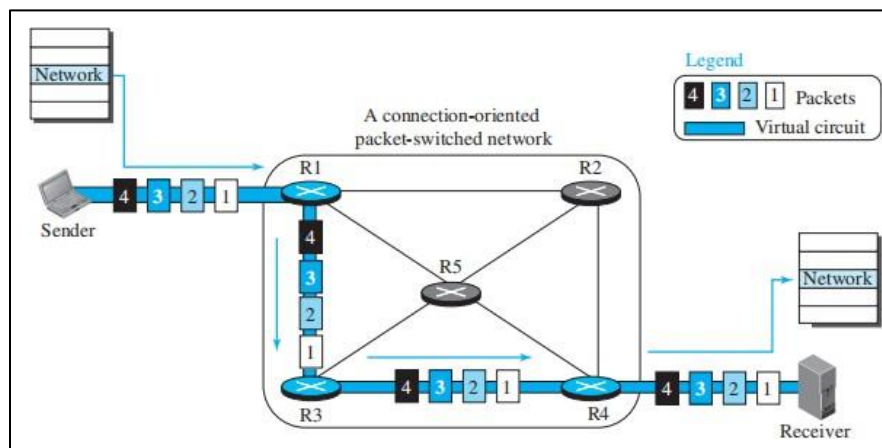
Datagram Approach: Connectionless Service

- ➔ When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message.
- ➔ The switches in this type of network are called **routers**.
- ➔ A packet belonging to a message may be followed by a packet belonging to the same message or to a different message.
- ➔ A packet may be followed by a packet coming from the same or from a different source.
- ➔ Each packet is routed based on the information contained in its header: **source and destination addresses**. The destination address defines where it should go; the source address defines where it comes from. T
- ➔ he router in this case routes the packet based only on the destination address. The source address may be used to send an error message to the source if the packet is discarded.
- ➔ Figure shows the forwarding process in a router in this case. We have used symbolic addresses such as A and B.

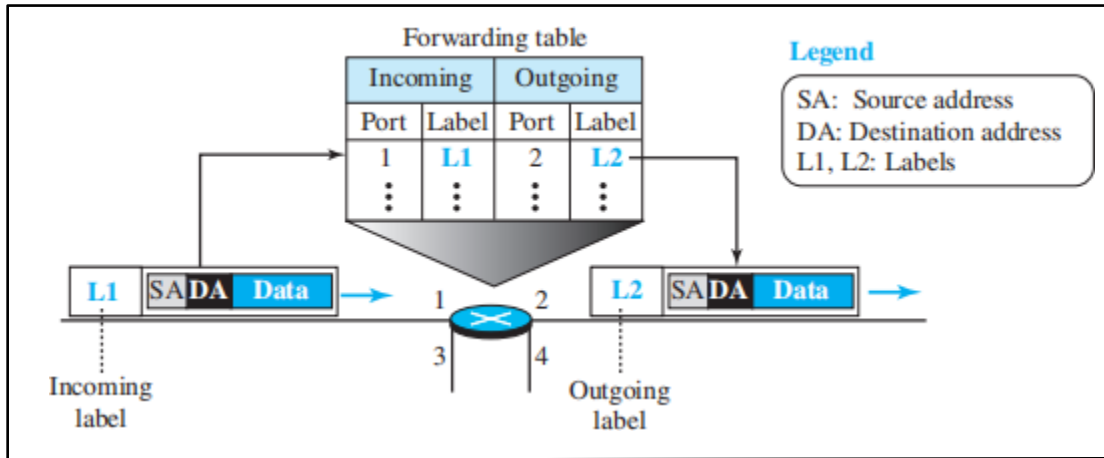


Virtual-Circuit Approach: Connection-Oriented Service

- ➔ In a connection-oriented service (also called virtual-circuit approach), there is a relationship between all packets belonging to a message.
- ➔ Before all datagrams in a message can be sent, a virtual connection should be set up to define the path for the datagrams.
- ➔ After connection setup, the datagrams can all follow the same path.
- ➔ In this type of service, not only must the packet contain the source and destination addresses, it must also contain a **flow label**, a virtual circuit identifier that defines the virtual path the packet should follow.



Each packet is forwarded based on a label in the packet. In a connection-oriented design, we assume the packet has a label when it reaches the router. The router uses this label, also called a virtual circuit identifier, to decide where to send the packet next.



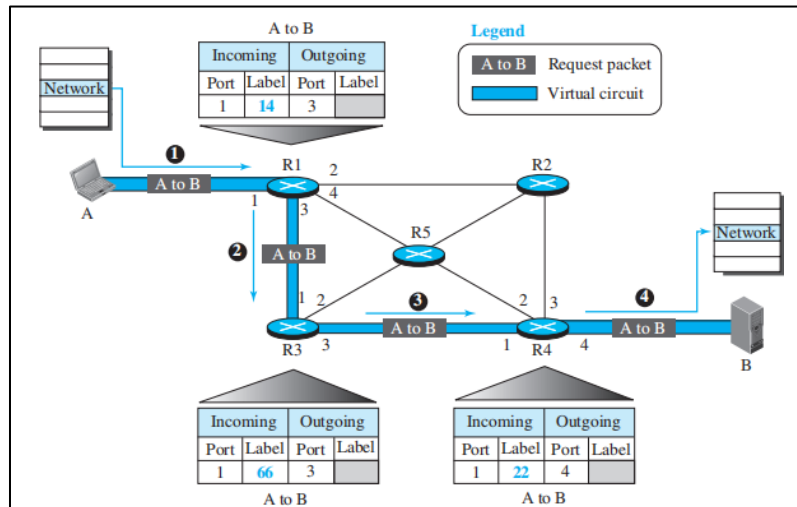
To create a connection-oriented service, a three-phase process is used: setup, data transfer, and teardown.

Setup Phase

In the setup phase, a router creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to destination B. Two auxiliary packets need to be exchanged between the sender and the receiver: **the request packet and the acknowledgment packet**.

Request packet

A request packet is sent from the source to the destination. This auxiliary packet carries the source and destination addresses.



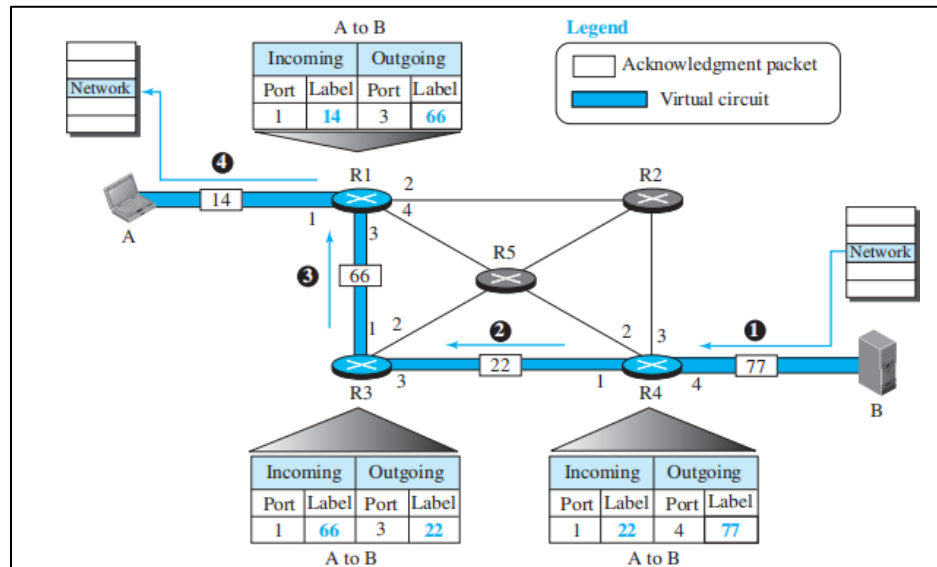
1. Source A sends a request packet to router R1.
2. Router R1 receives the request packet. It knows that a packet going from A to B goes out through port 3. How the router has obtained this information is a point covered later. For the moment, assume that it knows the output port. The router creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The router assigns the incoming port (1) and chooses an available incoming label (14) and the

outgoing port (3). It does not yet know the outgoing label, which will be found during the acknowledgment step. The router then forwards the packet through port 3 to router R3.

- Router R3 receives the setup request packet. The same events happen here as at router R1; three columns of the table are completed: in this case, incoming port (1), incoming label (66), and outgoing port (3).
- Router R4 receives the setup request packet. Again, three columns are completed: incoming port (1), incoming label (22), and outgoing port (4).
- Destination B receives the setup packet, and if it is ready to receive packets from A, it assigns a label to the incoming packets that come from A, in this case 77. This label lets the destination know that the packets come from A, and not from other sources.

Acknowledgment Packet

A special packet, called the acknowledgment packet, completes the entries in the switching tables.



- The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router R4 uses this label to complete the outgoing label column for this entry. Note that 77 is the incoming label for destination B, but the outgoing label for router R4.
- Router R4 sends an acknowledgment to router R3 that contains its incoming label in the table, chosen in the setup phase. Router R3 uses this as the outgoing label in the table.
- Router R3 sends an acknowledgment to router R1 that contains its incoming label in the table, chosen in the setup phase. Router R1 uses this as the outgoing label in the table.

4. Finally, router R1 sends an acknowledgment to source A that contains its incoming label in the table, chosen in the setup phase.
5. The source uses this as the outgoing label for the data packets to be sent to destination B.

Data-Transfer Phase

The second phase is called the data-transfer phase. After all routers have created their forwarding table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another.

Teardown Phase

In the teardown phase, source A, after sending all packets to B, sends a special packet called a teardown packet. Destination B responds with a confirmation packet. All routers delete the corresponding entries from their tables.

IPV4 ADDRESSES

The identifier used in the IP layer of the TCP/IP protocol suite to identify the connection of each device to the Internet is called the Internet address or IP address. An IPv4 address is a 32-bit address that uniquely and universally defines the connection of a host or a router to the Internet. The IP address is the address of the connection, not the host or the router, because if the device is moved to another network, the IP address may be changed.

IPv4 addresses are unique in the sense that each address defines one, and only one, connection to the Internet. If a device has two connections to the Internet, via two networks, it has two IPv4 addresses. IPv4 addresses are universal in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

Address Space

A protocol like IPv4 that defines addresses has an address space. An address space is the total number of addresses used by the protocol. If a protocol uses b bits to define an address, the address space is 2^b because each bit can have two different values (0 or 1).

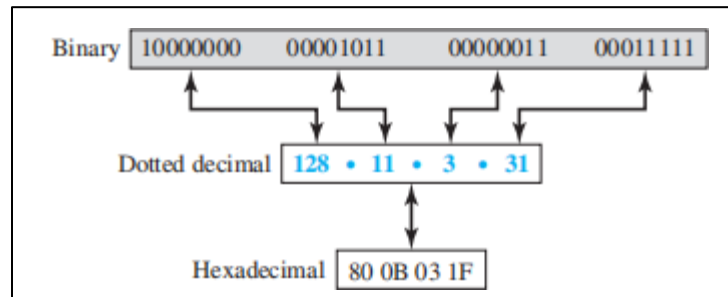
IPv4 uses 32-bit addresses, which means that the address space is 2^{32} or 4,294,967,296 (more than four billion). If there were no restrictions, more than 4 billion devices could be connected to the Internet.

Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16). In binary notation, an IPv4 address is displayed as 32 bits.

To make the address more readable, one or more spaces are usually inserted between each octet (8 bits). Each octet is often referred to as a byte. To make the IPv4 address more compact and easier to read, it is usually written in decimal form with a decimal point (dot) separating the bytes. This format is referred to as dotted-decimal notation. Note that because each byte (octet) is only 8 bits, each number in the dotted-decimal notation is between 0 and 255. We sometimes see

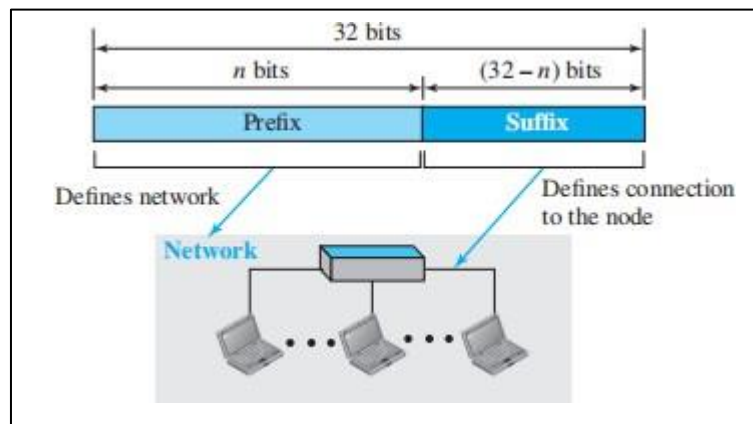
an IPv4 address in hexadecimal notation. Each hexadecimal digit is equivalent to four bits. This means that a 32-bit address has 8 hexadecimal digits. This notation is often used in network programming.



Hierarchy in Addressing

In any communication network that involves delivery, such as a telephone network or a postal network, the addressing system is hierarchical. In a postal network, the postal address (mailing address) includes the country, state, city, street, house number, and the name of the mail recipient. Similarly, a telephone number is divided into the country code, area code, local exchange, and the connection.

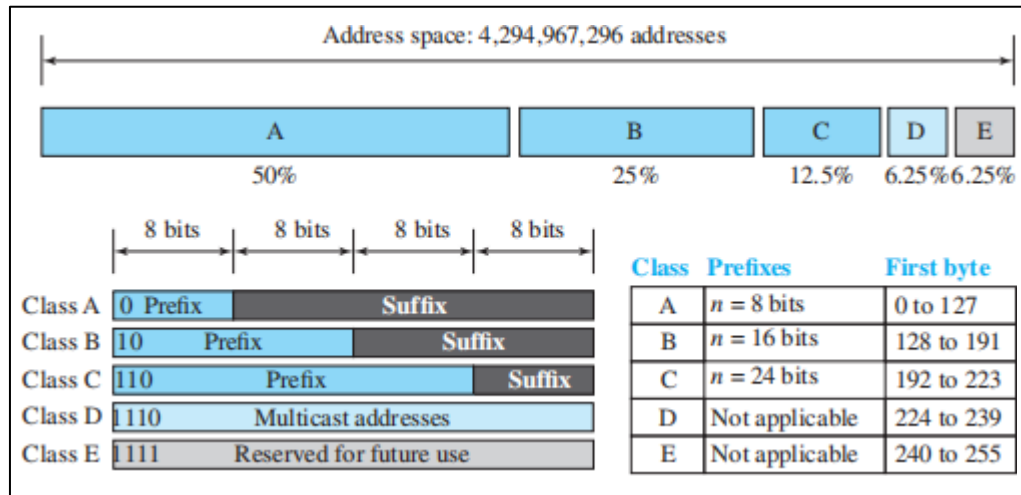
A 32-bit IPv4 address is also hierarchical, but divided only into two parts. The first part of the address, called the *prefix*, defines the network; the second part of the address, called the *suffix*, defines the node (connection of a device to the Internet). Figure shows the prefix and suffix of a 32-bit IPv4 address. The prefix length is n bits and the suffix length is $(32 - n)$ bits.



A prefix can be fixed length or variable length. The network identifier in the IPv4 was first designed as a fixed-length prefix. This scheme, which is now obsolete, is referred to as classful addressing. The new scheme, which is referred to as classless addressing, uses a variable-length network prefix. First, we briefly discuss classful addressing; then we concentrate on classless addressing.

Classful Addressing

- ➔ When the Internet started, an IPv4 address was designed with a fixed-length prefix, but to accommodate both small and large networks, three fixed-length prefixes were designed instead of one ($n = 8$, $n = 16$, and $n = 24$). The whole address space was divided into five classes (class A, B, C, D, and E). This scheme is referred to as **classful addressing**.
- ➔ In class A, the network length is 8 bits, but since the first bit, which is 0, defines the class, we can have only seven bits as the network identifier. This means there are only $2^7 = 128$ networks in the world that can have a class A address.
- ➔ In class B, the network length is 16 bits, but since the first two bits, which are $(10)_2$, define the class, we can have only 14 bits as the network identifier. This means there are only $2^{14} = 16,384$ networks in the world that can have a class B address.
- ➔ All addresses that start with $(110)_2$ belong to class C. In class C, the network length is 24 bits, but since three bits define the class, we can have only 21 bits as the network identifier. This means there are $2^{21} = 2,097,152$ networks in the world that can have a class C address.



Class D is not divided into prefix and suffix. It is used for multicast addresses. All addresses that start with 1111 in binary belong to class E. As in Class D, Class E is not divided into prefix and suffix and is used as reserve.

Address Depletion

Classful addressing became obsolete due to address depletion. The inefficient distribution of addresses led to a rapid shortage. For example, Class A, which could only be assigned to 128 organizations, provided more addresses (16.7 million per network) than most needed, resulting in wasted space. Class B, for midsize organizations, also had many unused addresses. Class C's flaw was offering too few addresses (256 per network), making it impractical for many companies. Class E was rarely used, leading to further waste.

Subnetting and Supernetting

Two strategies were proposed to address depletion: **subnetting** and **supernetting**. Subnetting divides large blocks (Class A or B) into smaller subnets with longer prefixes, but it failed because large organizations didn't want to share unused addresses. Supernetting combined multiple Class C blocks to create larger blocks for organizations needing more than 256 addresses, but this made packet routing more complex and also proved ineffective.

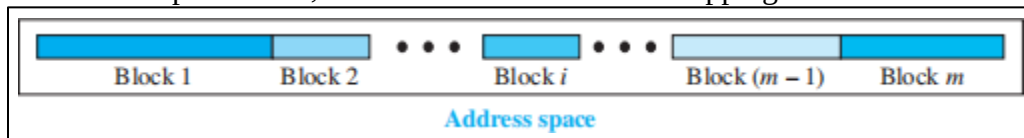
Advantage of Classful Addressing

Although classful addressing had several problems and became obsolete, it had one advantage: Given an address, we can easily find the class of the address and, since the prefix length for each class is fixed, we can find the prefix length immediately.

Classless Addressing

Subnetting and supernetting didn't fully solve address depletion. As the Internet grew, it became clear that a larger address space was needed, leading to the development of **IPv6** as a long-term solution. However, a short-term fix was **classless addressing**, which kept IPv4 but removed class boundaries to distribute addresses more fairly.

ISPs played a role in this change, providing Internet access by subdividing large address ranges and assigning smaller blocks (1, 2, 4, etc.) to customers. **Classless addressing**, introduced in 1996, allows for variable-length blocks of addresses (e.g., 1, 2, 4, or 128). The prefix of the address identifies the network, while the suffix identifies the device. The number of addresses in each block must be a power of 2, and the blocks are non-overlapping.



In classless addressing, the prefix length is variable, ranging from 0 to 32. A shorter prefix indicates a larger network, while a longer prefix means a smaller network. Classless addressing can be applied to classful addressing by treating Class A as having a prefix of 8, Class B with a prefix of 16, and so on.

Prefix Length and Slash Notation (CIDR)

In classless addressing, the prefix length (n) is added to the address, separated by a slash, known as slash notation or CIDR (Classless Inter-Domain Routing). For example, in 167.199.170.82/27, "27" is the prefix length. This notation is necessary since the address alone does not define the block.

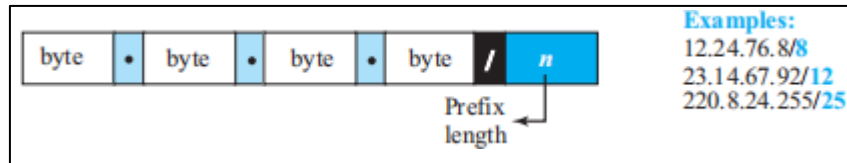
To extract information from a classless address:

Number of addresses = $2^{(32 - n)}$

First address: Keep the first n bits, set the rest to 0.

Last address: Keep the first n bits, set the rest to 1.

For 167.199.170.82/27, there are 32 addresses in the block.



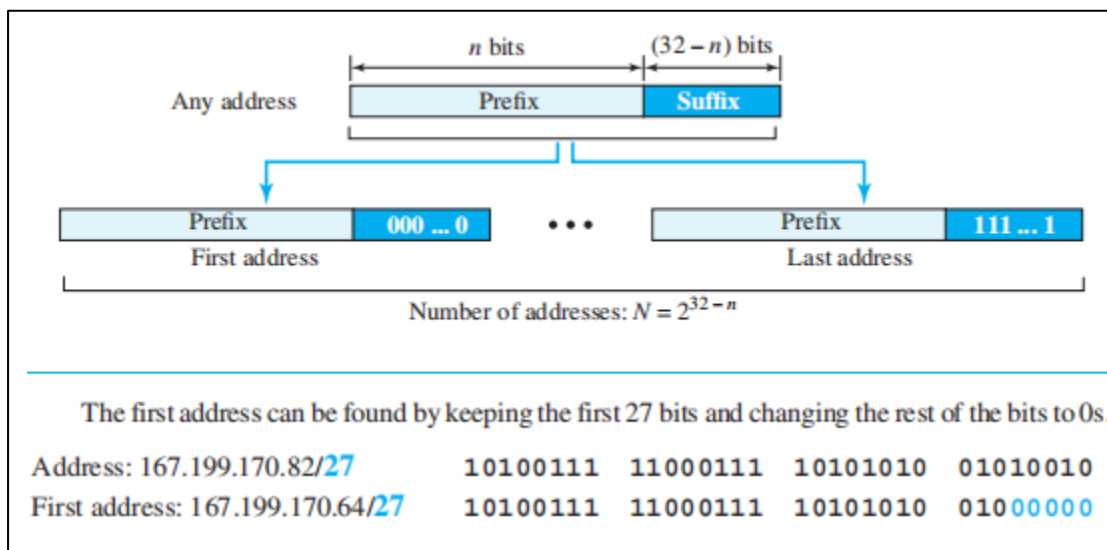
Extracting Information from an Address

Given any address in the block, we normally like to know three pieces of information about the block to which the address belongs: the number of addresses, the first address in the block, and the last address. Since the value of prefix length, n , is given, we can easily find these three pieces of information, as shown in Figure 18.21.

1. The number of addresses in the block is found as $N = 2^{32-n}$.
2. To find the first address, we keep the n leftmost bits and set the $(32 - n)$ rightmost bits all to 0s.
3. To find the last address, we keep the n leftmost bits and set the $(32 - n)$ rightmost bits all to 1s.

Example 1

A classless address is given as 167.199.170.82/27. We can find the above three pieces of information as follows. The number of addresses in the network is $2^{32-n} = 2^5 = 32$ addresses.



Address Mask

Another way to find the first and last addresses in the block is to use the address mask. The address mask is a 32-bit number in which the n leftmost bits are set to 1s and the rest of the bits $(32 - n)$ are set to 0s. A computer can easily find the address mask because it is the complement of $(2^{32-n} - 1)$. The reason for defining a mask in this way is that it can be used by a computer program to extract the information in a block, using the three bit-wise operations NOT, AND, and OR.

1. The number of addresses in the block $N = \text{NOT}(\text{mask}) + 1$.

2. The first address in the block = (Any address in the block) **AND** (mask).
3. The last address in the block = (Any address in the block) **OR** [(**NOT** (mask))].

Example 2

We repeat Example 1 using the mask. The mask in dotted-decimal notation is 256.256.256.224. The AND, OR, and NOT operations can be applied to individual bytes using calculators and applets at the book website.

Number of addresses in the block: $N = \text{NOT}(\text{mask}) + 1 = 0.0.0.31 + 1 = 32 \text{ addresses}$
 First address: First = (address) **AND** (mask) = 167.199.170.82
 Last address: Last = (address) **OR** (**NOT** mask) = 167.199.170.255

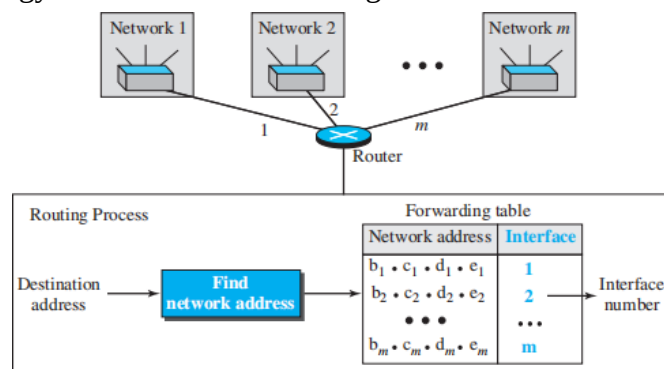
Example 3

In classless addressing, an address cannot per se define the block the address belongs to. For example, the address 230.8.24.56 can belong to many blocks. Some of them are shown below with the value of the prefix associated with that block.

Prefix length: 16	→	Block:	230.8.0.0	to	230.8.255.255
Prefix length: 20	→	Block:	230.8.16.0	to	230.8.31.255
Prefix length: 26	→	Block:	230.8.24.0	to	230.8.24.63
Prefix length: 27	→	Block:	230.8.24.32	to	230.8.24.63
Prefix length: 29	→	Block:	230.8.24.56	to	230.8.24.63
Prefix length: 31	→	Block:	230.8.24.56	to	230.8.24.57

Network Address

The above examples show that, given any address, we can find all information about the block. The first address, **the network address**, is particularly important because it is used in routing a packet to its destination network. For the moment, let us assume that an internet is made of m networks and a router with m interfaces. When a packet arrives at the router from any source host, the router needs to know to which network the packet should be sent: from which interface the packet should be sent out. When the packet arrives at the network, it reaches its destination host using another strategy that we discuss later. Figure shows the idea.



After the network address has been found, the router consults its forwarding table to find the corresponding interface from which the packet should be sent out. The network address is actually the identifier of the network; each network is identified by its network address.

Block Allocation

Block allocation is managed by the Internet Corporation for Assigned Names and Numbers (ICANN), which assigns large blocks of addresses to Internet Service Providers (ISPs) rather than individual users. For CIDR (Classless Inter-Domain Routing) to function correctly, two key restrictions are applied to the allocated block:

1. The number of requested addresses, N , needs to be a power of 2. The reason is that $N = 2^{32-n}$ or $n = 32 - \log_2 N$. If N is not a power of 2, we cannot have an integer value for n .
2. The requested block must be allocated where enough contiguous addresses are available. The first address in the block must be divisible by the total number of addresses. This ensures the first address is the prefix followed by $(32 - n)$ zeros, making it divisible by the block size.

$$\text{first address} = (\text{prefix in decimal}) \times 2^{32-n} = (\text{prefix in decimal}) \times N.$$

Example

An ISP has requested a block of 1000 addresses. Since 1000 is not a power of 2, 1024 addresses are granted. The prefix length is calculated as $n = 32 - \log_2 1024 = 22$. An available block, 18.14.12.0/22, is granted to the ISP. It can be seen that the first address in decimal is 302,910,464, which is divisible by 1024.

Subnetting

More levels of hierarchy can be created using subnetting. An organization (or an ISP) that is granted a range of addresses may divide the range into several subranges and assign each subrange to a subnetwork (or subnet). Note that nothing stops the organization from creating more levels. A subnetwork can be divided into several sub-subnetworks. A sub-subnetwork can be divided into several sub-sub-subnetworks, and so on.

Designing Subnets

The subnetworks in a network should be carefully designed to enable the routing of packets. We assume the total number of addresses granted to the organization is N , the prefix length is n , the assigned number of addresses to each subnetwork is N_{sub} , and the prefix length for each subnetwork is n_{sub} . Then the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

1. The number of addresses in each subnetwork should be a power of 2.
2. The prefix length for each subnetwork should be found using the following formula: first address = (prefix in decimal) $\times 2^{32-n} = (\text{prefix in decimal}) \times N$.

$$n_{\text{sub}} = 32 - \log_2 N_{\text{sub}}$$

- The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger subnetworks.

Finding Information about Each Subnetwork

After designing the subnetworks, the information about each subnetwork, such as first and last address, can be found using the process we described to find the information about each network in the Internet.

Example 18.5

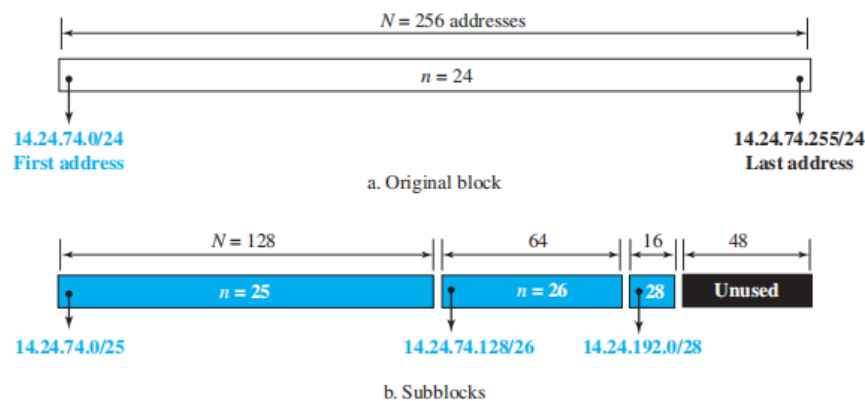
An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets: one subblock of 10 addresses, one subblock of 60 addresses, and one subblock of 120 addresses. Design the subblocks.

Solution

There are $2^{32-24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24. To satisfy the third requirement, we assign addresses to subblocks, starting with the largest and ending with the smallest one.

- The number of addresses in the largest subblock, which requires 120 addresses, is not a power of 2. We allocate 128 addresses. The subnet mask for this subnet can be found as $n_1 = 32 - \log_2 128 = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.
- The number of addresses in the second largest subblock, which requires 60 addresses, is not a power of 2 either. We allocate 64 addresses. The subnet mask for this subnet can be found as $n_2 = 32 - \log_2 64 = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.
- The number of addresses in the smallest subblock, which requires 10 addresses, is not a power of 2 either. We allocate 16 addresses. The subnet mask for this subnet can be found as $n_3 = 32 - \log_2 16 = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.208. The last address is 14.24.74.255. We don't know about the prefix length yet. Figure 18.23 shows the configuration of blocks. We have shown the first address in each block.

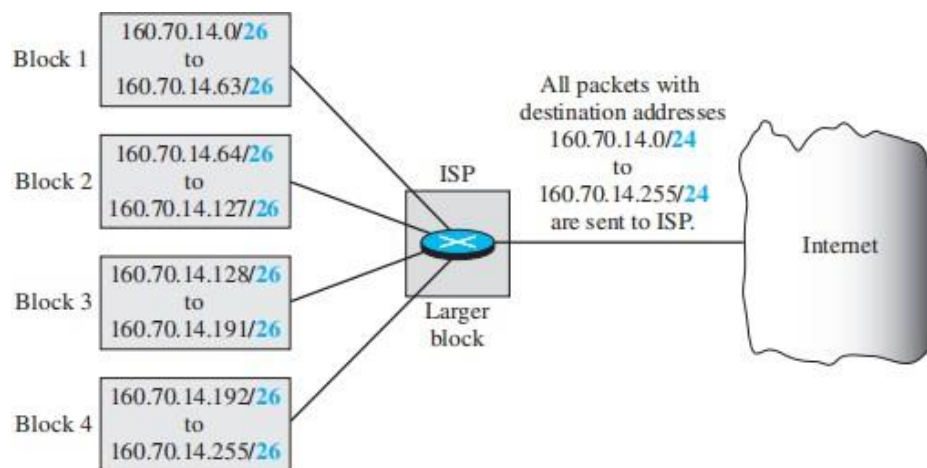


Address Aggregation

One of the advantages of the CIDR strategy is **address aggregation** (sometimes called *address summarization* or *route summarization*). When blocks of addresses are combined to create a larger block, routing can be done based on the prefix of the larger block. ICANN assigns a large block of addresses to an ISP. Each ISP in turn divides its assigned block into smaller subblocks and grants the subblocks to its customers.

Example

Figure 18.24 shows how four small blocks of addresses are assigned to four organizations by an ISP. The ISP combines these four blocks into one single block and advertises the larger block to the rest of the world. Any packet destined for this larger block should be sent to this ISP. It is the responsibility of the ISP to forward the packet to the appropriate organization. This is similar to routing we can find in a postal network. All packages coming from outside a country are sent first to the capital and then distributed to the corresponding destination.



Special Addresses

1. **This-host Address (0.0.0.0/32):** Used when a host doesn't know its own address.
2. **Limited-broadcast Address (255.255.255.255/32):** Used to send a datagram to all devices within a network. Routers block this outside the network.
3. **Loopback Address (127.0.0.0/8):** Used for testing software on the same host; packets never leave the host.
4. **Private Addresses:** Reserved for internal use (e.g., 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, 169.254.0.0/16); used with NAT.
5. **Multicast Addresses (224.0.0.0/4):** Reserved for multicast communication.

Dynamic Host Configuration Protocol (DHCP)

DHCP is an application-layer protocol that automates IP address assignment, using a client-server model. Widely used on the Internet, it is often referred to as a "plug-and-play" protocol.

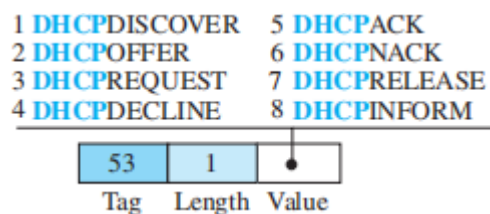
- **Address Assignment:** DHCP can assign permanent or temporary IP addresses. For example, ISPs can use DHCP to provide temporary addresses to users, allowing limited IP resources to serve more customers.

- **Additional Information:** DHCP can also provide essential details like the network prefix, default router address, and name server address.

DHCP Message Format

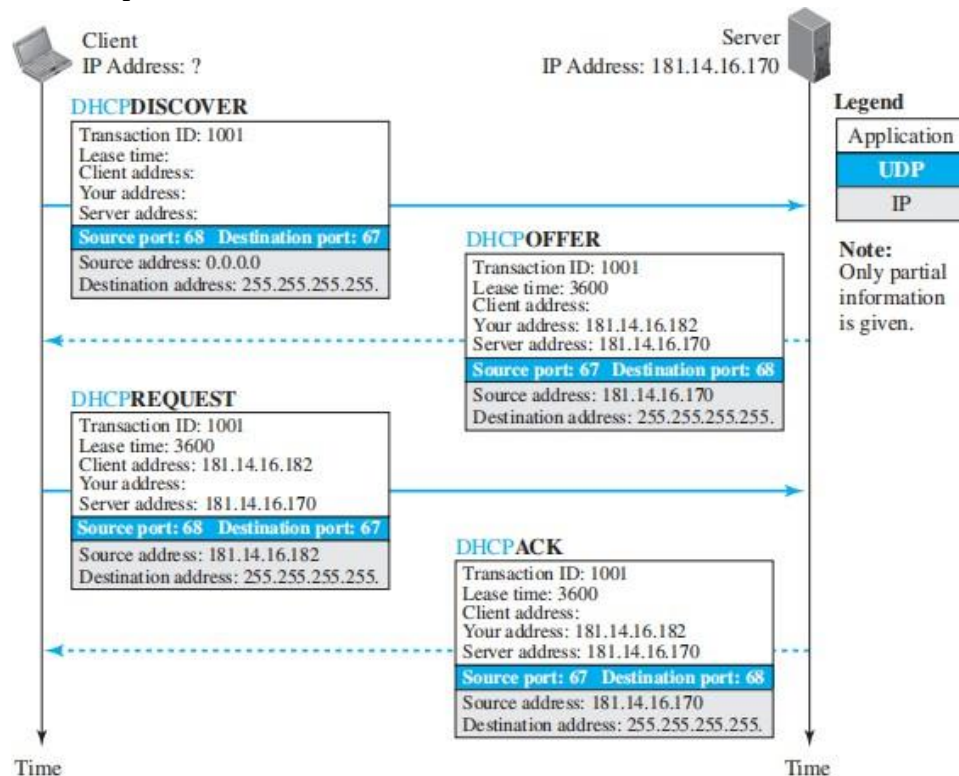
0	8	16	24	31	
Opcode	Htype	HLen	HCount		Fields:
Transaction ID					Opcode: Operation code, request (1) or reply (2)
Time elapsed		Flags			Htype: Hardware type (Ethernet, ...)
Client IP address					HLen: Length of hardware address
Your IP address					HCount: Maximum number of hops the packet can travel
Server IP address					Transaction ID: An integer set by the client and repeated by the server
Gateway IP address					Time elapsed: The number of seconds since the client started to boot
Client hardware address					Flags: First bit defines unicast (0) or multicast (1); other 15 bits not used
Server name					Client IP address: Set to 0 if the client does not know it
Boot file name					Your IP address: The client IP address sent by the server
Options					Server IP address: A broadcast IP address if client does not know it
					Gateway IP address: The address of default router
					Server name: A 64-byte domain name of the server
					Boot file name: A 128-byte file name holding extra information
					Options: A 64-byte field with dual purpose described in text

The 64-byte option field in DHCP serves two purposes: carrying additional or vendor-specific information. A special value, called a "magic cookie" (99.130.83.99), helps the client recognize options in the message. The next 60 bytes contain options, structured in three fields: a 1-byte tag, 1-byte length, and variable-length value. The tag field (e.g., 53) can indicate one of the 8 DHCP message types used by the protocol.



DHCP Operation

Figure shows a simple scenario.



1. The host creates a **DHCPDISCOVER** message with only a random transaction-ID, as it doesn't know its own IP address or the server's. The message is sent using UDP (source port 68, destination port 67) and broadcasted (source IP: 0.0.0.0, destination IP: 255.255.255.255).
2. The DHCP server responds with a **DHCPOFFER** message, containing the offered IP address, server address, and lease time. This message is sent with the same port numbers but reversed, using a broadcast address so other servers can also offer better options.
3. The host selects the best offer and sends a **DHCPREQUEST** to the server. The message includes the chosen IP address and is sent as a broadcast (source: new client IP, destination: 255.255.255.255) to notify other servers that their offers were declined.
4. The selected server responds with a **DHCPACK** if the IP address is valid, completing the process. If the IP address is unavailable, a **DHCPNACK** is sent, and the host must restart the process.

Two Well-Known Ports

DHCP uses well-known ports (68 for the client, 67 for the server) to avoid conflicts with other services using ephemeral ports. Since DHCP responses are broadcast, using a well-

known port (68) ensures the DHCP response is only delivered to the correct client, preventing confusion with other clients (e.g., DAYTIME) that might be using the same temporary port. If two DHCP clients are running simultaneously (e.g., after a power failure), they are distinguished by their unique transaction IDs.

Using FTP

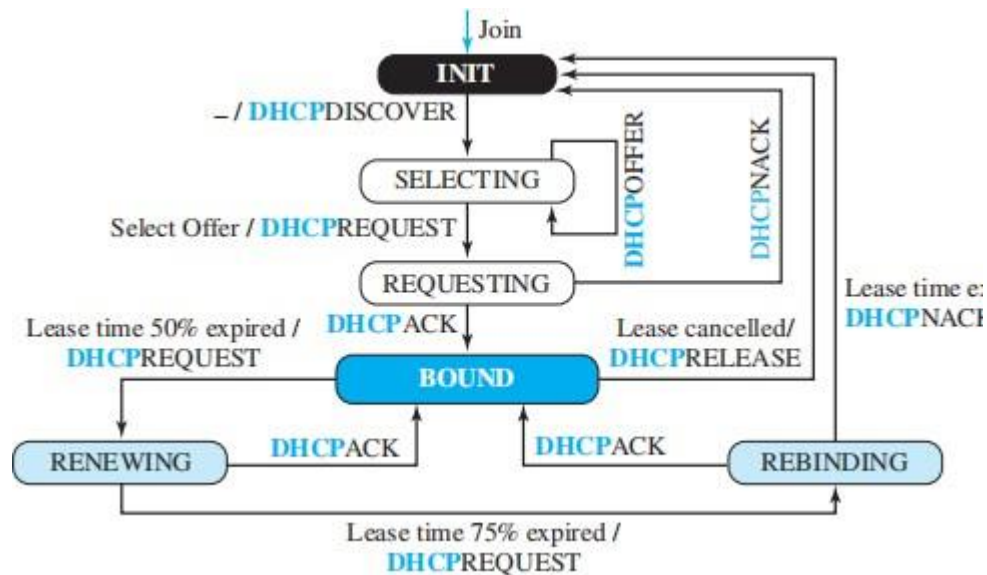
The DHCPACK message includes a pathname to a file with additional information (e.g., DNS server address). The client uses FTP to retrieve this information.

Error Control

Since DHCP relies on unreliable UDP, it ensures error control by requiring UDP checksums and using timers with a retransmission policy. To avoid traffic congestion (e.g., after a power failure), clients use random timers for retransmission.

Transition States

The operation of the DHCP were very simple. To provide dynamic address allocation, the DHCP client acts as a state machine that performs transitions from one state to another depending on the messages it receives or sends.



1. **INIT state:** The client starts here and sends a *Discover* message to find a DHCP server.
2. **SELECTING state:** After receiving one or more *Offer* messages, the client selects one offer.
3. **REQUESTING state:** The client sends a *Request* message to the selected server and waits.

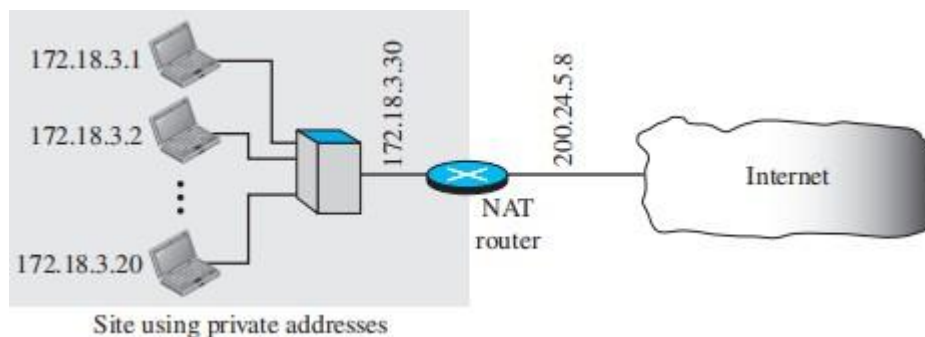
4. **BOUND state:** If the server responds with an *ACK* message, the client uses the assigned IP address.
5. **RENEWING state:** When 50% of the lease time is expired, the client tries to renew the lease by contacting the server. If successful, it stays in the BOUND state.
6. **REBINDING state:** If the lease is 75% expired and no response is received, the client tries to contact any DHCP server. If the server responds, it stays BOUND; otherwise, it goes back to INIT to request a new IP.

Timers:

- **Renewal timer:** Triggered at 50% lease expiration.
- **Rebinding timer:** Triggered at 75% lease expiration.
- **Expiration timer:** Triggered when the full lease expires.

Network Address Resolution (NAT)

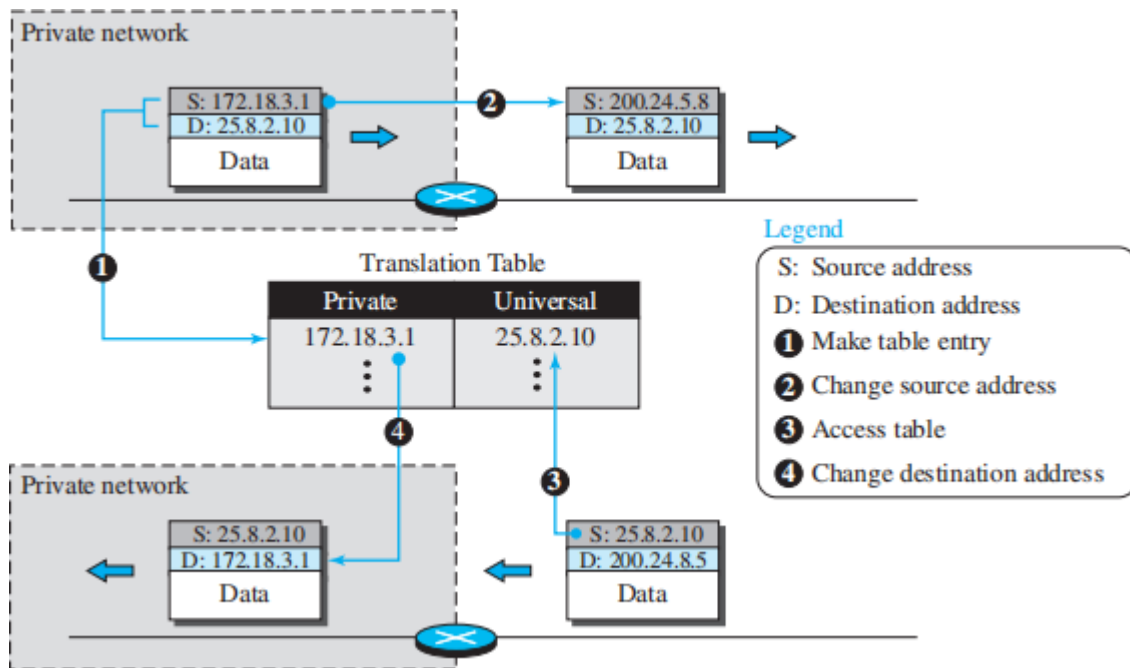
ISPs allocate limited IP address ranges to small businesses or households, but expanding these ranges can be difficult due to neighboring allocations. Often, not all devices in a network need simultaneous Internet access. For example, a small business with 20 computers may only need 4 to access the Internet at once, while the rest handle internal tasks. Using private IP addresses for internal communication and a few global IP addresses from the ISP for external communication solves this. Network Address Translation (NAT) allows a network to use private addresses internally and maps them to global addresses for Internet access via a NAT-enabled router.



The private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is invisible to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 200.24.5.8.

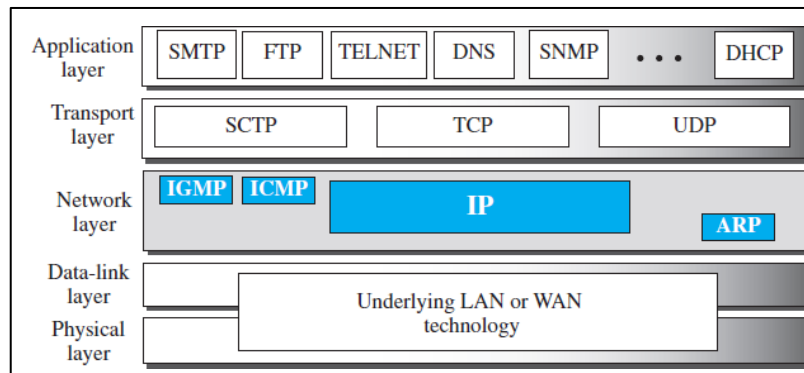
- ➔ In Network Address Translation (NAT), all outgoing packets pass through a NAT router, which replaces the source address with the global NAT address.
- ➔ Incoming packets also pass through the router, where the destination (global) address is replaced with the corresponding private address. To manage this, the NAT router uses a translation table that links private addresses to external addresses.
- ➔ When an outgoing packet is sent, the router records the destination address.

- ➔ When a response arrives, the router uses the source (external) address to find the corresponding private address, allowing proper routing within the local network.



INTERNET PROTOCOL (IP)

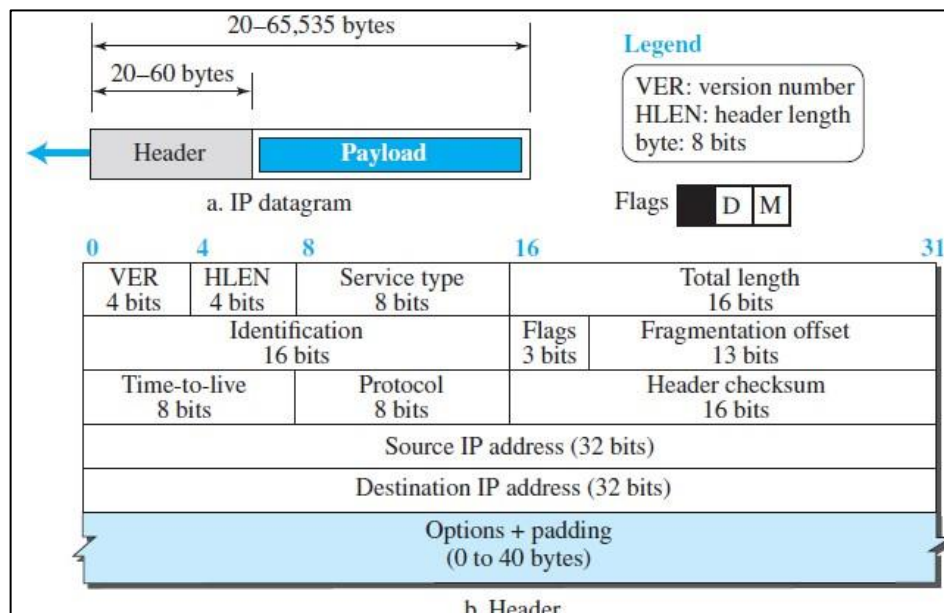
- ➔ The network layer in version 4 can be thought of as one main protocol and three auxiliary ones.
- ➔ The main protocol, Internet Protocol version 4 (IPv4), is responsible for packetizing, forwarding, and delivery of a packet at the network layer.
- ➔ The Internet Control Message Protocol version 4 (ICMPv4) helps IPv4 to handle some errors that may occur in the network-layer delivery. The Internet Group Management Protocol (IGMP) is used to help IPv4 in multicasting.
- ➔ The Address Resolution Protocol (ARP) is used to glue the network and data-link layers in mapping network-layer addresses to link-layer addresses. Figure shows the positions of these four protocols in the TCP/IP protocol suite.



- ➔ IPv4 is an unreliable datagram protocol—a best-effort delivery service. The term *best-effort* means that IPv4 packets can be corrupted, be lost, arrive out of order, or be delayed, and may create congestion for the network.
- ➔ IPv4 is also a connectionless protocol that uses the datagram approach. This means that each datagram is handled independently, and each datagram can follow a different route

Datagram Format

A datagram is a variable-length packet consisting of two parts: **header and payload (data)**. The header is 20 to 60 bytes in length and contains information essential to routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections.

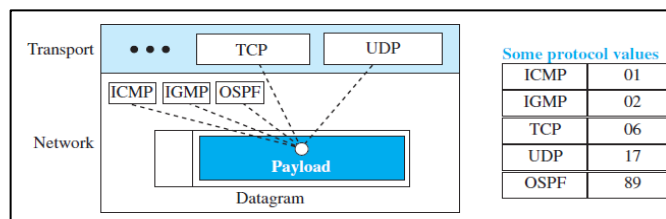


1. **Version Number.** The 4-bit version number (VER) field defines the version of the IPv4 protocol, which, obviously, has the value of 4.
2. **Header Length.** The 4-bit header length (HLEN) field defines the total length of the datagram header in 4-byte words. The IPv4 datagram has a variable-length header.
3. **Service Type.** In the original design of the IP header, this field was referred to as type of service (TOS), which defined how the datagram should be handled.

4. **Total Length.** This 16-bit field defines the total length (header plus data) of the IP datagram in bytes. A 16-bit number can define a total length of up to 65,535 (when all bits are 1s).

$$\text{Length of data} = \text{total length} - (\text{HLEN}) \times 4$$

5. **Identification, Flags, and Fragmentation Offset.** These three fields are related to the fragmentation of the IP datagram when the size of the datagram is larger than the underlying network can carry.
6. **Time-to-live.** The time-to-live (TTL) field is used to control the maximum number of hops (routers) visited by the datagram.
7. **Protocol.** In TCP/IP, the data section of a packet, called the *payload*, carries the whole packet from another protocol. This field provides multiplexing at the source and demultiplexing at the destination, as shown in Figure .



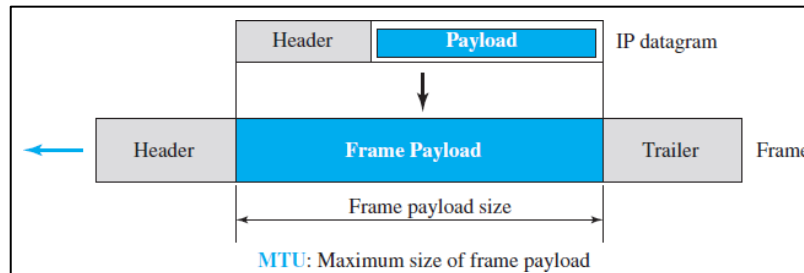
8. **Header checksum.** Errors in the IP header can be a disaster. IP adds a header checksum field to check the header, but not the payload.
9. **Source and Destination Addresses.** These 32-bit source and destination address fields define the IP address of the source and destination respectively.
10. **Options.** A datagram header can have up to 40 bytes of options. Options can be used for network testing and debugging.
11. **Payload.** Payload, or data, is the main reason for creating a datagram. Payload is the packet coming from other protocols that use the service of IP.

Fragmentation

- ➔ A datagram can travel through different networks.
- ➔ Each router decapsulates the IP datagram from the frame it receives, processes it, and then encapsulates it in another frame.
- ➔ The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel.
- ➔ For example, if a router connects a LAN to a WAN, it receives a frame in the LAN format and sends a frame in the WAN format.

Maximum Transfer Unit (MTU)

- ➔ Each link-layer protocol has its own frame format.
- ➔ One of the features of each format is the maximum size of the payload that can be encapsulated.



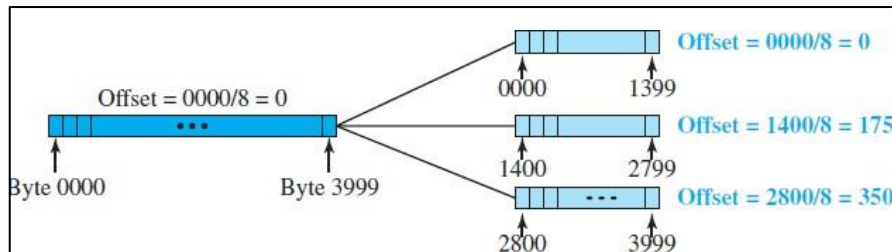
- ➔ The value of the MTU differs from one physical network protocol to another.
- ➔ In order to make the IP protocol independent of the physical network, the designers decided to make the maximum length of the IP datagram equal to 65,535 bytes..
- ➔ For other physical networks, we must divide the datagram to make it possible for it to pass through these networks. This is called **fragmentation**.
- ➔ When a datagram is fragmented, each fragment has its own header with most of the fields repeated, but some have been changed.
- ➔ A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU.
- ➔ The *reassembly* of the datagram, is done only by the destination host, because each fragment becomes an independent datagram.
- ➔ In fragmentation, the payload of the IP datagram is fragmented. most parts of the header, with the exception of some options, must be copied by all fragments.
- ➔ The host or router that fragments a datagram must change the values of three fields: flags, fragmentation offset, and total length.
- ➔ The value of the checksum must be recalculated regardless of fragmentation.

Fields Related to Fragmentation

Three fields in an IP datagram are related to fragmentation:

1. **Identification**-The 16-bit *identification field* identifies a datagram originating from the source host. The combination of the identification and source IP address must uniquely define a datagram as it leaves the source host. To guarantee uniqueness, the IP protocol uses a counter to label the datagrams. The identification number helps the destination in reassembling the datagram.
2. **Flags**- The 3-bit *flags field* defines three flags.
 - ➔ The leftmost bit is reserved (not used).
 - ➔ The second bit (D bit) is called the *do not fragment* bit. If its value is 1, the machine must not fragment the datagram. If its value is 0, the datagram can be fragmented if necessary.
 - ➔ The third bit (M bit) is called the *more fragment bit*. If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment.
3. **fragmentation offset** The 13-bit *fragmentation offset field* shows the relative position of this fragment with respect to the whole datagram. It is the offset of the data in the original datagram measured in units of 8 bytes.

Figure shows a datagram with a data size of 4000 bytes fragmented into three fragments. The bytes in the original datagram are numbered 0 to 3999. The first fragment carries bytes 0 to 1399. The offset for this datagram is $0/8 = 0$. The second fragment carries bytes 1400 to 2799; the offset value for this fragment is $1400/8 = 175$. Finally, the third fragment carries bytes 2800 to 3999. The offset value for this fragment is $2800/8 = 350$.



Options

- ➔ The header of the IPv4 datagram is made of two parts: a fixed part and a variable part.
 - ➔ The fixed part is 20 bytes long.
 - ➔ The variable part comprises the options that can be a maximum of 40 bytes (in multiples of 4-bytes) to preserve the boundary of the header.
 - ➔ Options, are not required for a datagram. They can be used for network testing and debugging.
 - ➔ Options are divided into two broad categories: single-byte options and multiple-byte options.
1. **Single-Byte Options** There are two single-byte options.
 - I. **No Operation** A *no-operation option* is a 1-byte option used as a filler between options.
 - II. **End of Option** An *end-of-option option* is a 1-byte option used for padding at the end of the option field. It, however, can only be used as the last option.
 2. **Multiple-Byte Options** There are four multiple-byte options.
 - I. **Record Route** A *record route option* is used to record the Internet routers that handle the datagram.
 - II. **Strict Source Route** A *strict source route option* is used by the source to predetermine a route for the datagram as it travels through the Internet.
 - III. **Loose Source Route** A *loose source route option* is similar to the strict source route, but it is less rigid. Each router in the list must be visited, but the datagram can visit other routers as well.
 - IV. **Timestamp** A *timestamp option* is used to record the time of datagram processing by a router.

Security of IPv4 Datagrams

There are three security issues that are particularly applicable to the IP protocol

1. Packet Sniffing

- ➔ An intruder may intercept an IP packet and make a copy of it. Packet sniffing is a passive attack, in which the attacker does not change the contents of the packet.
- ➔ This type of attack is very difficult to detect because the sender and the receiver may never know that the packet has been copied.
- ➔ Although packet sniffing cannot be stopped, encryption of the packet can make the attacker's effort useless.
- ➔ The attacker may still sniff the packet, but the content is not detectable.

2. **Packet Modification**

- ➔ The attacker intercepts the packet, changes its contents, and sends the new packet to the receiver.
- ➔ The receiver believes that the packet is coming from the original sender. This type of attack can be detected using a data integrity mechanism.
- ➔ The receiver, before opening and using the contents of the message, can use this mechanism to make sure that the packet has not been changed during the transmission.

3. **IP Spoofing**

- ➔ An attacker can masquerade as somebody else and create an IP packet that carries the source address of another computer.
- ➔ An attacker can send an IP packet to a bank pretending that it is coming from one of the customers. T
- ➔ This type of attack can be prevented using an origin authentication mechanism.

IPSec

The IP packets today can be protected from the previously mentioned attacks using a protocol called IPSec (IP Security). IPSec provides the following four services:

1. **Defining Algorithms and Keys.** The two entities that want to create a secure channel between them can agree on some available algorithms and keys to be used for security purposes.
2. **Packet Encryption.** The packets exchanged between two parties can be encrypted for privacy using one of the encryption algorithms and a shared key agreed upon in the first step.
3. **Data Integrity.** Data integrity guarantees that the packet is not modified during the transmission. If the received packet does not pass the data integrity test, it is discarded.
4. **Origin Authentication.** IPSec can authenticate the origin of the packet to be sure that the packet is not created by an imposter.

IPv6 ADDRESSING

The main reason for migration from IPv4 to IPv6 is the small size of the address space in IPv4. An IPv6 address is 128 bits or 16 bytes (octets) long, four times the address length in IPv4.

Representation

A computer normally stores the address in binary, but it is clear that 128 bits cannot easily be handled by humans. There are two of these notations: binary and colon hexadecimal.

Binary (128 bits)	1111111011110110 ... 1111111100000000
Colon Hexadecimal	FEF6:BA98:7654:3210:ADEF:BBFF:2922:FF00

Binary notation is used when the addresses are stored in a computer. The **colon hexadecimal notation** (or *colon hex* for short) divides the address into eight sections, each made of four hexadecimal digits separated by colons.

Abbreviation

Although an IPv6 address, even in hexadecimal format, is very long, many of the digits are zeros. In this case, we can abbreviate the address. The leading zeros of a section can be omitted. Further abbreviation, often called **zero compression**, can be applied to colon hex notation if there are consecutive sections consisting of zeros only. We can remove all the zeros and replace them with a double semicolon.

FDEC:0:0:0:0:BBFF:0:FFFF → **FDEC::BBFF:0:FFFF**

Note that this type of abbreviation is allowed only once per address. If there is more than one run of zero sections, only one of them can be compressed.

Mixed Notation

Sometimes we see a mixed representation of an IPv6 address: colon hex and dotted decimal notation. This is appropriate during the transition period in which an IPv4 address is embedded in an IPv6 address (as the rightmost 32 bits).

CIDR Notation

IPv6 uses hierarchical addressing. For this reason, IPv6 allows slash or CIDR notation. For example, the following shows how we can define a prefix of 60 bits using CIDR. We will later show how an IPv6 address is divided into a prefix and a suffix.

FDEC::BBFF:0:FFFF/60

Address Space

The address space of IPv6 contains 2128 addresses. This address space is 296 times the IPv4 address—definitely no address depletion—as shown, the size of the space is

340, 282, 366, 920, 938, 463, 374, 607, 431, 768, 211, 456.

Three Address Types

In IPv6, a destination address can belong to one of three categories: unicast, anycast, and multicast.

Unicast Address

A unicast address defines a single interface (computer or router). The packet sent to a unicast address will be routed to the intended recipient.

Anycast Address

An **anycast address** defines a group of computers that all share a single address. A packet with an anycast address is delivered to only one member of the group, the most reachable one.

Multicast Address

A multicast address also defines a group of computers. Each member of the group receives a copy. IPv6 considers broadcasting as a special case of multicasting.

Address Space Allocation

Like the address space of IPv4, the address space of IPv6 is divided into several blocks of varying size and each block is allocated for a special purpose. Most of the blocks are still unassigned and have been set aside for future use.

THE IPv6 PROTOCOL

The change of the IPv6 address size requires the change in the IPv4 packet format. The following shows other changes implemented in the protocol in addition to changing address size and format.

Better header format. IPv6 uses a new header format in which options are separated from the base header and inserted, when needed, between the base header and the data.

New options. IPv6 has new options to allow for additional functionalities.

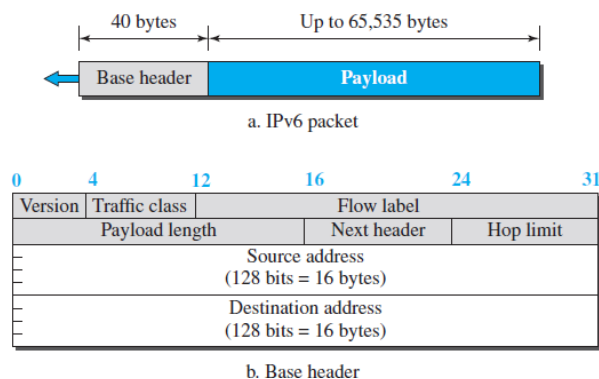
Allowance for extension. IPv6 is designed to allow the extension of the protocol if required by new technologies or applications.

Support for resource allocation. In IPv6, the type-of-service field has been removed, but two new fields, traffic class and flow label, have been added to enable the source to request special handling of the packet. This mechanism can be used to support traffic such as real-time audio and video.

Support for more security. The encryption and authentication options in IPv6 provide confidentiality and integrity of the packet.

Packet Format

The IPv6 packet is shown in Figure. Each packet is composed of a base header followed by the payload. The base header occupies 40 bytes, whereas payload can be up to 65,535 bytes of information. The description of fields follows.



Version. The 4-bit version field defines the version number of the IP. For IPv6, the value is 6.

Traffic class. The 8-bit traffic class field is used to distinguish different payloads with different delivery requirements. It replaces the *type-of-service* field in IPv4.

Flow label. The flow label is a 20-bit field that is designed to provide special handling for a particular flow of data.

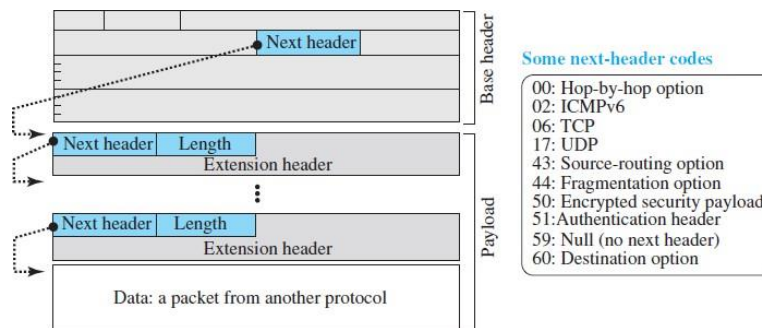
Payload length. The 2-byte payload length field defines the length of the IP datagram excluding the header.

Next header. The **next header** is an 8-bit field defining the type of the first extension header (if present) or the type of the data that follows the base header in the datagram..

Hop limit. The 8-bit hop limit field serves the same purpose as the TTL field in IPv4.

Source and destination addresses. The source address field is a 16-byte (128-bit) Internet address that identifies the original source of the datagram. The destination address field is a 16-byte (128-bit) Internet address that identifies the destination of the datagram.

Payload. Compared to IPv4, the payload field in IPv6 has a different format and meaning, as shown in Figure.



- ➔ The payload in IPv6 means a combination of zero or more extension headers (options) followed by the data from other protocols (UDP, TCP, and so on).
- ➔ In IPv6, options, which are part of the header in IPv4, are designed as extension headers. The payload can have as many extension headers as required by the situation.
- ➔ Each extension header has two mandatory fields, next header and the length, followed by information related to the particular option.
- ➔ Note that each next header field value (code) defines the type of the next header (hop-by-hop option, sourcerouting option, . . .); the last next header field defines the protocol (UDP, TCP, . . .) that is carried by the datagram.

Concept of Flow and Priority in IPv6

- ➔ In version 6, the flow label has been added to the format of the IPv6 datagram to allow us to use IPv6 as a connection-oriented protocol.
- ➔ To a router, a flow is a sequence of packets that share the same characteristics, such as traveling the same path, using the same resources, having the same kind of security, and so on.
- ➔ A router that supports the handling of flow labels has a flow label table. The table has an entry for each active flow label; each entry defines the services required by the corresponding flow label.
- ➔ In its simplest form, a flow label can be used to speed up the processing of a packet by a router.

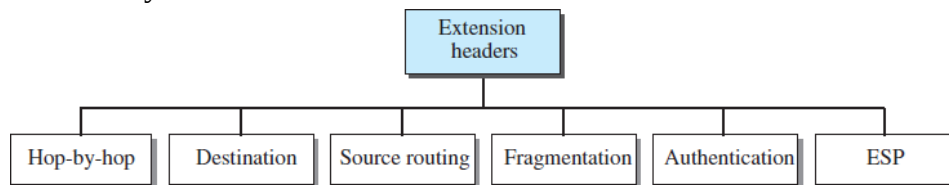
Fragmentation and Reassembly

- ➔ IPv6 datagrams can be fragmented only by the source, not by the routers; the reassembly takes place at the destination.

- ➔ The fragmentation of a packet in a router needs a lot of processing hence processing at router is not allowed.
- ➔ In IPv6, the source can check the size of the packet and make the decision to fragment the packet or not.
- ➔ When a router receives the packet, it can check the size of the packet and drop it if the size is larger than allowed by the MTU of the network ahead.
- ➔ The router then sends a packet-too-big ICMPv6 error message to inform the source.

Extension Header

An IPv6 packet is made of a base header and some extension headers. The length of the base header is fixed at 40 bytes. The base header can be followed by up to six **extension headers**, to give more functionality.



Hop-by-Hop Option

The *hop-by-hop option* is used when the source needs to pass information to all routers visited by the datagram. So far, only three hop by- hop options have been defined: Pad1, PadN, and jumbo payload.

Pad1. This option is 1 byte long and is designed for alignment purposes. Some options need to start at a specific bit of the 32-bit word. If an option falls short of this requirement by exactly one byte, Pad1 is added.

PadN. PadN is similar in concept to Pad1. The difference is that PadN is used when 2 or more bytes are needed for alignment.

Jumbo payload. If for any reason a longer payload(>65,535 bytes) is required, the jumbo payload option is used to define this longer length.

Destination Option

The **destination option** is used when the source needs to pass information to the destination only. Intermediate routers are not permitted access to this information.

Source Routing

The source routing extension header combines the concepts of the strict source route and the loose source route options of IPv4.

Fragmentation

In IPv6, only the original source can fragment. A source must use a **Path MTU Discovery technique** to find the smallest MTU supported by any network on the path. The source then fragments using this knowledge.

Authentication

The **authentication** extension header has a dual purpose: it validates the message sender and ensures the integrity of data.

Encrypted Security Payload

The **encrypted security payload (ESP)** is an extension that provides confidentiality and guards against eavesdropping.

Comparison of Options between IPv4 and IPv6

The following shows a quick comparison between the options used in IPv4 and the options used in IPv6 (as extension headers).

- ➔ The no-operation and end-of-option options in IPv4 are replaced by Pad1 and PadN options in IPv6.
- ➔ The record route option is not implemented in IPv6 because it was not used.
- ➔ The timestamp option is not implemented because it was not used.
- ➔ The source route option is called the *source route extension header* in IPv6.
- ➔ The fragmentation fields in the base header section of IPv4 have moved to the fragmentation extension header in IPv6.
- ➔ The authentication extension header is new in IPv6.
- ➔ The encrypted security payload extension header is new in IPv6

Introduction to Routing Algorithms

Routing algorithms are essential in computer networks to determine the most efficient path for data packets to travel from a source to a destination. These algorithms help manage network traffic, minimize delays, and ensure optimal usage of network resources.

If a datagram is destined for only one destination (one-to-one delivery), we have *unicast routing*. If the datagram is destined for several destinations (one-to-many delivery), we have *multicast routing*.

Unicast Routing Algorithm

Introduction

Unicast routing in the Internet, with a large number of routers and a huge number of hosts, can be done only by using hierarchical routing: routing in several steps using different routing algorithms.

General Idea

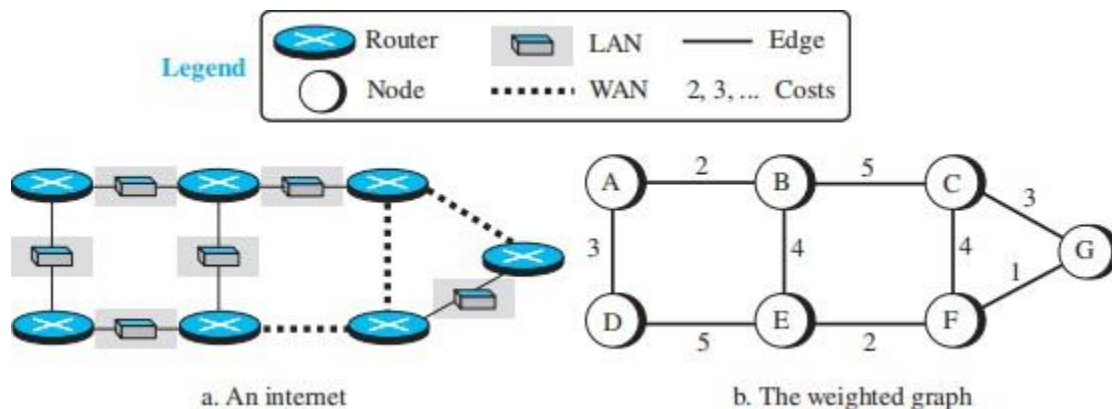
- ➔ In unicast routing, a packet is routed, hop by hop, from its source to its destination by the help of forwarding tables.
- ➔ The source host and the destination host need no forwarding table either because they send and receive the packet from its default router in its local network.
- ➔ The only the routers that glue together the networks in the internet need forwarding tables.
- ➔ Routing a packet from its source to its destination means routing the packet from a *source router* (the default router of the source host) to a *destination router* (the router connected to the destination network).
- ➔ Although a packet needs to visit the source and the destination routers, the packet should visit intermediate routers. There are several routes that a packet can travel from the source to the destination; it must be determined is which route the packet should take.

An Internet as a Graph

- ➔ To find the best route, an internet can be modelled as a graph. A graph in computer science is a set of nodes and edges (lines) that connect the nodes.
- ➔ To model an internet as a graph, we can think of each router as a node and each network between a pair of routers as an edge.
- ➔ An internet is, in fact, modelled as a weighted graph, in which each edge is associated with a cost.
- ➔ If a weighted graph is used to represent a geographical area, the nodes can be cities and the edges can be roads connecting the cities; the weights, in this case, are distances between cities.
- ➔ In routing, however, the cost of an edge has a different interpretation in different routing protocols.

Least-Cost Routing

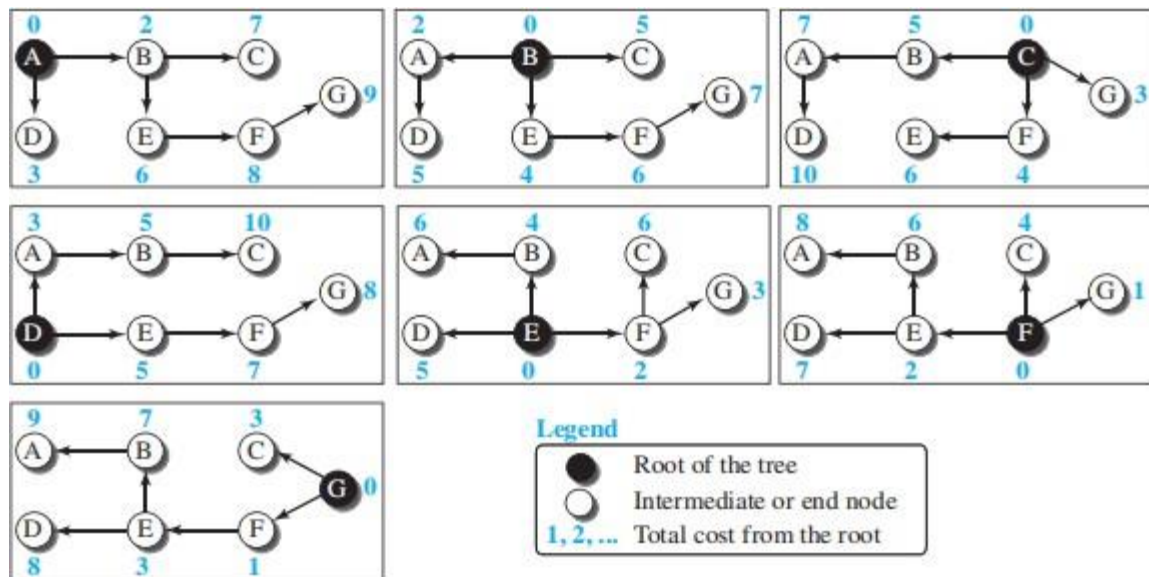
- ➔ When an internet is modelled as a weighted graph, one of the ways to interpret the **best route** from the source router to the destination router is to find the *least cost* between the two.
- ➔ In Figure below, the best route between A and E is A-B-E, with the cost of 6. This means that each router needs to find the least-cost route between itself and all the other routers to be able to route a packet using this criterion.



Least-Cost Trees

- ➔ If there are N routers in a network, each router needs to find the cheapest (least-cost) path to every other router. For example, with 10 routers, you would need to calculate 90 different paths to connect all routers.
- ➔ Instead of calculating all the paths separately, we can group them into something called a least-cost tree (also known as a shortest-path tree).
- ➔ A least-cost tree starts from one router (the source) and connects to all the other routers using the cheapest path for each connection. This way, you can use just one tree to find the best routes from that source router to every other router.

- ➔ Instead of calculating $N \times (N - 1)$ paths (which is 90 paths for 10 routers), you only need to calculate N trees (one for each router). So, with 10 routers, you only need 10 trees instead of 90 paths.



ROUTING ALGORITHMS

Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node.

The Distance-Vector (DV) Routing Algorithm

- ➔ The distance vector (DV) algorithm is iterative, asynchronous, and distributed.
- ➔ It is iterative that the process continues on until no more information is exchanged between neighbours.
- ➔ The algorithm is asynchronous in that it does not require all of the nodes to operate in lockstep with each other.
- ➔ Let $dx(y)$ be the cost of the least-cost path from node x to node y . Then the least costs are related by the celebrated Bellman-Ford equation, namely,

$$dx(y) = \min_v \{c(x,v) + dv(y)\}$$

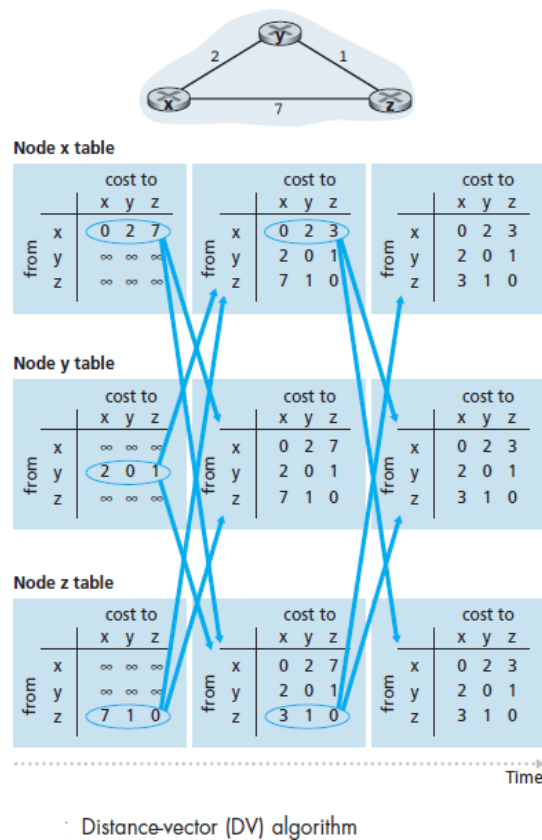
- ➔ With the DV algorithm, each node x maintains the following routing information:
 - For each neighbour v , the cost $c(x,v)$ from x to directly attached neighbour, v
 - Node x 's distance vector, that is, $Dx = [Dx(y): y \text{ in } N]$, containing x 's estimate of its cost to all destinations, y , in N
 - The distance vectors of each of its neighbours, that is, $Dv = [Dv(y): y \text{ in } N]$ for each neighbour v of x

```

1  Initialization:
2    for all destinations y in N:
3       $D_x(y) = c(x,y)$  /* if y is not a neighbor then  $c(x,y) = \infty$  */
4    for each neighbor w
5       $D_w(y) = ?$  for all destinations y in N
6    for each neighbor w
7      send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to w
8
9  loop
10   wait (until I see a link cost change to some neighbor w or
11         until I receive a distance vector from some neighbor w)
12
13   for each y in N:
14      $D_x(y) = \min_v \{c(x,v) + D_v(y)\}$ 
15
16   if  $D_x(y)$  changed for any destination y
17     send distance vector  $D_x = [D_x(y): y \text{ in } N]$  to all neighbors
18
19  forever

```

Figure illustrates the operation of the DV algorithm for the simple three node network shown at the top of the figure.

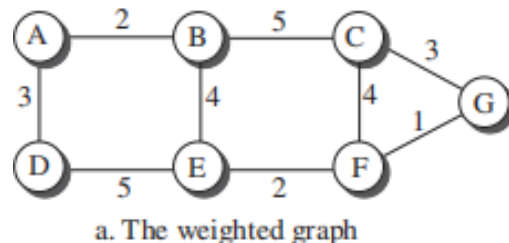


Count to Infinity

A problem with distance-vector routing is that any decrease in cost (good news) propagates quickly, but any increase in cost (bad news) will propagate slowly. For a routing protocol to work properly, if a link is broken (cost becomes infinity), every other router should be aware of it immediately, but in distance-vector routing, this takes some time. The problem is referred to as count to infinity. It sometimes takes several updates before the cost for a broken link is recorded as infinity by all routers.

Link-State Routing

- ➔ Link-State (LS) routing is a method used by routers to determine the best, or least-cost, path through a network.
- ➔ In this method, each connection between routers is referred to as a **link**, and the **link-state** describes the cost of using that connection. Lower costs are preferred, while a link with an infinite cost means it is unavailable or broken.
- ➔ To create efficient routing paths, each router must have a complete map of the network, which includes the state (cost) of every link. This information is stored in the **Link-State Database (LSDB)**, which is shared across all routers.
- ➔ The LSDB is often represented as a matrix, where each value shows the cost of a link between two routers.
- ➔ With this database, each router can build the least-cost tree, ensuring it knows the shortest routes to every other router.



	A	B	C	D	E	F	G
A	0	2	∞	3	∞	∞	∞
B	2	0	5	∞	4	∞	∞
C	∞	5	0	∞	∞	4	3
D	3	∞	∞	0	5	∞	∞
E	∞	4	∞	5	0	2	∞
F	∞	∞	4	∞	2	0	1
G	∞	∞	3	∞	∞	1	0

b. Link state database

- ➔ To create the **Link-State Database (LSDB)**, routers use a process called **flooding**. Each router sends a greeting to its direct neighbors to gather two pieces of information: the neighbor's identity and the cost of the link.
- ➔ This data is packaged into a **Link-State Packet (LSP)** and sent to all neighboring routers. When a router receives an LSP, it checks if the information is new by comparing the sequence number.
- ➔ If the LSP is new, the router keeps it and sends copies to other neighbors, except the one from which it was received.
- ➔ This process repeats until all routers have the latest LSPs, allowing each router to create an identical **LSDB**, which provides a complete map of the network. This LSDB helps routers calculate the least-cost paths to all other nodes.

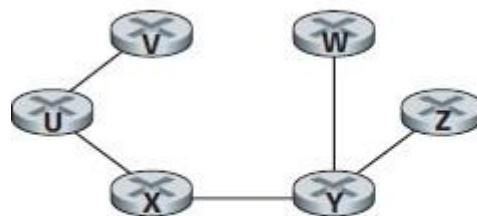
- ➔ The link-state routing algorithm is known as **Dijkstra's algorithm**. Dijkstra's algorithm computes the least-cost path from one node to all other nodes in the network.
- ➔ The following notations are used:
- $D(v)$: cost of the least-cost path from the source node to destination v as of this iteration of the algorithm.
 - $p(v)$: previous node (neighbor of v) along the current least-cost path from the source to v .
 - N' : subset of nodes; v is in N' if the least-cost path from the source to v is definitively known.

```

1  Initialization:
2     $N' = \{u\}$ 
3    for all nodes  $v$ 
4      if  $v$  is a neighbor of  $u$ 
5        then  $D(v) = c(u, v)$ 
6      else  $D(v) = \infty$ 
7
8  Loop
9    find  $w$  not in  $N'$  such that  $D(w)$  is a minimum
10   add  $w$  to  $N'$ 
11   update  $D(v)$  for each neighbor  $v$  of  $w$  and not in  $N'$ :
12      $D(v) = \min( D(v), D(w) + c(w, v) )$ 
13   /* new cost to  $v$  is either old cost to  $v$  or known
14     least path cost to  $w$  plus cost from  $w$  to  $v$  */
15 until  $N' = N$ 

```

step	N'	$D(v), p(v)$	$D(w), p(w)$	$D(x), p(x)$	$D(y), p(y)$	$D(z), p(z)$
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Destination	Link
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

Least cost path and forwarding table for node u

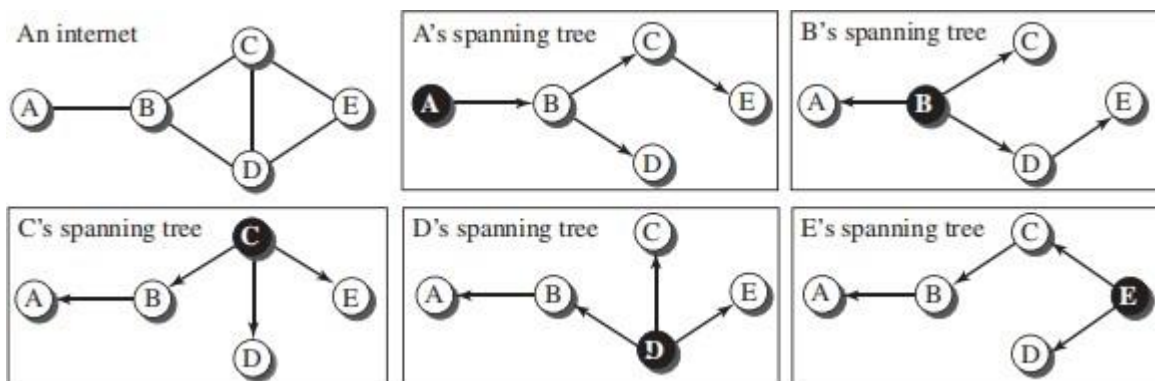
Path-Vector Routing

The **link-state (LS)** and **distance-vector (DV)** routing focus on finding the least-cost path, but there are situations where this goal isn't the priority. For example, a sender may want to avoid certain routers due to security concerns or competition, but least-cost routing doesn't allow for such policies. These methods only focus on minimizing cost, meaning packets could still pass through undesired routers if they are part of the least-cost path. In some cases, such as for security or policy reasons, the sender may prioritize **reachability** over cost, ensuring that packets reach their destination efficiently without considering the route's cost.

Path-vector (PV) routing is a routing algorithm designed to address the limitations of **link-state (LS)** and **distance-vector (DV)** routing, which focus on least-cost paths. Unlike those methods, PV routing allows the source to control the route based on policies, such as avoiding certain routers for security or business reasons. While primarily used for routing between ISPs, PV routing offers flexibility by letting the source determine the best route according to its own policies rather than cost.

Spanning Trees

In **path-vector routing**, the route from a source to all destinations is determined by the **best spanning tree**, but unlike the least-cost tree, the best spanning tree is chosen based on the source's policies. If multiple routes exist, the source can pick the route that aligns with its preferences, which may include avoiding certain nodes or minimizing the number of nodes visited. For example, in a network of five nodes, sources like **A** and **E** may create spanning trees that avoid node **D** as an intermediary, while **B** may avoid node **C**. These policies allow the source to shape the path according to its specific needs.



Creation of Spanning Trees

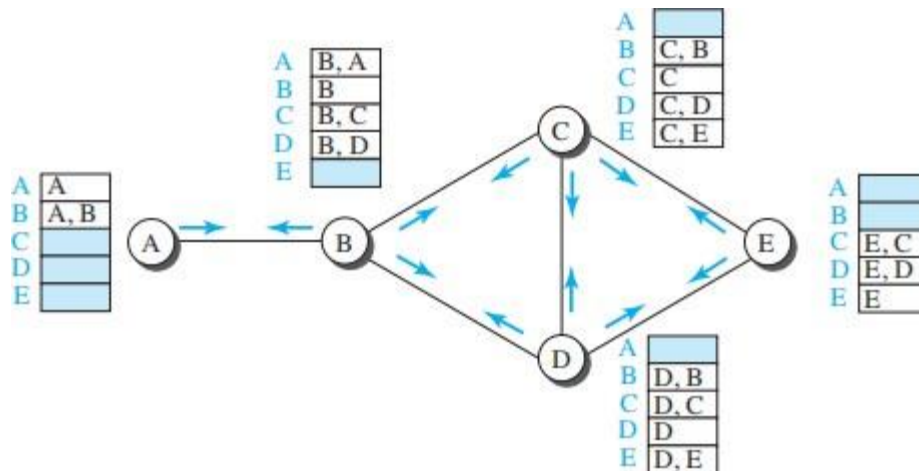
Path-vector routing, like distance-vector routing, is an asynchronous and distributed routing algorithm. The spanning trees are made, gradually and asynchronously, by each node. When a node is booted, it creates a path vector based on the information it can obtain about its immediate

neighbour. A node sends greeting messages to its immediate neighbours to collect these pieces of information.

Each node, after the creation of the initial path vector, sends it to all its immediate neighbours. Each node, when it receives a path vector from a neighbour, updates its path vector using an equation similar to the Bellman-Ford, but applying its own policy instead of looking for the least cost. We can define this equation as

$$\text{Path}(x, y) = \text{best} \{ \text{Path}(x, y), [(x + \text{Path}(v, y))] \} \quad \text{for all } v\text{'s in the internet.}$$

In this equation, the operator (+) means to add x to the beginning of the path. We also need to be cautious to avoid adding a node to an empty path because an empty path means one that does not exist.



The policy is defined by selecting the *best* of multiple paths. Path-vector routing also imposes one more condition on this equation: If Path (v, y) includes x, that path is discarded to avoid a loop in the path. In other words, x does not want to visit itself when it selects a path to y.

```

1 Path_Vector_Routing ( )
2 {
3     // Initialization
4     for (y = 1 to N)
5     {
6         if (y is myself)
7             Path[y] = myself
8         else if (y is a neighbor)
9             Path[y] = myself + neighbor node
10        else
11            Path[y] = empty
12    }
13    Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
14    // Update
15    repeat (forever)
16    {
17        wait (for a vector Pathw from a neighbor w)
18        for (y = 1 to N)
19        {
20            if (Pathw includes myself)
21                discard the path // Avoid any loop
22            else
23                Path[y] = best {Path[y], (myself + Pathw[y])}
24        }
25        If (there is a change in the vector)
26            Send vector {Path[1], Path[2], ..., Path[y]} to all neighbors
27    }
28 } // End of Path Vector

```

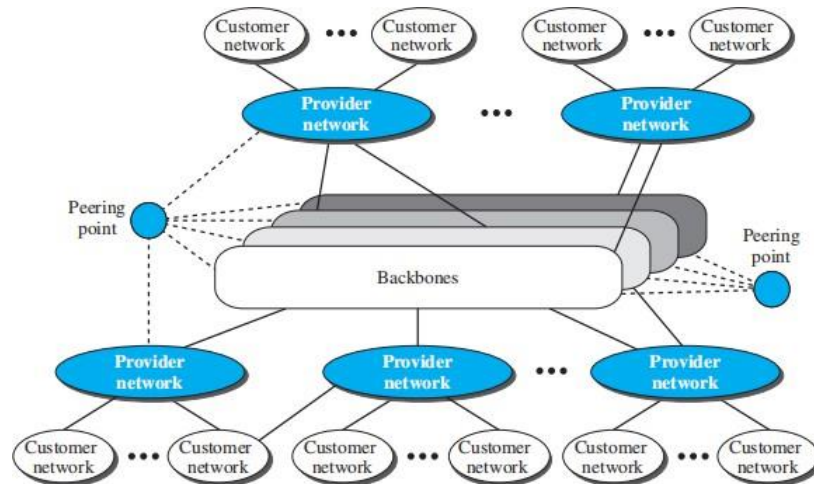
UNICAST ROUTING PROTOCOLS

Routing protocols used in the Internet are based on unicast routing algorithms, but protocols involve more than just algorithms. They also define the domain of operation, the messages exchanged, communication between routers, and interaction with protocols across different domains. The three main Internet routing protocols discussed are Routing Information Protocol (RIP), based on the distance-vector algorithm; Open Shortest Path First (OSPF), based on the

link-state algorithm; and Border Gateway Protocol (BGP), based on the path-vector algorithm. These protocols handle different aspects of routing within and between networks.

Internet Structure

The Internet has changed from a tree-like structure, with a single backbone, to a multi-backbone structure run by different private corporations today. Although it is difficult to give a general view of the Internet today, we can say that the Internet has a structure similar to what is shown in Figure.



There are several *backbones* run by private communication companies that provide global connectivity. These backbones are connected by some *peering points* that allow connectivity between backbones. At a lower level, there are some *provider networks* that use the backbones for global connectivity but provide services to Internet customers.

Finally, there are some *customer networks* that use the services provided by the provider networks. Any of these three entities (backbone, provider network, or customer network) can be called an Internet Service Provider or ISP. They provide services, but at different levels.

Hierarchical Routing

- The Internet is vast, and routing cannot rely on a single protocol due to scalability and administrative challenges:
 - **Scalability problem:** Large forwarding tables make searching slow and traffic updates overwhelming.
 - **Administrative issue:** Each ISP is managed independently and may have unique needs like using specific routing algorithms or hardware.
- Hierarchical routing divides the Internet into **Autonomous Systems (AS)**, where each ISP operates as an AS with control over its routing protocol (intra-AS routing).
- **Intra-AS Routing Protocols (Interior Gateway Protocols or IGPs):** Examples include RIP and OSPF, used within an AS.

- **Inter-AS Routing Protocol (Exterior Gateway Protocol or EGP):** BGP is currently the only inter-AS routing protocol, connecting multiple ASs globally.

Types of Autonomous Systems:

1. **Stub AS:**
 - Connected to only one other AS. Traffic can originate or terminate here, but it does not pass through.
 - Example: A customer network.
2. **Multihomed AS:**
 - Connected to more than one AS but doesn't allow traffic to pass through.
 - Example: A customer AS using services from multiple providers but blocking transit traffic.
3. **Transient AS:**
 - Connected to multiple ASs and allows traffic to pass through.
 - Example: Backbone or provider networks.

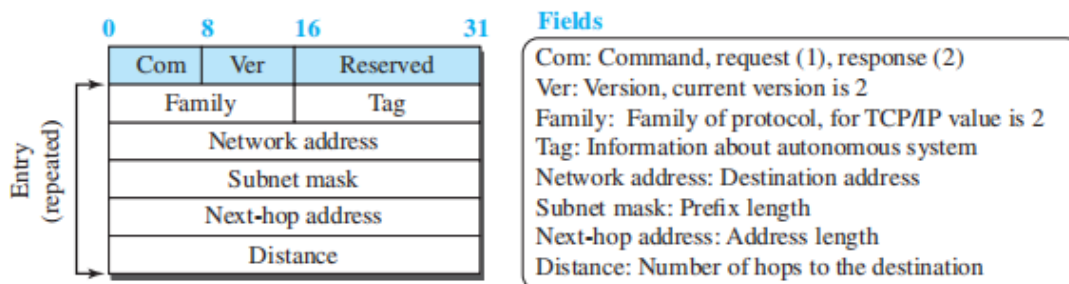
Each AS is assigned a unique Autonomous System Number (ASN) to identify it in the Internet structure.

Routing Information Protocol (RIP)

The Routing Information Protocol (RIP) is one of the most widely used intradomain routing protocols based on the distance-vector routing algorithm we described earlier. RIP was started as part of the Xerox Network System (XNS), but it was the Berkeley Software Distribution (BSD) version of UNIX that helped make the use of RIP widespread.

RIP Messages

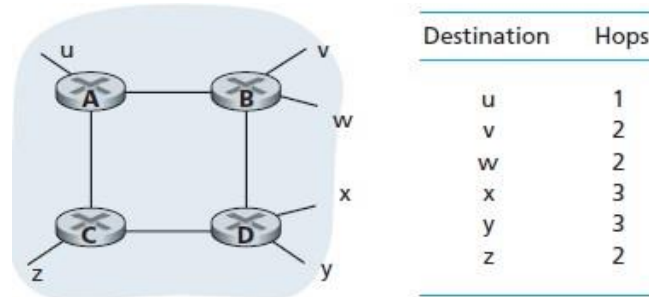
Two RIP processes, a client and a server, like any other processes, need to exchange messages. RIP defines the format of the message. Part of the message, which we call *entry*, can be repeated as needed in a message. Each entry carries the information related to one line in the forwarding table of the router that sends the message.



RIP has two types of messages: request and response. A request message is sent by a router that has just come up or by a router that has some time-out entries. A request message can ask about specific entries or all entries. A response (or update) message can be either solicited or unsolicited. A solicited response message is sent only in answer to a request message. It contains information about the destination specified in the corresponding request message. An unsolicited response message, on the other hand, is sent periodically, every 30 seconds or when there is a change in the forwarding table.

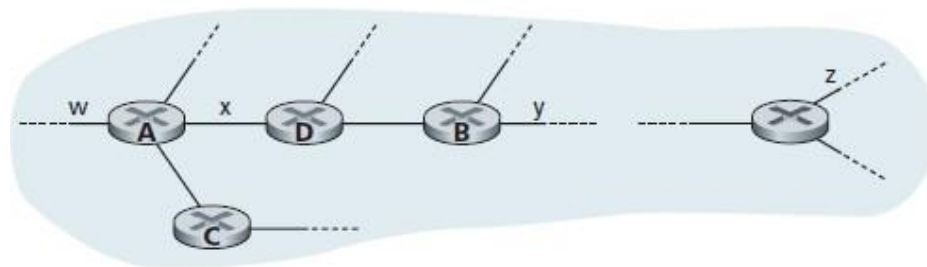
RIP Algorithm

- ➔ RIP is a distance-vector protocol. Figure illustrates an AS with six leaf subnets. The table in the figure indicates the number of hops from the source A to each of the leaf subnets.
- ➔ In RIP, routing updates are exchanged between neighbours approximately every 30 seconds using a RIP response message.



Number of hops from source router A to various subnets

- ➔ Response messages are also known as RIP advertisements. Consider the portion of an AS shown in Figure.



A portion of an autonomous system

- ➔ Figure shows the routing table for router D. suppose that 30 seconds later, router D receives from router A the advertisement shown in next Figure.

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	B	7
x	—	1
....		

Routing table in router D before receiving advertisement from router A

Destination Subnet	Next Router	Number of Hops to Destination
z	C	4
w	—	1
x	—	1
....		

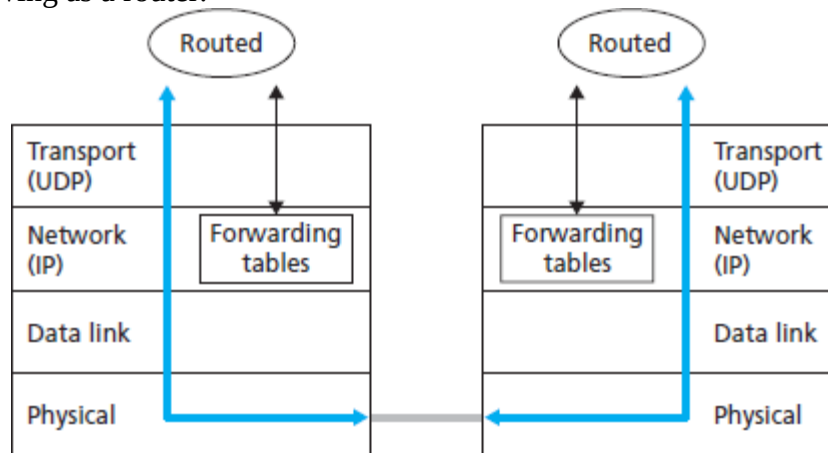
Advertisement from router A

- ➔ Subnet z is only four hops away from router A. Router D, upon receiving the advertisement, merges the advertisement with the old routing table.
- ➔ Router D learns that there is a path through router A to subnet z that is shorter than the path through router B.
- ➔ Thus, router D updates its routing table to account for the shorter shortest path, as shown in Figure.

Destination Subnet	Next Router	Number of Hops to Destination
w	A	2
y	B	2
z	A	5
....		

Routing table in router D after receiving advertisement from router A

Figure below shows how RIP is implemented in a UNIX system, for example, a UNIX workstation serving as a router.



Implementation of RIP as the *routed* daemon

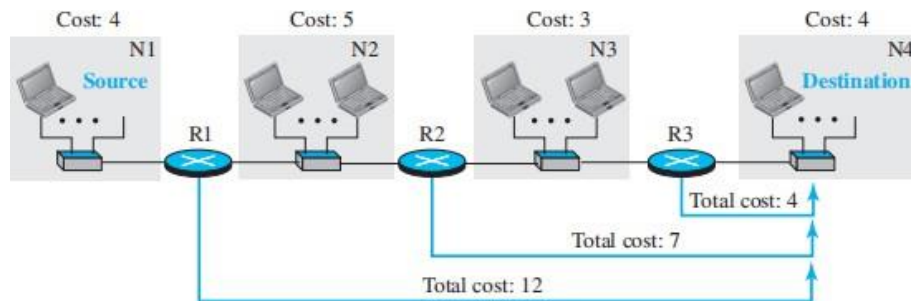
- ➔ A process called **routed** executes RIP, that is, maintains routing information and exchanges messages with routed processes running in neighbouring routers.
- ➔ Because RIP is implemented as an application-layer process, it can send and receive messages over a standard socket and use a standard transport protocol.
- ➔ As shown, RIP is implemented as an application-layer protocol running over UDP.

Open Shortest Path First (OSPF)

Open Shortest Path First (OSPF) is an intradomain routing protocol like RIP but is based on the link-state routing protocol. It is an open protocol, meaning the specification is publicly available. Unlike RIP, OSPF allows each link to be assigned a weight based on factors such as throughput, round-trip time, or reliability, though administrators can use hop count as a cost. Different Types of Service (TOS) can have different weights for cost calculation.

Metric:

OSPF calculates the cost of reaching a destination from a source by considering link weights, which can vary based on the type of service. This is shown in Figure below, where the total cost to the destination is calculated by summing the costs of individual links.



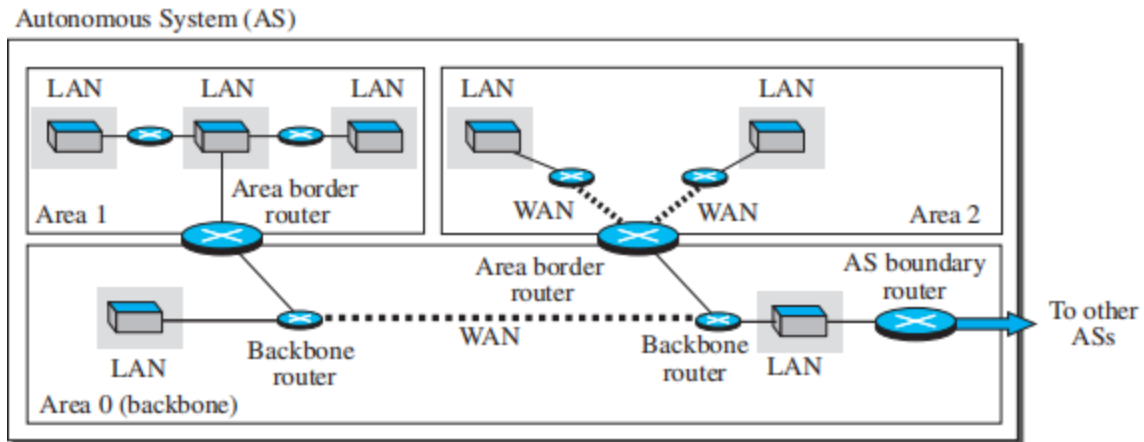
Forwarding Tables:

OSPF routers use Dijkstra's algorithm to create forwarding tables by building the shortest-path tree to destinations. The difference between OSPF and RIP forwarding tables is mainly in the cost values. If OSPF uses hop count as its metric, its forwarding tables would be identical to those of RIP. Both protocols determine the best route using shortest-path trees.

Forwarding table for R1			Forwarding table for R2			Forwarding table for R3		
Destination network	Next router	Cost	Destination network	Next router	Cost	Destination network	Next router	Cost
N1	—	4	N1	R1	9	N1	R2	12
N2	—	5	N2	—	5	N2	R2	8
N3	R2	8	N3	—	3	N3	—	3
N4	R2	12	N4	R3	7	N4	—	4

Areas:

OSPF is designed for both small and large autonomous systems (AS). In large ASs, flooding link-state packets (LSPs) across the entire network can cause congestion, so OSPF introduces areas to localize LSP flooding. The AS is divided into smaller sections called areas, with one backbone area (Area 0) responsible for inter-area communication.



Link-State Advertisement (LSA):

OSPF routers advertise their link states to neighbors for forming a global link-state database (LSDB). Unlike the simple graph model, OSPF distinguishes between different types of nodes and links, requiring various types of advertisements:

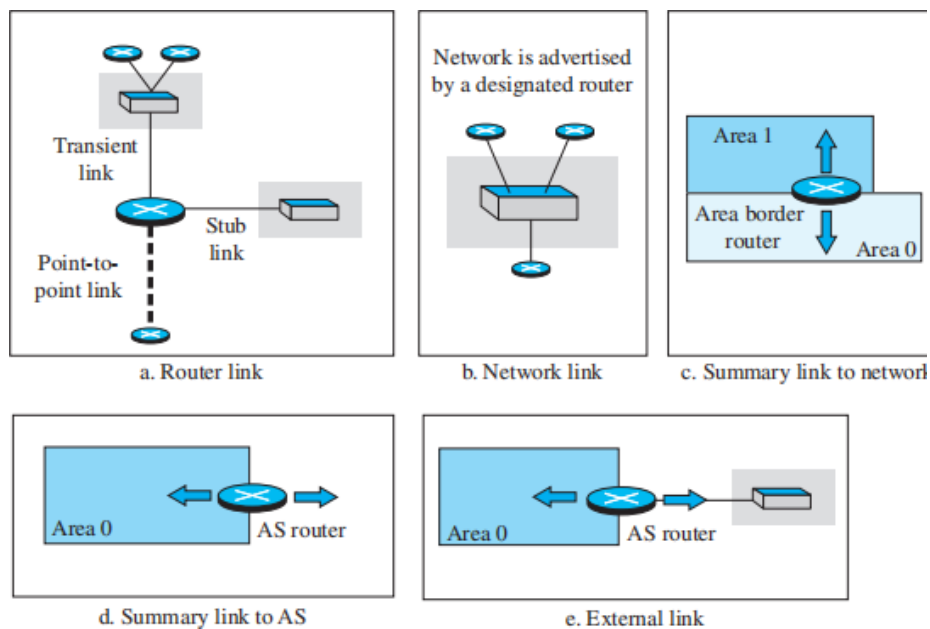
Router Link: Announces router existence and its connection to other entities.

Network Link: Advertises the existence of a network, but with no associated cost.

Summary Link to Network: Advertised by area border routers to summarize links between areas.

Summary Link to AS: Announced by AS boundary routers to inform other areas of external AS links.

External Link: Advertises external network routes to the AS.

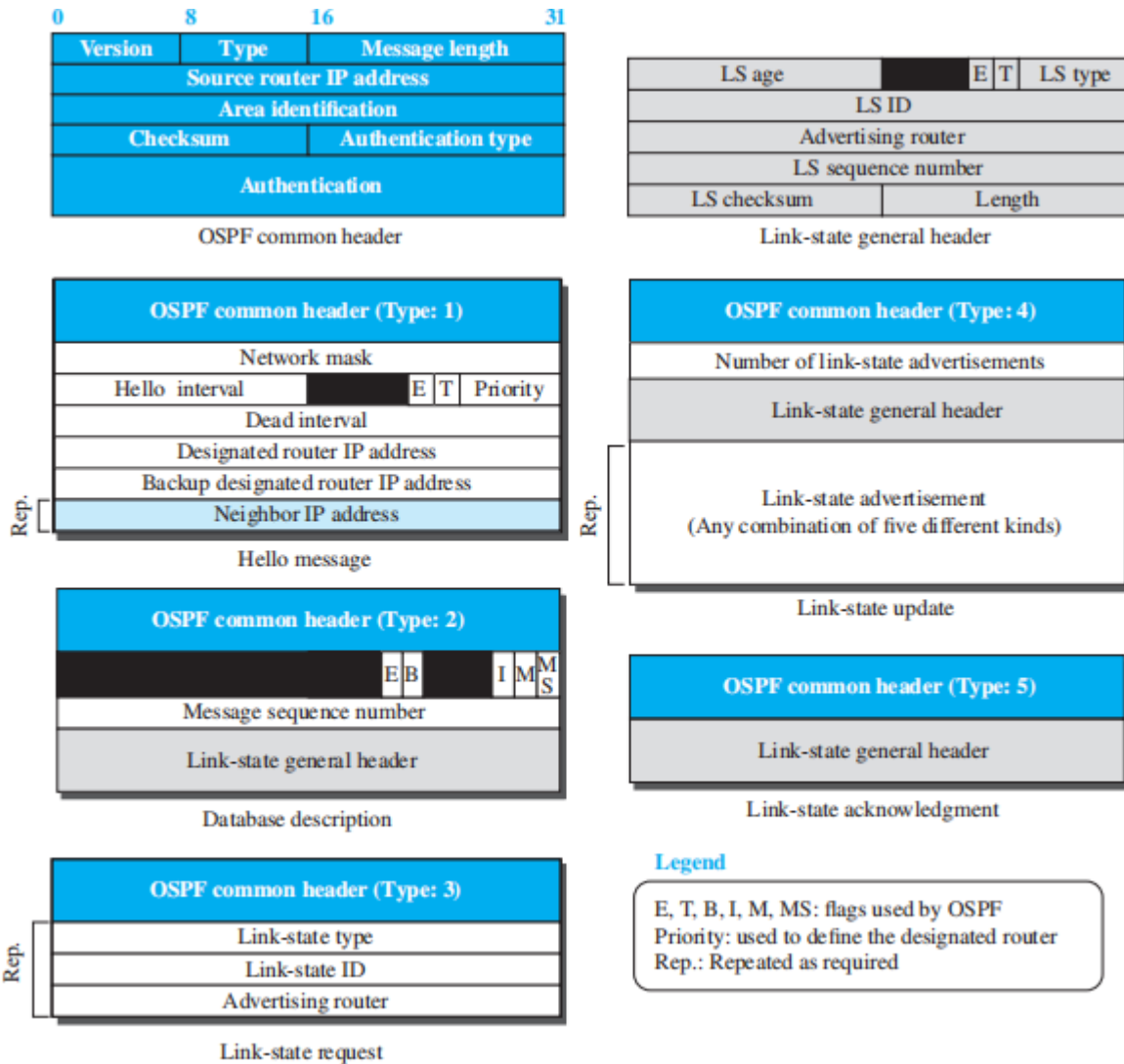


OSPF Implementation:

OSPF operates at the network layer and uses IP for message propagation. OSPF messages are encapsulated in IP datagrams with a protocol field value of 89. OSPF has two versions, with version 2 being the most widely implemented.

OSPF Messages:

OSPF uses five message types, each with a distinct format :



Hello Message (Type 1): Used by routers to introduce themselves and announce their known neighbors.

Database Description Message (Type 2): Sent in response to Hello messages, allowing routers to acquire the full LSDB.

Link-State Request (Type 3): Sent when a router requires specific LS information.

Link-State Update (Type 4): The primary message used to build the LSDB, with versions for each type of link.

Link-State Acknowledgment (Type 5): Provides reliability by confirming receipt of link-state updates.

Authentication: OSPF supports message authentication to prevent unauthorized routers from joining the routing system. This is critical for network security and preventing malicious interference.

OSPF Algorithm:

OSPF uses a modified link-state routing algorithm. After routers form their shortest-path trees, they create corresponding routing tables. The algorithm also handles OSPF message exchange.

Performance:

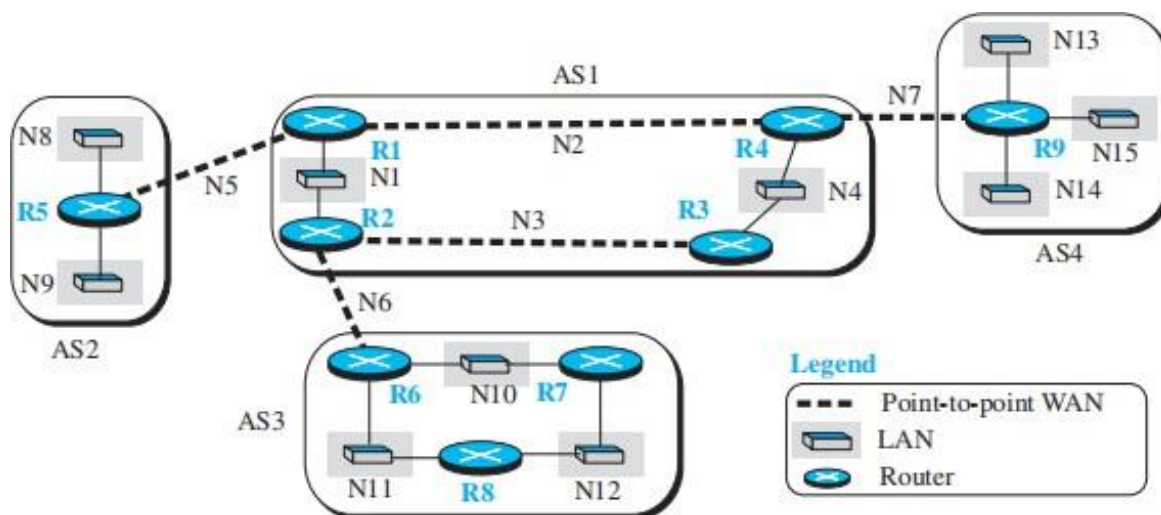
Update Messages: OSPF's LSPs are complex and can create heavy traffic in large areas, using considerable bandwidth.

Convergence of Forwarding Tables: OSPF converges relatively quickly once flooding is complete, although Dijkstra's algorithm can take time to run.

Robustness: OSPF is more robust than RIP since routers operate independently after constructing their LSDBs. A failure in one router has less impact on the overall network.

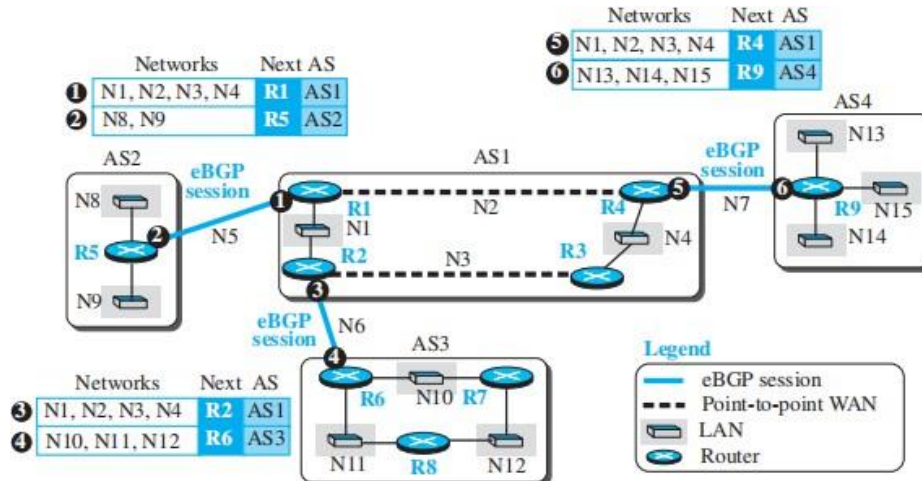
Border Gateway Protocol Version 4 (BGP4)

BGP4 is the predominant interdomain routing protocol used across the internet. It is a path-vector protocol, which means it provides the entire path to the destination instead of just the next hop. This path information is useful for avoiding routing loops and ensuring stable paths between autonomous systems (AS). Figure illustrates a network with four ASs: AS1 is a transient AS, and AS2, AS3, and AS4 are stub ASs. Data exchanges between the stub ASs occur through AS1.



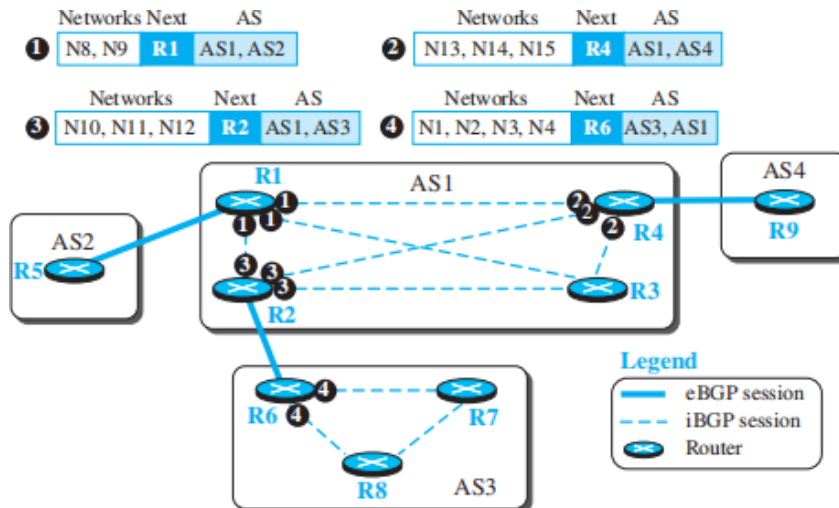
External BGP (eBGP) Operation

BGP operates over Transmission Control Protocol (TCP) connections, ensuring reliable communication between routers. The protocol defines two types of operations: external (eBGP) and internal (iBGP). eBGP sessions are established between routers belonging to different ASs, as seen in Figure. In these sessions, routers share routing information to other ASs by exchanging reachability information about network prefixes. However, this only provides visibility between the border routers, so internal routers need iBGP for full route dissemination.



Internal BGP (iBGP) Operation

Within the same AS, routers establish iBGP sessions to disseminate the routes learned from eBGP sessions. A fully connected mesh of iBGP sessions is required to prevent routing loops. Each iBGP router must have a direct session with every other iBGP router, as depicted in Figure.



This ensures that all routers inside an AS have complete knowledge of external networks. After iBGP sessions converge, all routers will have consistent routing information about the external networks, allowing them to build accurate path tables, as demonstrated in Figure.

Networks	Next	Path	Networks	Next	Path	Networks	Next	Path
N8, N9	R5	AS1, AS2	N8, N9	R1	AS1, AS2	N8, N9	R2	AS1, AS2
N10, N11, N12	R2	AS1, AS3	N10, N11, N12	R6	AS1, AS3	N10, N11, N12	R2	AS1, AS3
N13, N14, N15	R4	AS1, AS4	N13, N14, N15	R1	AS1, AS4	N13, N14, N15	R4	AS1, AS4

Path table for R1

Networks	Next	Path
N8, N9	R1	AS1, AS2
N10, N11, N12	R1	AS1, AS3
N13, N14, N15	R9	AS1, AS4

Path table for R2

Networks	Next	Path
N1, N2, N3, N4	R1	AS2, AS1
N10, N11, N12	R1	AS2, AS1, AS3
N13, N14, N15	R1	AS2, AS1, AS4

Path table for R3

Networks	Next	Path
N1, N2, N3, N4	R2	AS3, AS1
N8, N9	R2	AS3, AS1, AS2
N13, N14, N15	R2	AS3, AS1, AS4

Path table for R4

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R5

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R6

Networks	Next	Path
N1, N2, N3, N4	R4	AS4, AS1
N8, N9	R4	AS4, AS1, AS2
N10, N11, N12	R4	AS4, AS1, AS3

Path table for R7

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

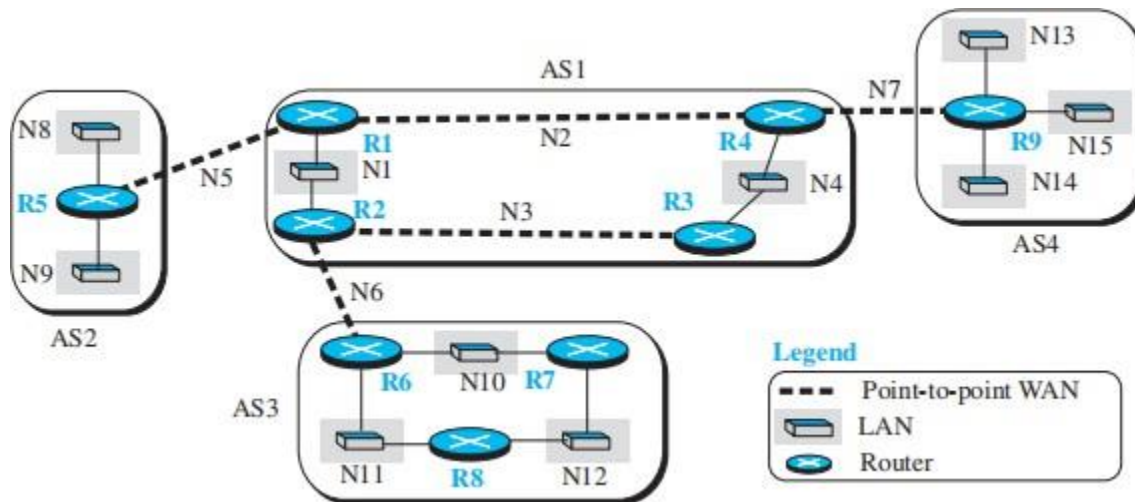
Path table for R8

Networks	Next	Path
N1, N2, N3, N4	R6	AS3, AS1
N8, N9	R6	AS3, AS1, AS2
N13, N14, N15	R6	AS3, AS1, AS4

Path table for R9

Injection into Intradomain Routing

BGP injects routes learned from external sources into the intradomain routing protocol (such as RIP or OSPF). In a stub AS like AS2, the border router typically injects a default route into the internal network, pointing to external destinations. In contrast, in a transient AS such as AS1, more detailed route information is injected into the intradomain routing protocol. This ensures that all routers within the AS have the necessary information to forward packets outside their local AS. Figure shows an example of how the forwarding tables are updated with routes from external ASs.

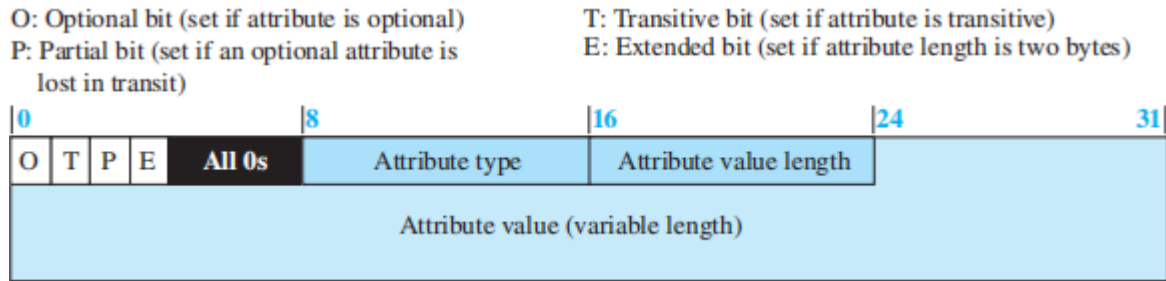


Address Aggregation

BGP employs techniques such as address aggregation to reduce the size of routing tables by summarizing multiple IP prefixes into a single route advertisement. This approach minimizes the number of entries in the forwarding table, optimizing router performance.

Path Attributes

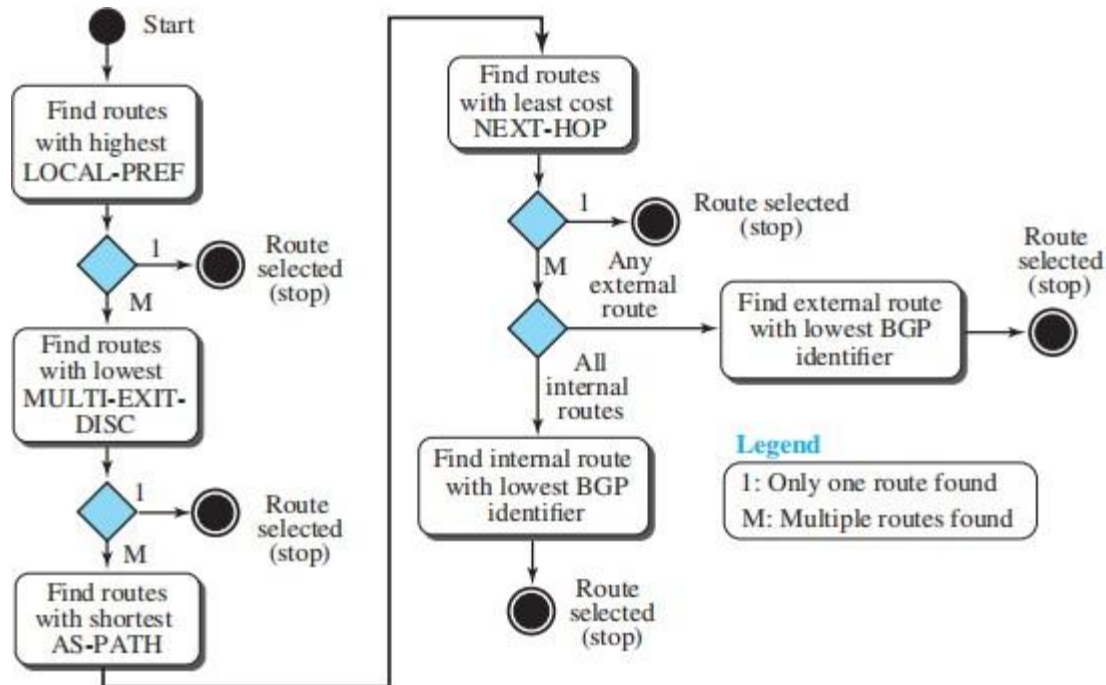
Additionally, BGP uses several path attributes to influence route selection and ensure routing stability. Important attributes include the AS-PATH, which lists all ASs a route has traversed, and the ORIGIN, which indicates how the route was learned. These attributes help routers make intelligent decisions about which paths to choose and avoid loops, as illustrated in Figure.



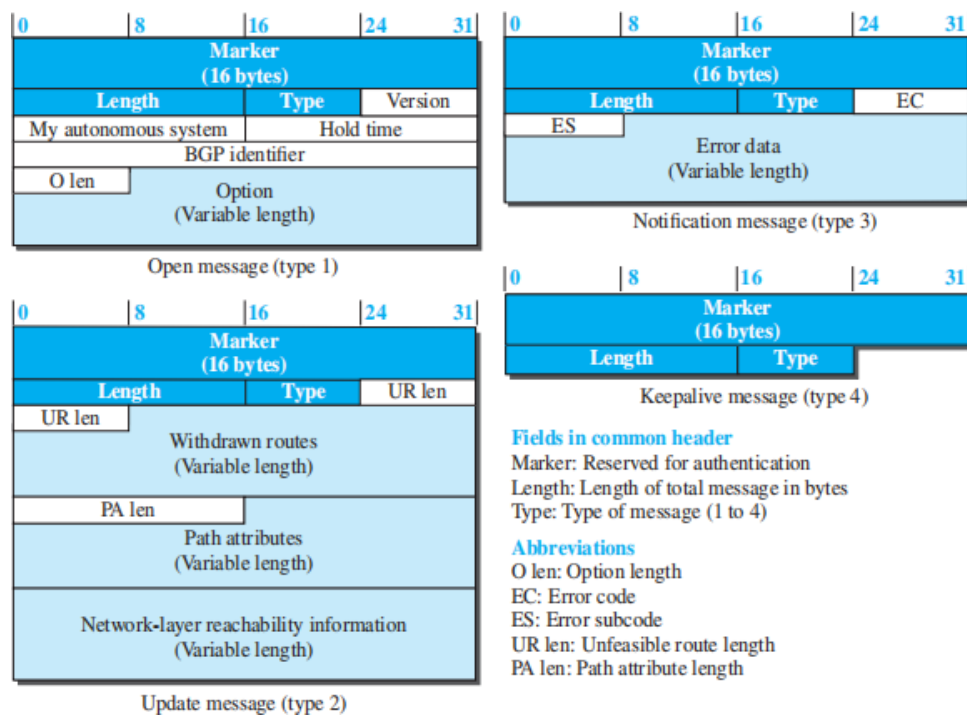
- **ORIGIN (type 1):** A well-known mandatory attribute defining the source of routing information. Value 1 indicates an intradomain protocol (RIP or OSPF), Value 2 indicates BGP, and Value 3 indicates an unknown source.
- **AS-PATH (type 2):** A well-known mandatory attribute listing the autonomous systems through which the destination can be reached. It helps prevent loops and assists in route selection.
- **NEXT-HOP (type 3):** A well-known mandatory attribute defining the next router to forward the data packet. It injects path information into intradomain protocols such as RIP or OSPF.
- **MULT-EXIT-DISC (type 4):** An optional intransitive attribute, used to select among multiple exit paths to a destination. Its value is a 4-byte unsigned integer, typically defined by the metric in intradomain protocols. The path with the lowest value is selected.
- **LOCAL-PREF (type 5):** A well-known discretionary attribute used by administrators to prioritize routes. Routes with higher preference values are chosen based on organizational policy.
- **ATOMIC-AGGREGATE (type 6):** A well-known discretionary attribute with no value field, indicating that the destination prefix is not aggregated.
- **AGGREGATOR (type 7):** An optional transitive attribute, indicating that the destination prefix is an aggregate. The value specifies the AS number and IP address of the last router that performed the aggregation.

Route Selection in BGP

BGP route selection is complex compared to intradomain routing protocols, as it relies on attributes such as LOCAL-PREF and others. The process begins by extracting routes that meet specific criteria in each step. If one route remains, it is selected; otherwise, the next criteria are applied until one route is chosen.

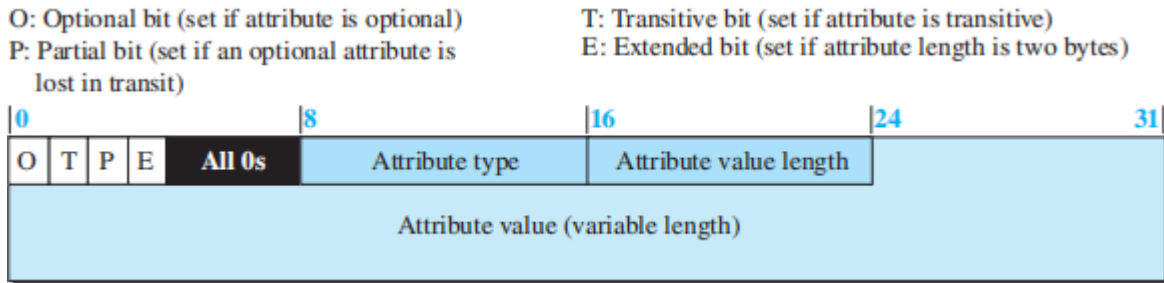


BGP Messages



- **Open Message:** Establishes a neighborhood relationship by initiating a TCP connection.
- **Update Message:** Announces new routes or withdraws previously advertised routes.
- **Keepalive Message:** Regularly exchanged between BGP peers to confirm they are still active.
- **Notification:** Sent when an error is detected or when a router wants to close the session.

Format of Path Attribute



- **Flags:**
 - O: Optional bit (set if the attribute is optional)
 - P: Partial bit (set if an optional attribute is lost in transit)
 - T: Transitive bit (set if the attribute is transitive)
 - E: Extended bit (set if the attribute length is two bytes)
- **Fields:**
 - Attribute type
 - Attribute value length
 - Attribute value (variable)

Multicast Link State (MOSPF)

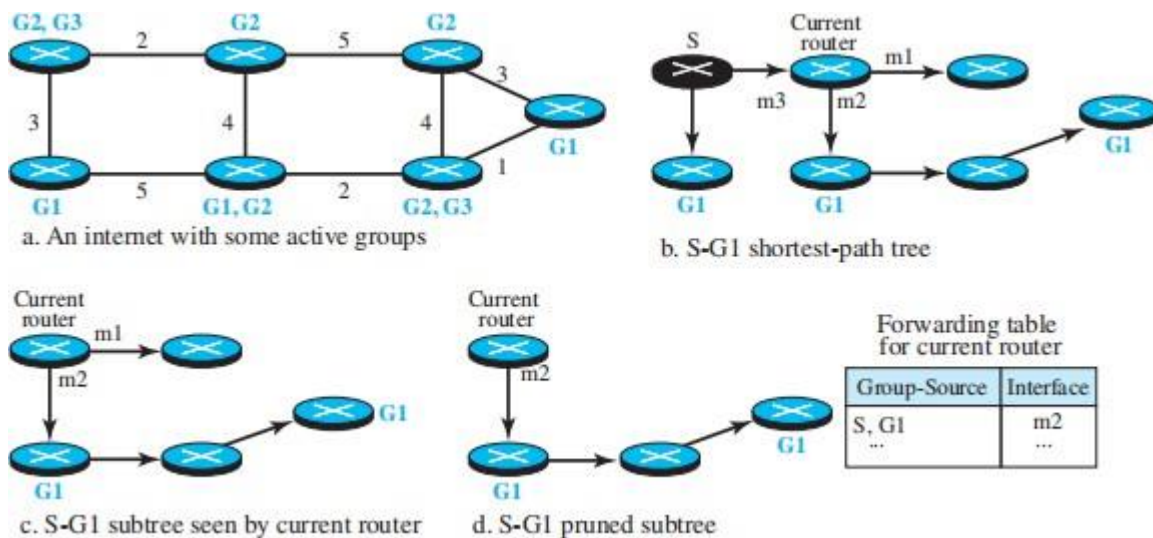
Path First (OSPF) protocol, which is used in unicast routing. It also uses the source-based tree approach to multicasting. If the internet is running a unicast link-state routing algorithm, the idea can be extended to provide a multicast link-state routing algorithm.

To extend unicasting to multicasting, each router needs to have another database, as with the case of unicast distance-vector routing, to show which interface has an active member in a particular group. Now a router goes through the following steps to forward a multicast packet received from source S and to be sent to destination G (a group of recipients):

1. The router uses the Dijkstra algorithm to create a shortest-path tree with S as the root and all destinations in the internet as the leaves. Note that this shortest-path tree is different from the one the router normally uses for unicast forwarding, in which the root of the tree is the router itself. In this case, the root of the tree is the source of the packet defined in the source address of the packet. The router is capable of creating this tree because it has the LSDB, the whole topology of the internet; the Dijkstra algorithm can be used to create a tree with any root, no matter which router is using it. The point we need to remember is that the shortest-path tree created this way depends on the specific source. For each source we need to create a different tree.

2. The router finds itself in the shortest-path tree created in the first step. In other words, the router creates a shortest-path subtree with itself as the root of the subtree.
3. The shortest-path subtree is actually a broadcast subtree with the router as the root and all networks as the leaves. The router now uses a strategy similar to the one we describe in the case of DVMRP to prune the broadcast tree and to change it to a multicast tree. The IGMP protocol is used to find the information at the leaf level. MOSPF has added a new type of link state update packet that floods the membership to all routers. The router can use the information it receives in this way and prune the broadcast tree to make the multicast tree.
4. The router can now forward the received packet out of only those interfaces that correspond to the branches of the multicast tree. We need to make certain that a copy of the multicast packet reaches all networks that have active members of the group and that it does not reach those networks that do not.

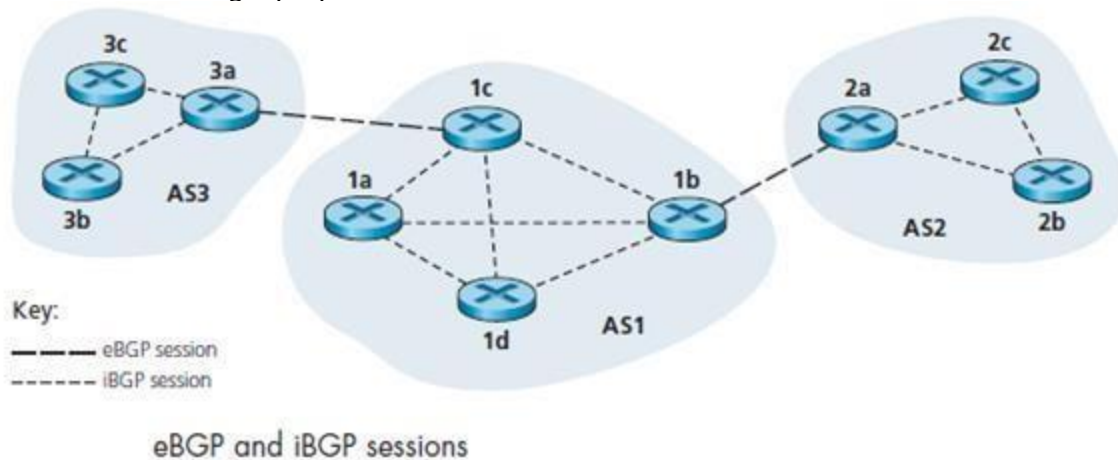
Figure shows an example of using the steps to change a graph to a multicast tree. For simplicity, we have not shown the network, but we added the groups to each router. The figure shows how a source-based tree is made with the source as the root and changed to a multicast subtree with the root at the current router.



NOTE: Simpler Version of BGP(Not in Prescribed text book)

The Border Gateway Protocol version 4 is the standard inter-AS routing protocol in today's Internet.

- It is commonly referred to as BGP4 or as BGP. It provides each AS a means to
 1. Obtain subnet reachability information from neighboring ASs.
 2. Propagate the reachability information to all routers internal to the AS.
 3. Determine "good" routes to subnets based on the reachability information and on AS policy.
- In BGP, pairs of routers exchange routing information over semi-permanent TCP connections.
- In Figure 31, there is a TCP connection between gateway routers 3a and 1c and another TCP connection between gateway routers 1b and 2a.
- There are also semi-permanent BGP TCP connections between routers within an AS.
- For each TCP connection, the two routers at the end of the connection are called BGP peers, and the TCP connection along with all the BGP messages sent over the connection is called a BGP session.
- A BGP session that spans two ASs is called an external BGP (eBGP) session, and a BGP session between routers in the same AS is called an internal BGP (iBGP) session.
- Suppose there are four subnets attached to AS2: 138.16.64/24, 138.16.65/24, 138.16.66/24, and 138.16.67/24. Then AS2 could aggregate the prefixes for these four subnets and use BGP to advertise the single prefix to 138.16.64/22 to AS1.

**Path Attributes and BGP Routes**

When a router advertises a prefix across a BGP session, it includes with the prefix a number of BGP attributes. A prefix along with its attributes is called a route. Two important attributes are **AS-PATH** and **NEXT-HOP**:

1. **AS-PATH.** This attribute contains the ASs through which the advertisement for the prefix has passed. When a prefix is passed into an AS, the AS adds its ASN to the AS-PATH attribute. For example in Figure 31, suppose the prefix 138.16.64/24 is first advertised from AS2 to AS1 and then to AS3, AS-PATH would be AS2 AS1. Routers use the AS-PATH attribute to detect and prevent looping advertisements.

2. Providing the critical link between inter-AS and intra-AS routing protocols. The NEXT-HOP is the router interface that begins the AS-PATH. In Figure 31, suppose the gateway router 3a in AS3 advertises a route to gateway router 1c in AS1 using eBGP. The route includes the advertised prefix, say x, and an AS-PATH to the prefix. This advertisement also includes the NEXT-HOP, which is the IP address of the router 3a interface that leads to 1c.

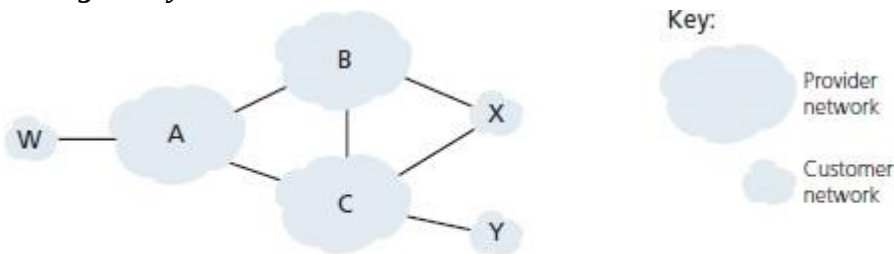
BGP Route Selection

If there are two or more routes to the same prefix, then BGP invokes the following elimination rules until one route remains:

- Routes are assigned a local preference value as one of their attributes. The policy decision is left to the AS's network administrator. The routes with the highest local preference values are selected.
- From the remaining routes, the route with the shortest AS-PATH is selected. If this is the only rule for route selection, then BGP would use a DV algorithm for path determination, where the distance metric uses the number of AS hops.
- From the remaining routes, the route with the closest NEXT-HOP router is selected. It is determined by the intra-AS algorithm by hot potato routing process.

If more than one route still remains, the router uses BGP identifiers to select the route.

Routing Policy



A simple BGP scenario

- Figure shows six interconnected autonomous systems: A, B, C, W, X, and Y.
- Assume that autonomous systems W, X, and Y are stub networks and that A, B, and C are backbone provider networks.
- Also assume that A, B, and C, all peer with each other, and provide full BGP information to their customer networks.
- All traffic entering a stub network must be destined for that network, and all traffic leaving a stub network must have originated in that network. W and Y are clearly stub networks.
- X is a multi-homed stub network, since it is connected to the rest of the network via two different providers.
- X is prevented from forwarding traffic between B and C by controlling the manner in which BGP routes are advertised.
- That is, even though X may know of a path, say XCY, that reaches network Y, it will not advertise this path to B.
- Since B is unaware that X has a path to Y, B would never forward traffic destined to Y (or C) via X.

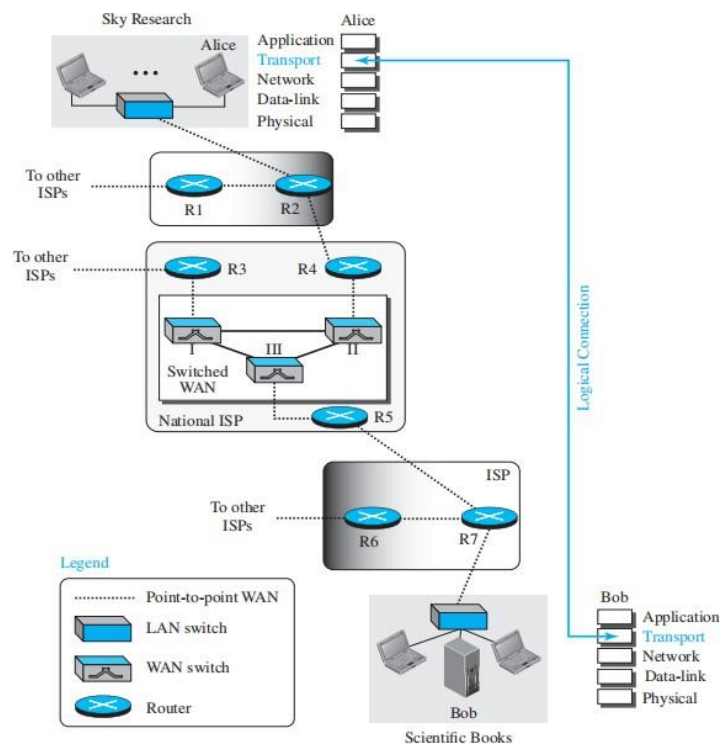
o This simple example shows how a selective route advertisement policy can be used to implement customer/provider routing relationships.

Module 4

Introduction to Transport layer

INTRODUCTION

The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host. Communication is provided using a logical connection, which means that the two application layers, which can be located in different parts of the globe, assume that there is an imaginary direct connection through which they can send and receive messages.

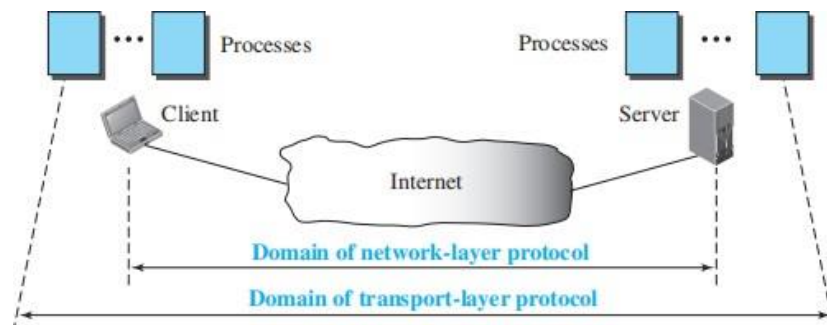


Transport-Layer Services

The transport layer is responsible for providing services to the application layer; it receives services from the network layer.

The first duty of a transport-layer protocol is to provide process-to-process communication. A process is an application-layer entity (running program) that uses the services of the transport layer.

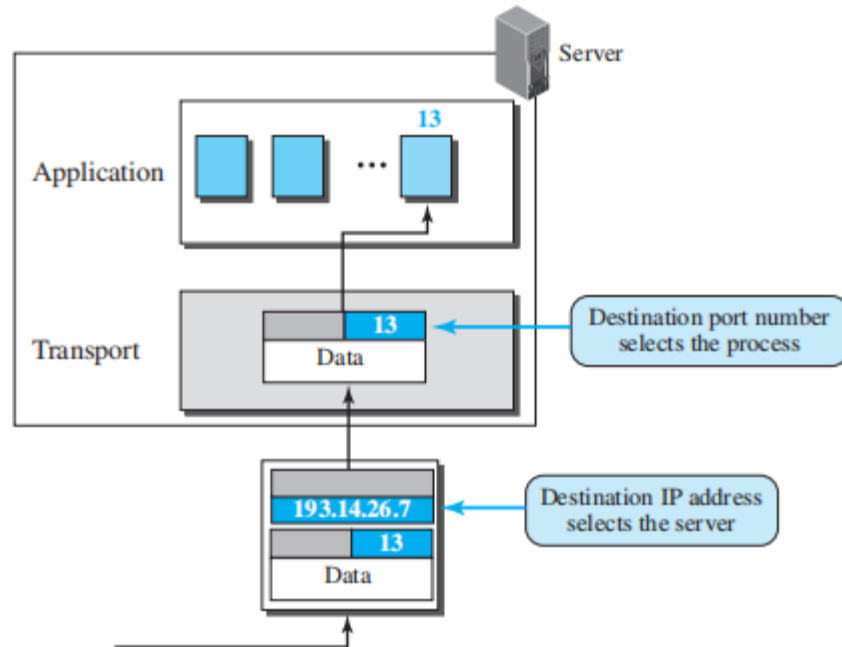
The network layer is responsible for communication at the computer level (host-to-host communication). A network-layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport-layer protocol takes over. A transport-layer protocol is responsible for delivery of the message to the appropriate process.



Addressing: Port Numbers

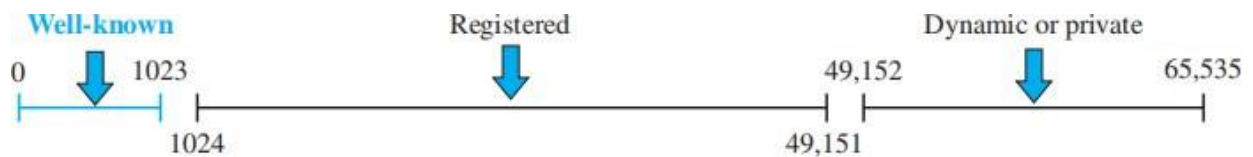
Although there are a few ways to achieve process-to-process communication, the most common is through the **client-server paradigm**.

- ➔ A process on the local host, called a *client*, needs services from a process usually on the remote host, called a *server*.
- ➔ The operating systems today support both multiuser and multiprogramming environments. A remote computer can run multiple server programs at once, just like different local computers can each run one or more client programs at the same time.
- ➔ For communication, we must define the local host, local process, remote host, and remote process. The local host and the remote host are defined using IP addresses. To define the processes, we need **port numbers** (0-65,535 (16 bits)).
- ➔ The client program defines itself with a port number, called the **ephemeral port number**. The word ephemeral means “short-lived” and is used because the life of a client is normally short.
- ➔ The server process must also define itself with a port number. TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers.



ICANN Ranges

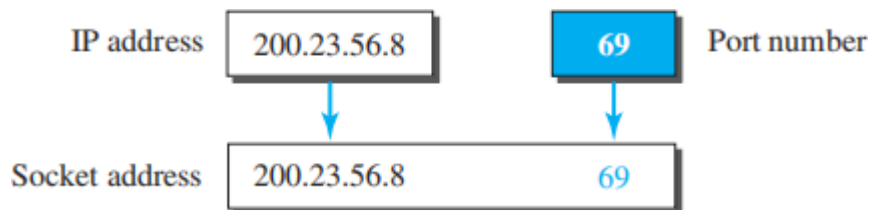
ICANN has divided the port numbers into three ranges: well-known, registered, and dynamic (or private).



1. Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by ICANN. These are the well-known ports.
2. Registered ports. The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
3. Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.

Socket Addresses

A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection. The combination of an IP address and a port number is called a **socket address**. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely.

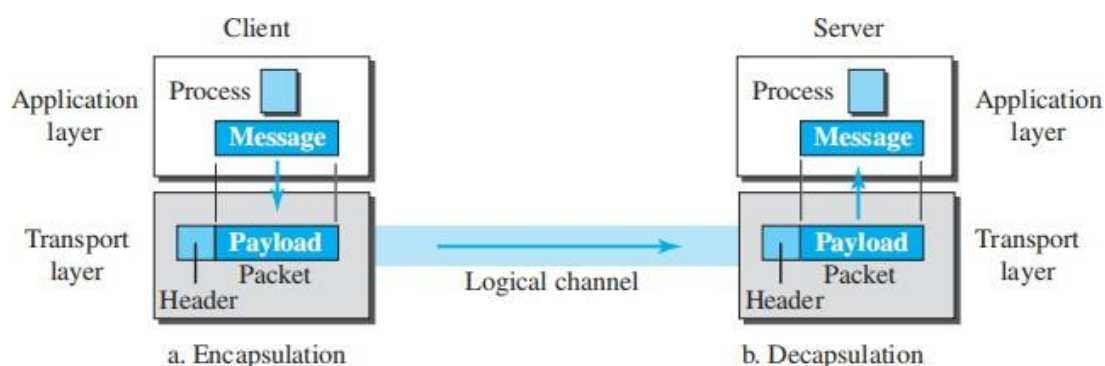


To use the services of the transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header. The first header contains the IP addresses; the second header contains the port numbers.

Encapsulation and Decapsulation

Encapsulation happens on the sender's side: When a program wants to send a message, it gives the message to the transport layer, along with the socket addresses and other necessary info. The transport layer then adds a header to the message, creating a packet. Depending on the transport protocol, this packet might be called a user datagram, segment, or just a packet.

Decapsulation happens on the receiver's side: When the packet reaches its destination, the transport layer removes the header and delivers the message to the application program. The sender's socket address is also passed along so the program can respond if needed.



Multiplexing and Demultiplexing

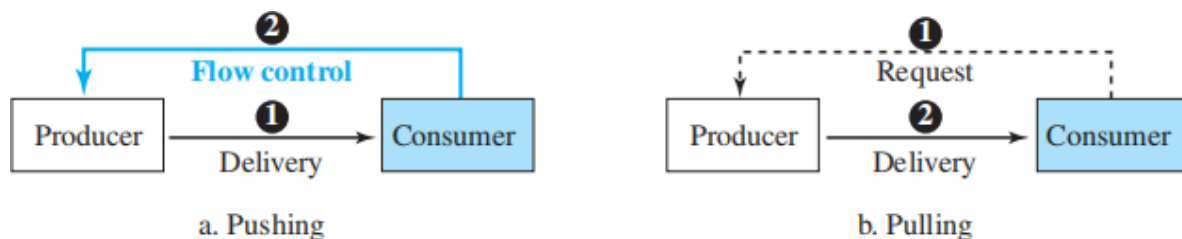
Whenever an entity accepts items from more than one source, this is referred to as **multiplexing** (many to one); whenever an entity delivers items to more than one source, this is referred to as **demultiplexing** (one to many). The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.

Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items. If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

Pushing or Pulling

Delivery of items from a producer to a consumer can occur in one of two ways: *pushing* or *pulling*. If the sender delivers items whenever they are produced without a prior request from the consumer the delivery is referred to as *pushing*. If the producer delivers the items after the consumer has requested them, the delivery is referred to as *pulling*.

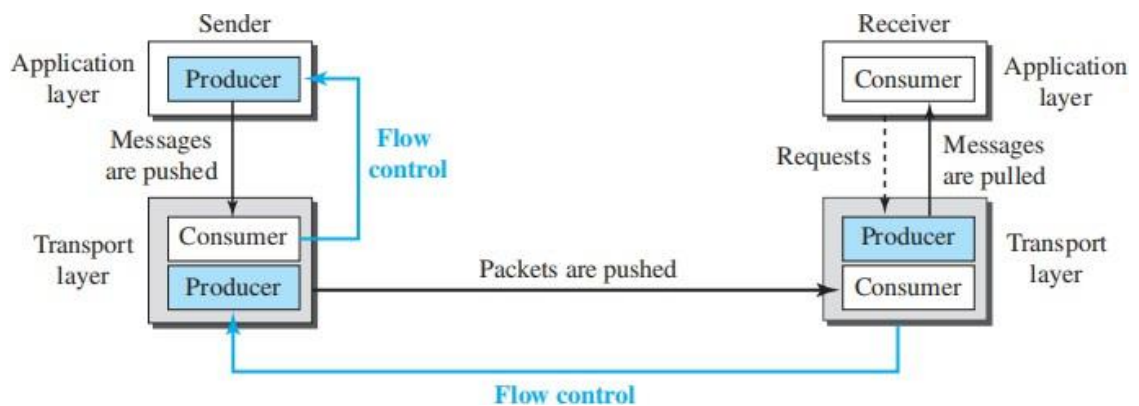


When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items.

Flow Control at Transport Layer

- ➔ In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process.
- ➔ The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer.

- ➔ The sending transport layer has a double role: it is both a consumer and a producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer.
- ➔ The receiving transport layer also has a double role: it is the consumer for the packets received from the sender and the producer that decapsulates the messages and delivers them to the application layer.
- ➔ The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.
- ➔ Figure shows that we need at least two cases of flow control: from the sending transport layer to the sending application layer and from the receiving transport layer to the sending transport layer.



Buffers

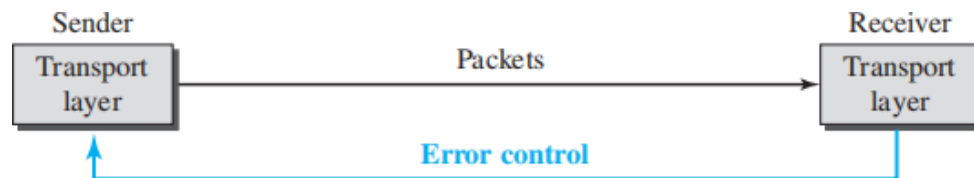
A buffer is a set of memory locations that can hold packets at the sender and receiver. The flow control communication can occur by sending signals from the consumer to the producer. When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.

When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send packets again.

Error Control

In the Internet, since the underlying network layer (IP) is unreliable, we need to make the transport layer reliable if the application requires reliability. Reliability can be achieved to add error control services to the transport layer. Error control at the transport layer is responsible for

1. Detecting and discarding corrupted packets.
2. Keeping track of lost and discarded packets and resending them.
3. Recognizing duplicate packets and discarding them.
4. Buffering out-of-order packets until the missing packets arrive.



Sequence Numbers

- ➔ Error control requires that the sending transport layer knows which packet is to be resent and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order. This can be done if the packets are numbered.
- ➔ We can add a field to the transport-layer packet to hold the **sequence number** of the packet. When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.
- ➔ The receiving transport layer can also detect duplicate packets if two received packets have the same sequence number.
- ➔ The out-of-order packets can be recognized by observing gaps in the sequence numbers. Packets are numbered sequentially.
- ➔ Packets are numbered sequentially. However, because we need to include the sequence number of each packet in the header, we need to set a limit.
- ➔ If the header of the packet allows m bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$.
- ➔ For example, if m is 4, the only sequence numbers are 0 through 15, inclusive. However, we can wrap around the sequence. So the sequence numbers in this case are

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

Acknowledgment

The receiver side can send an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound. The receiver can simply discard the corrupted packets. The sender can detect lost packets if it uses a timer. When a packet is sent, the sender starts a timer. If an ACK does not arrive before the timer expires, the sender resends the packet. Duplicate packets can be silently discarded by the receiver. Out-of-order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing one arrives.

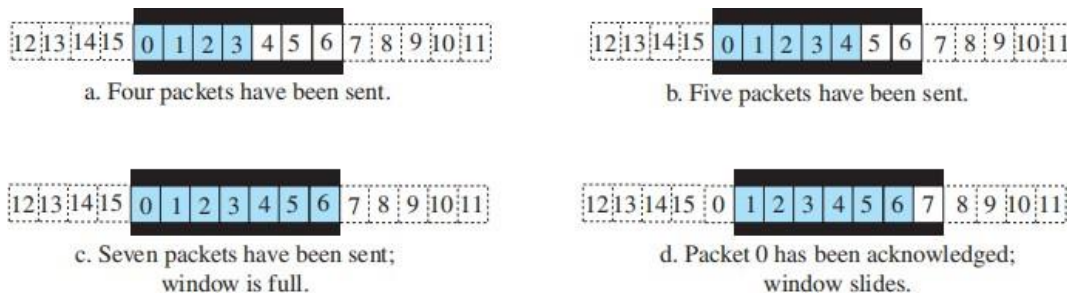
Combination of Flow and Error Control

The flow control requires the use of two buffers, one at the sender site and the other at the receiver site. The error control requires the use of sequence and acknowledgment numbers by both sides. These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver. At the sender, when a packet is prepared to be sent, we use the number of the next free location, x , in the buffer as the sequence number of the packet. When the packet is sent, a copy is stored at memory location x , awaiting the acknowledgment from the other end. When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free. At the receiver, when a packet with sequence number y arrives, it is stored at the memory location y until the application layer is ready to receive it. An acknowledgment can be sent to announce the arrival of packet y .

Sliding Window

The sliding window is a technique for sending multiple frames at a time. It controls the data packets between the two devices where reliable and gradual delivery of data frames is needed. It is also used in [TCP \(Transmission Control Protocol\)](#).

In this technique, each frame has sent from the sequence number. The sequence numbers are used to find the missing data in the receiver end. The purpose of the sliding window technique is to avoid duplicate data, so it uses the sequence number.



Congestion Control

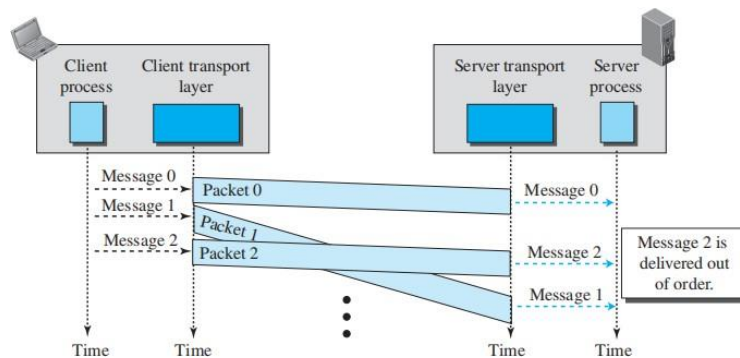
Congestion in a packet-switched network, like the Internet, happens when the load—number of packets sent—exceeds the network's capacity to handle them. Congestion control refers to methods used to prevent this by keeping the load below capacity. Similar to traffic jams on a freeway, congestion occurs in networks when there are delays, such as when routers and switches can't process packets as fast as they receive them. Routers have input and output queues, and when these queues get overloaded, congestion happens. This problem starts at the network layer but affects the transport layer, where protocols like TCP implement their own congestion control if the network layer doesn't handle it.

Connectionless and Connection-Oriented Protocols

A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented

Connectionless Service

In a **connectionless service**, the source application divides its message into chunks of data and sends them to the **transport layer**, which treats each chunk independently. There is no relationship between the chunks, so they may arrive out of order at the destination. For example, a client might send three chunks (0, 1, and 2), but due to delays, the server could receive them out of order (0, 2, 1), as shown in **Figure** . This could result in a garbled message. If one packet is lost, since there is no numbering or coordination between the transport layers, the receiving side won't know and will deliver incomplete data. This lack of **flow control**, **error control**, and **congestion control** makes the system inefficient.

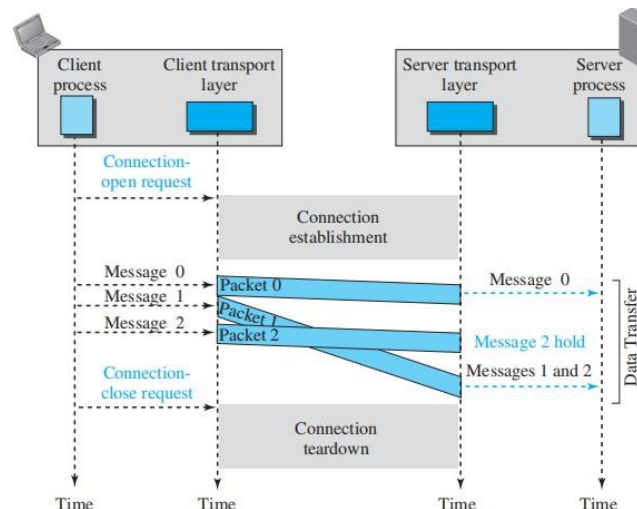


Connection-Oriented Service

In a connection-oriented service, the client and the server first need to establish a logical connection between themselves. The data exchange can only happen after the connection establishment.

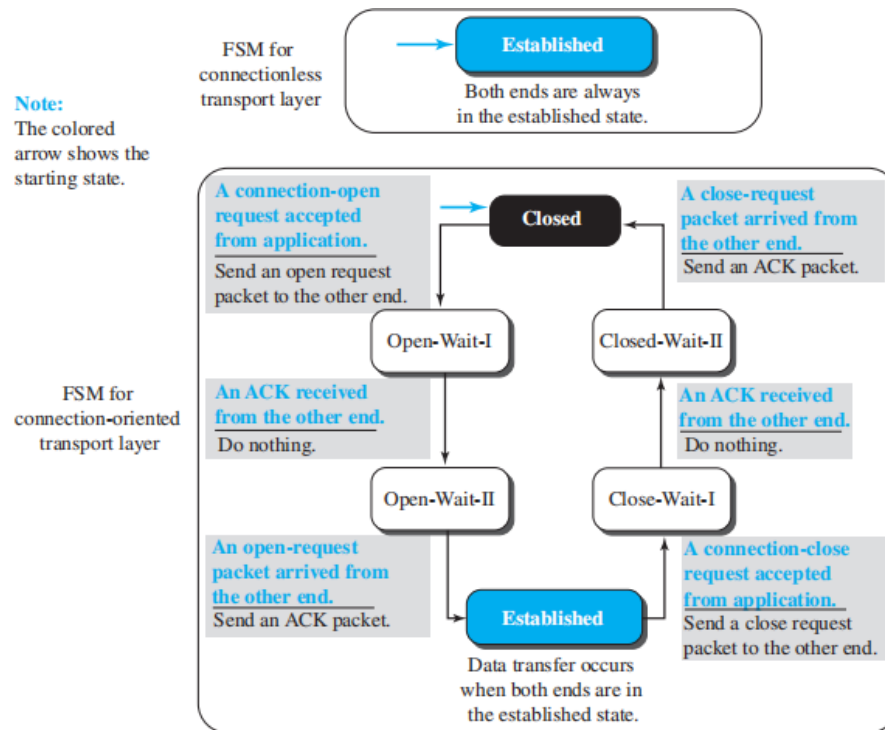
Figure shows the connection establishment, data-transfer, and tear-down phases in a connection-oriented service at the transport layer.

We can implement flow control, error control, and congestion control in a connection oriented protocol.



The behavior of a transport-layer protocol, both when it provides a connectionless and when it provides a connection-oriented protocol, can be better shown as a **finite state machine (FSM)**. Using this tool, each transport layer (sender or receiver) is taught as a machine with a finite number of states.

Every FSM must have an initial state, which is where the machine starts when it turns on. In diagrams, rounded rectangles are used to represent states, colored text indicates events, and black text shows actions. A horizontal line or a slash separates the event from the action, and arrows depict the transition to the next state.



In a **connectionless transport layer**, the FSM has only one state: the established state. The machines on both the client and server sides remain in the established state, always ready to send and receive transport-layer packets.

A **connection-oriented FSM** requires several states for **establishment** and **termination**:

- **Closed State:** The FSM starts here when there is no connection. It stays in this state until it receives an **open request** from the local process.
- The machine sends an **open request packet** to the remote transport layer and transitions to **open-wait-I**.
- When the acknowledgment is received, it moves to **open-wait-II**. At this point, a **unidirectional connection** is established.

- If a **bidirectional connection** is needed, it waits in this state until the other end also sends a connection request. Upon receiving it, the machine sends an acknowledgment and moves to the **established state**.

In both connectionless and connection-oriented transport layers, the established state represents the stage where data and acknowledgments can be exchanged between the two ends. The established state encompasses a set of data transfer states.

When **tearing down** a connection, the application layer sends a close request to the transport layer, which then sends a close request packet to the other end and transitions to close-wait-I.

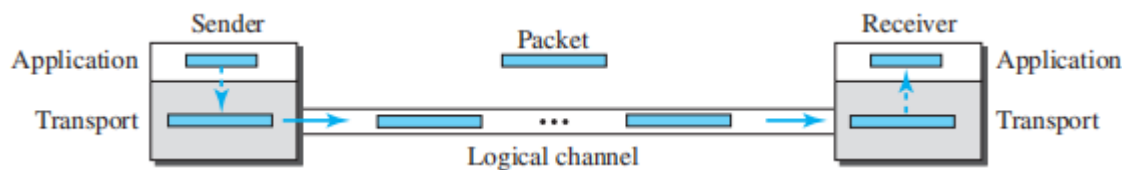
Once an acknowledgment is received, the machine moves to close-wait-II and waits for a close request from the other side. When the request arrives, the machine sends an acknowledgment and moves back to the closed state.

There are various versions of the connection-oriented FSM, where states can be condensed, expanded, or renamed depending on the protocol being used.

TRANSPORT-LAYER PROTOCOLS

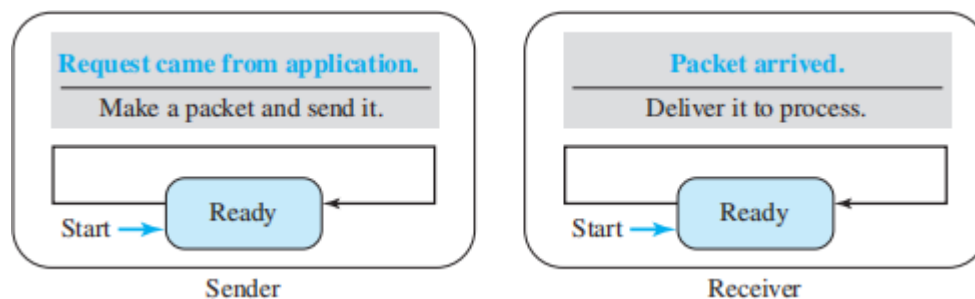
Simple Protocol

Our first protocol is a simple connectionless protocol with neither **flow nor error control**. We assume that the receiver can immediately handle any packet it receives.



The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet. The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer.

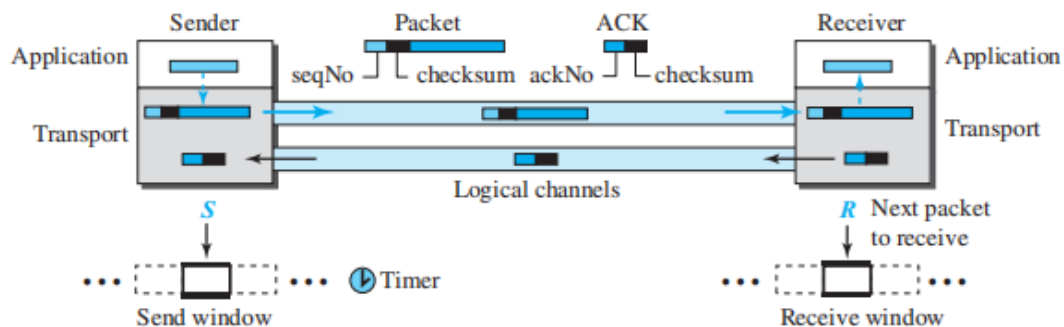
FSMs



- ➔ The sender site should not send a packet until its application layer has a message to send. The receiver site cannot deliver a message to its application layer until a packet arrives.
- ➔ We can show these requirements using two FSMs. Each FSM has only one state, the *ready state*.
- ➔ The sending machine remains in the ready state until a request comes from the process in the application layer. When this event occurs, the sending machine encapsulates the message in a packet and sends it to the receiving machine.
- ➔ The receiving machine remains in the ready state until a packet arrives from the sending machine. When this event occurs, the receiving machine decapsulates the message out of the packet and delivers it to the process at the application layer.

Stop-and-Wait Protocol

- ➔ Stop-and-Wait is a connection-oriented protocol, which uses both **flow** and **error control**.
- ➔ Both the sender and the receiver use a **sliding window of size 1**. The sender sends one packet at a time and waits for an acknowledgment before sending the next one.
- ➔ To detect corrupted packets, we need to add a **checksum** to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded.
- ➔ The silence of the receiver is a signal for the sender that a packet was either corrupted or lost. Every time the sender sends a packet, it starts a timer.
- ➔ If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet (if it has one to send).
- ➔ If the timer expires, the sender resends the previous packet, assuming that the packet was either lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives.



Sequence Numbers

To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. A field is added to the packet header to hold the sequence number of that packet.

Acknowledgment Numbers

Since the sequence numbers must be suitable for both data packets and acknowledgments, we use this convention: The acknowledgment numbers always announce the sequence number of the *next packet expected* by the receiver.

FSMs

Since the protocol is a connection-oriented protocol, both ends should be in the *established* state before exchanging data packets. The states are actually nested in the *established* state.

Sender

The sender is initially in the ready state, but it can move between the ready and blocking state. The variable S is initialized to 0.

Ready state. When the sender is in this state, it is only waiting for one event to occur. If a request comes from the application layer, the sender creates a packet with the sequence number set to S . A copy of the packet is stored, and the packet is sent. The sender then starts the only timer. The sender then moves to the blocking state.

Blocking state. When the sender is in this state, three events can occur:

- a. If an error-free ACK arrives with the ackNo related to the next packet to be sent, which means $\text{ackNo} = (S + 1) \text{ modulo } 2$, then the timer is stopped. The window slides, $S = (S + 1) \text{ modulo } 2$. Finally, the sender moves to the ready state.
- b. If a corrupted ACK or an error-free ACK with the $\text{ackNo} \neq (S + 1) \text{ modulo } 2$ arrives, the ACK is discarded.
- c. If a time-out occurs, the sender resends the only outstanding packet and restarts the timer.

Receiver

The receiver is always in the *ready* state. Three events may occur:

- a. If an error-free packet with $\text{seqNo} = R$ arrives, the message in the packet is delivered to the application layer. The window then slides, $R = (R + 1) \text{ modulo } 2$. Finally an ACK with $\text{ackNo} = R$ is sent.
- b. If an error-free packet with $\text{seqNo} \neq R$ arrives, the packet is discarded, but an ACK with $\text{ackNo} = R$ is sent.
- c. If a corrupted packet arrives, the packet is discarded.

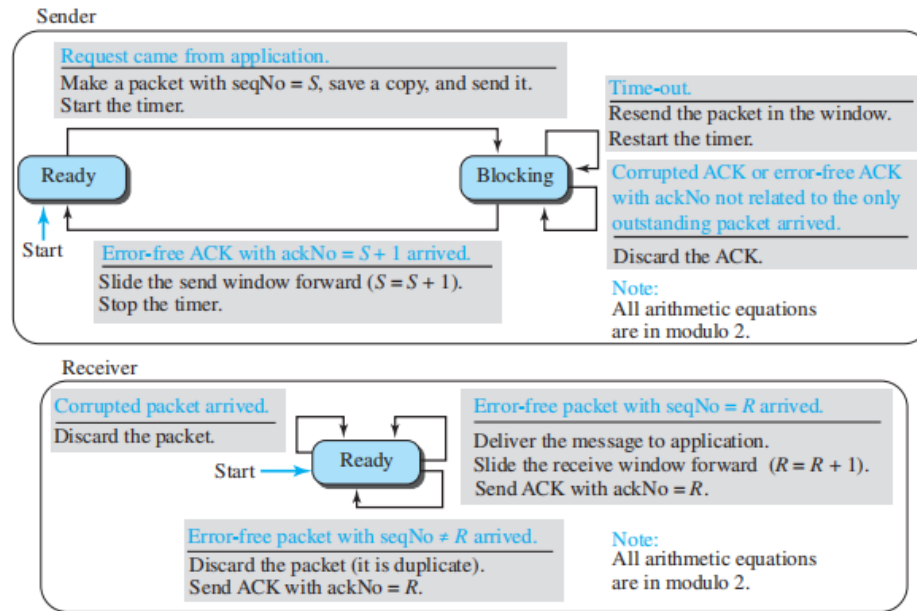
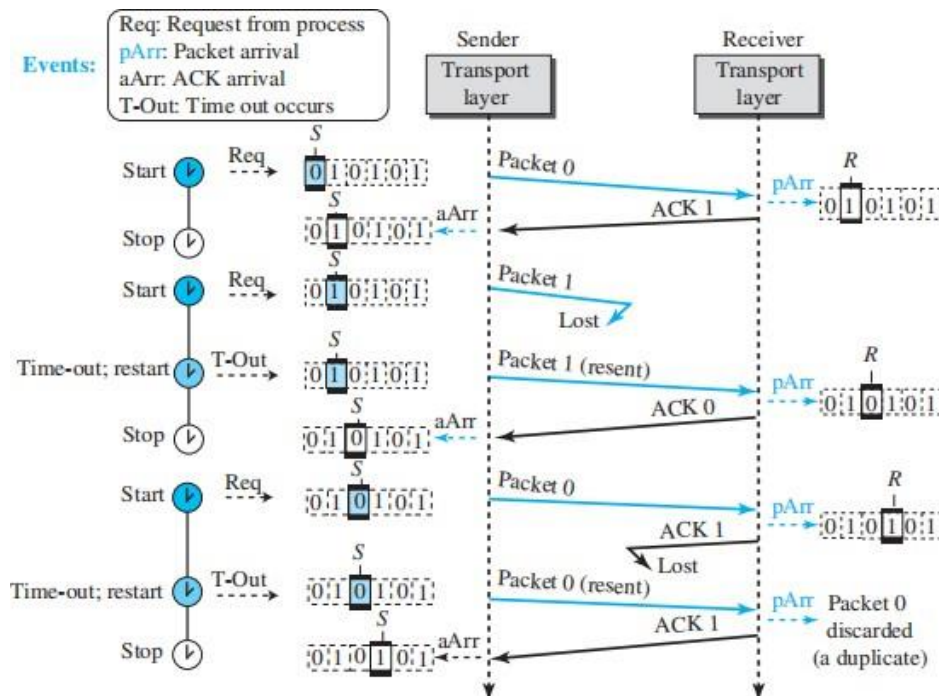


Figure shows an example of the Stop-and-Wait protocol. Packet 0 is sent and acknowledged. Packet 1 is lost and resent after the time-out. The resent packet 1 is acknowledged and the timer stops. Packet 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the packet or the acknowledgment is lost, so after the time-out, it resends packet 0, which is acknowledged.

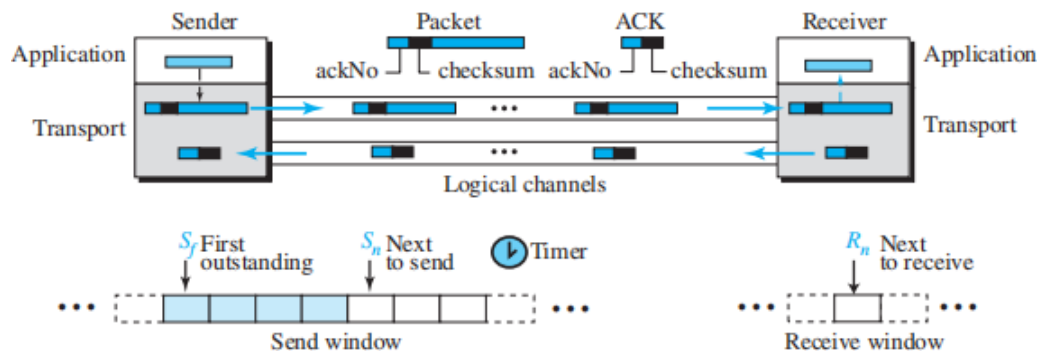


Pipelining

In networking and in other areas, a task is often begun before the previous task has ended. This is known as **pipelining**. There is no pipelining in the Stop-and-Wait protocol because a sender must wait for a packet to reach the destination and be acknowledged before the next packet can be sent.

Go-Back-N Protocol (GBN)

The key to Go-back- N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the acknowledgments arrive. Figure shows the outline of the protocol. Note that several data packets and acknowledgments can be in the channel at the same time.



Sequence Numbers

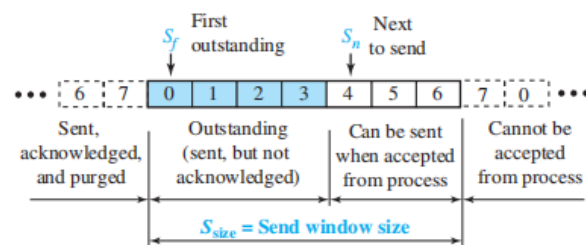
The sequence numbers are modulo $2m$, where m is the size of the sequence number field in bits.

Acknowledgment Numbers

An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected. For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

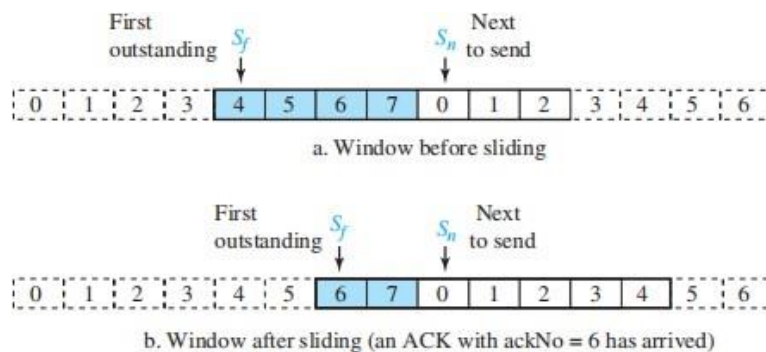
Send Window

The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent. In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent. The maximum size of the window is $2m - 1$.



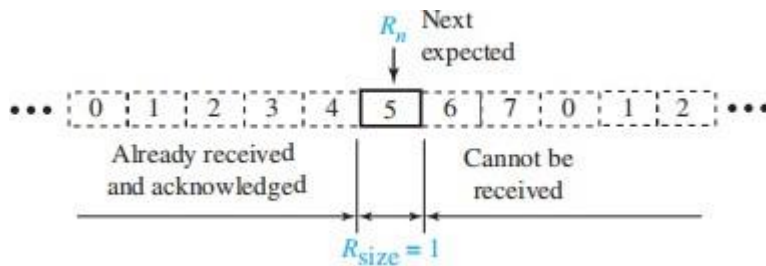
The send window at any time divides sequence numbers into four regions. The first region includes acknowledged packets, which the sender no longer tracks. The second region contains outstanding packets that have been sent but have an unknown status. The third region defines sequence numbers for packets that can be sent but for which data hasn't been received from the application layer. The fourth region consists of sequence numbers that cannot be used until the window slides.

The window itself is an abstraction; three variables define its size and location at any time. We call these variables S_f (send window, the first outstanding packet), S_n (send window, the next packet to be sent), and S_{size} (send window, size). The variable S_f defines the sequence number of the first (oldest) outstanding packet. The variable S_n holds the sequence number that will be assigned to the next packet to be sent. Finally, the variable S_{size} defines the size of the window, which is fixed in our protocol.



Receive Window

The receive window ensures correct packet reception and acknowledgment. In Go-Back-N, its size is always 1, as the receiver expects a specific packet. Out-of-order packets are discarded and must be resent. Only packets matching the expected sequence number, R_n , are accepted and acknowledged. The window slides by one slot upon receiving the correct packet, with R_n updated as $(R_n + 1) \text{ modulo } 2m$.



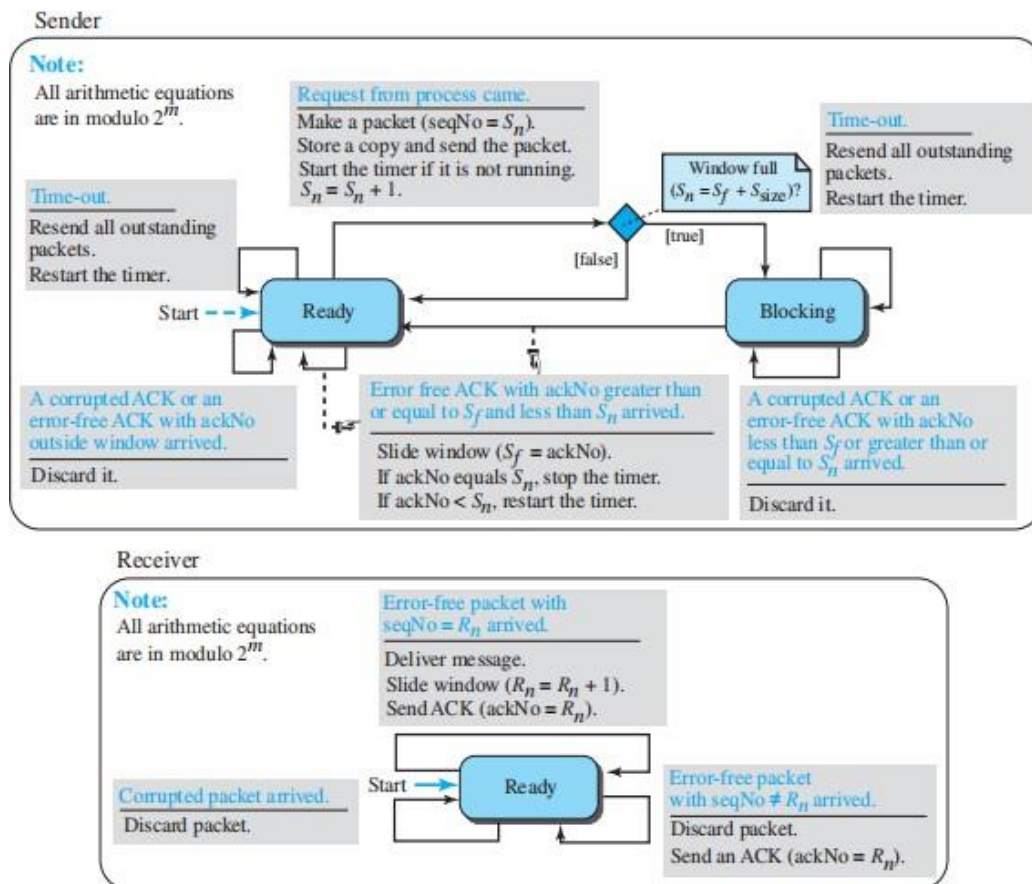
Timers

Although there can be a timer for each packet that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding packet always expires first. We resend all outstanding packets when this timer expires.

Resending packets

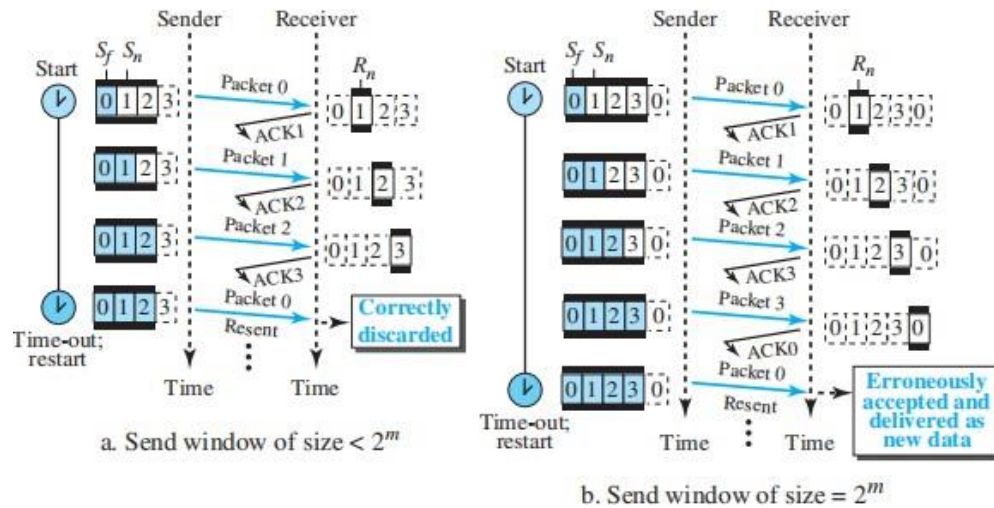
When the timer expires, the sender resends all outstanding packets. For example, suppose the sender has already sent packet 6 ($S_n = 7$), but the only timer expires. If $S_f = 3$, this means that packets 3, 4, 5, and 6 have not been acknowledged; the sender goes back and resends packets 3, 4, 5, and 6. That is why the protocol is called *Go-Back-N*. On a time-out, the machine goes back N locations and resends all packets.

FSMs



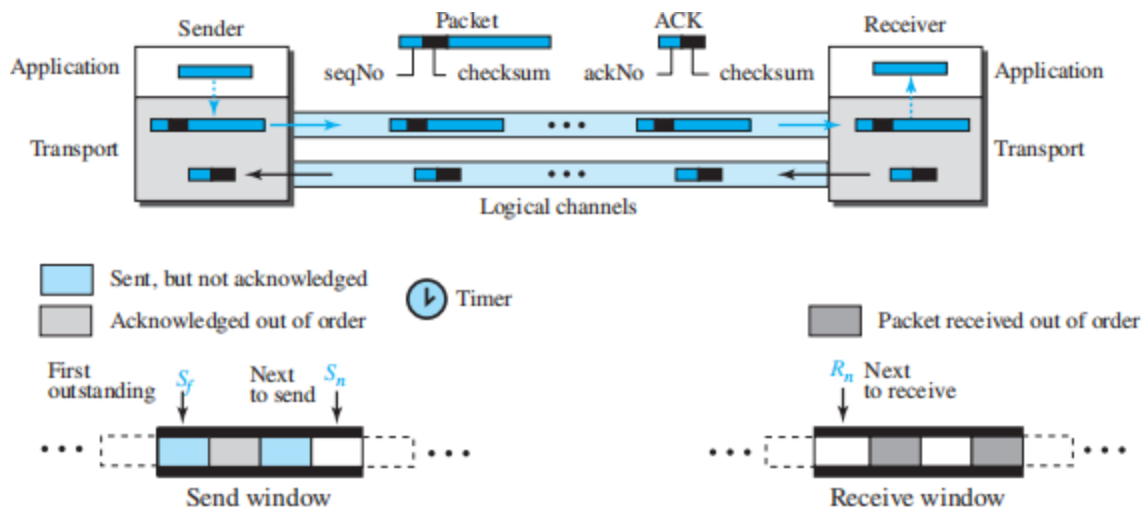
Send Window Size

We can now show why the size of the send window must be less than $2m$. As an example, we choose $m = 2$, which means the size of the window can be $2m - 1$, or 3.



Selective-Repeat Protocol

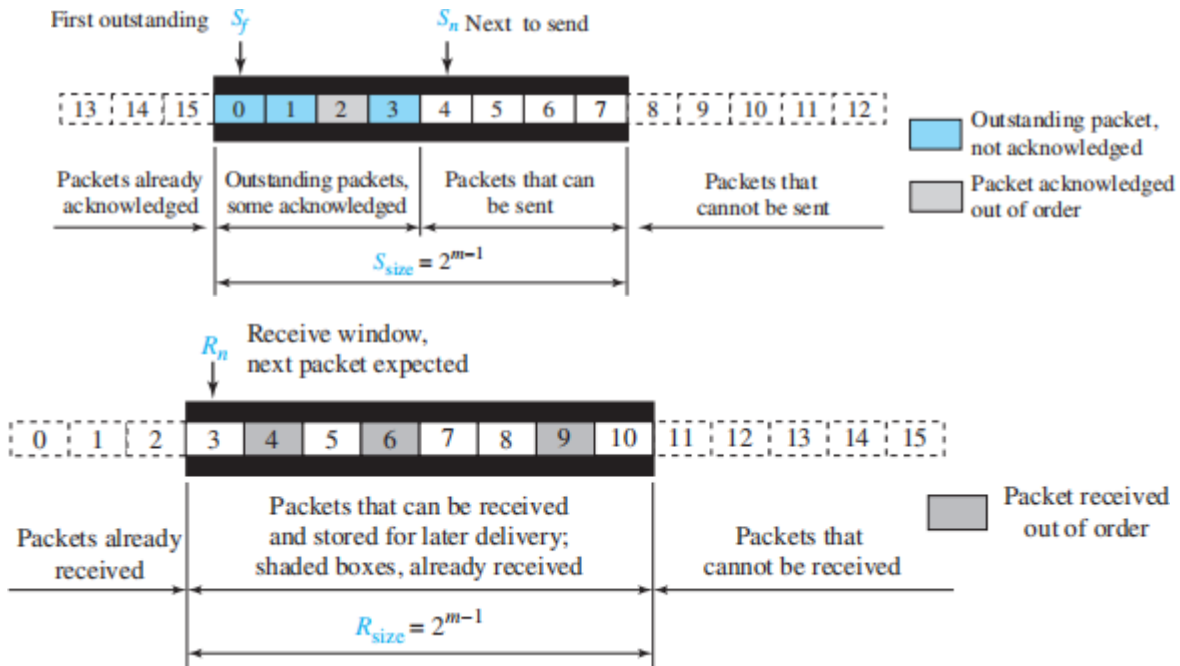
Selective-Repeat (SR) protocol, as the name implies, resends only selective packets, those that are actually lost.



Windows

The Selective-Repeat protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-N. First, the maximum size of the send window is much smaller; it is 2^{m-1} . Second, the receive window is the same size as the send window.

The send window maximum size can be 2^{m-1} . For example, if $m = 4$, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8 (it is 15 in the Go-Back-N Protocol).



Timer

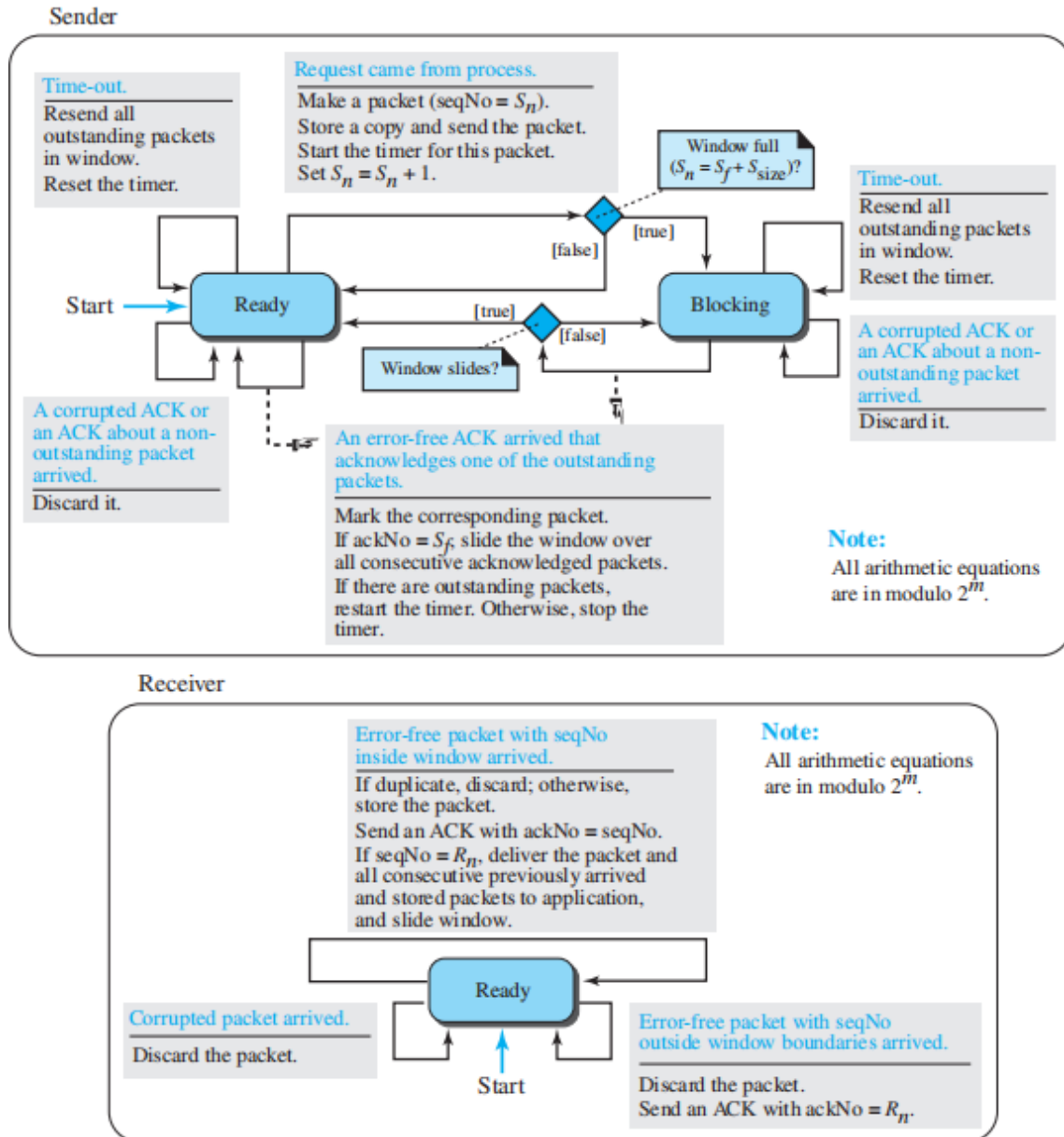
Theoretically, Selective-Repeat uses one timer for each outstanding packet. When a timer expires, only the corresponding packet is resent.

Acknowledgments

There is yet another difference between the two protocols. In GBN an ackNo is cumulative; it defines the sequence number of the next packet expected, confirming that all previous packets have been received safe and sound. The semantics of acknowledgment is different in SR. In SR, an ackNo defines the sequence number of a single packet that is received safe and sound; there is no feedback for any other.

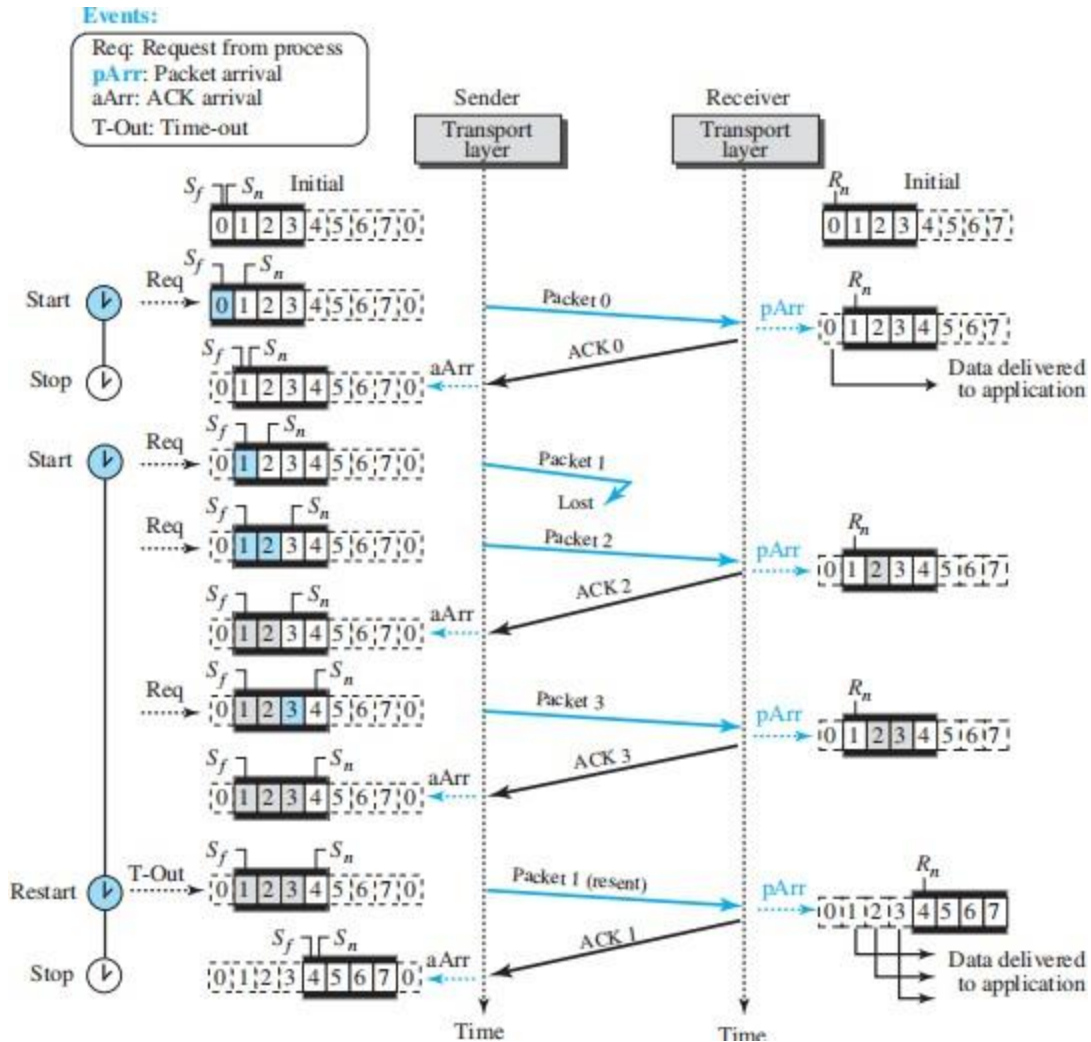
FSMs

Figure shows the FSMs for the Selective-Repeat protocol.



Example

In this example packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure shows the situation.

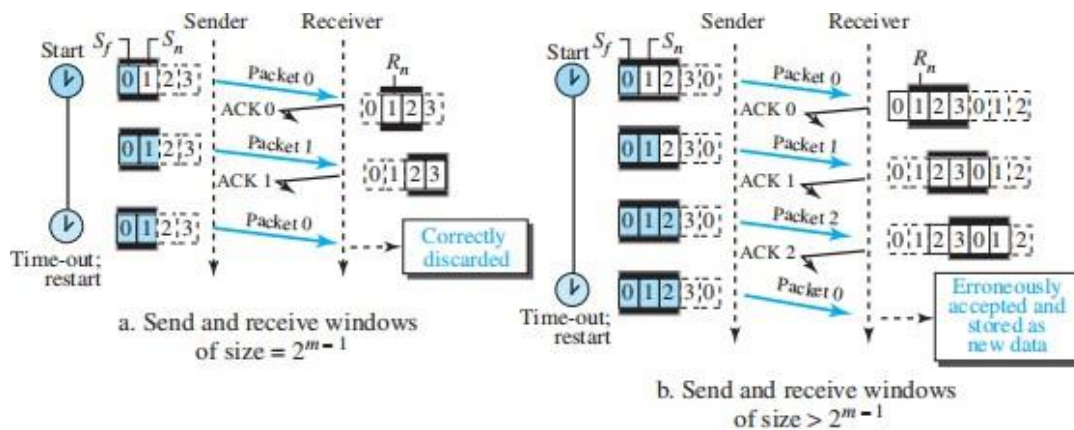


Window Sizes

We can now show why the size of the sender and receiver windows can be at most one-half of 2^m . For an example, we choose $m = 2$, which means the size of the window is $2^{m/2}$ or $2^{(m-1)} = 2$. Figure compares a window size of 2 with a window size of 3.

If the size of the window is 2 and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window). When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of packet 0. However, this time, the window of the receiver expects to receive packet 0

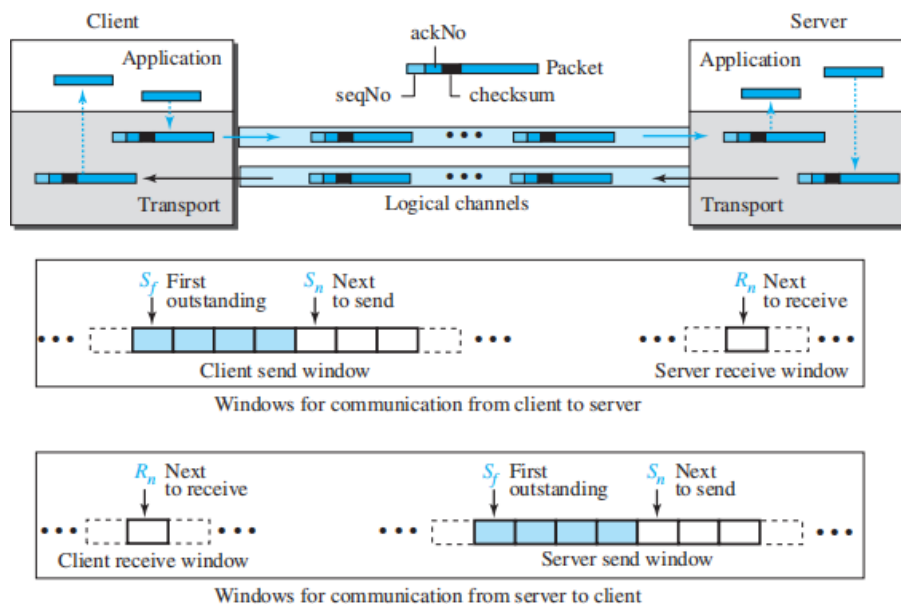
(0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle. This is clearly an error.



Bidirectional Protocols: Piggybacking

The four protocols discussed are all unidirectional: data packets flow in only one direction and acknowledgments travel in the other direction. In real life, data packets are normally flowing in both directions: from client to server and from server to client. This means that acknowledgments also need to flow in both directions. A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a packet is carrying data from A to B, it can also carry acknowledgment feedback about arrived packets from B; when a packet is carrying data from B to A, it can also carry acknowledgment feedback about the arrived packets from A.

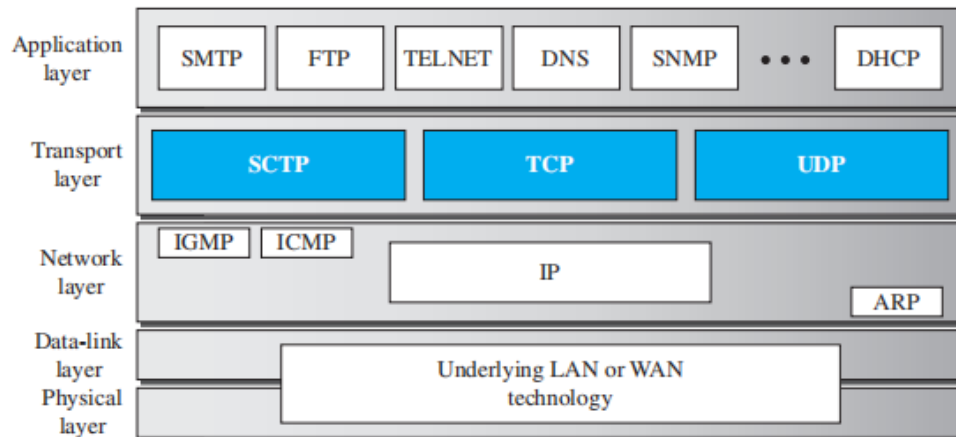
Figure shows the layout for the GBN protocol implemented bidirectionally using piggybacking. The client and server each use two independent windows: send and receive.



Transport-Layer Protocols

INTRODUCTION

The transport layer in the TCP/IP suite is located between the application layer and the network layer. It provides services to the application layer and receives services from the network layer. The transport layer acts as a liaison between a client program and a server program, a process-to-process connection. The transport layer is the heart of the TCP/IP protocol suite; it is the end-to-end logical vehicle for transferring data from one point to another in the Internet.



Services

Each protocol provides a different type of service and should be used appropriately.

UDP

UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

TCP

TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

SCTP

SCTP is a new transport-layer protocol that combines the features of UDP and TCP.

Port Numbers

One of the responsibilities of transport layer is to create a process-to-process communication; these protocols use port numbers to accomplish this. Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and demultiplexing at this layer, just as IP addresses do at the network layer. Table gives some common port numbers for all three protocols.

Some well-known ports used with UDP and TCP

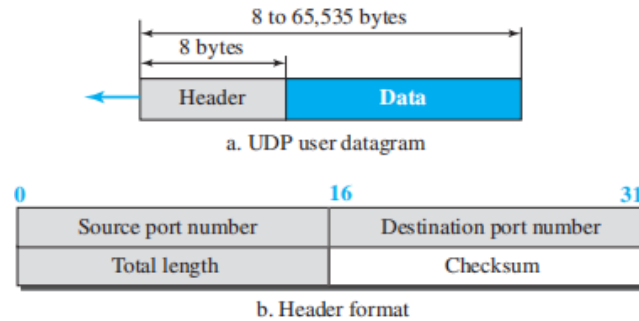
Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name Service
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	HyperText Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

USER DATAGRAM PROTOCOL

The **User Datagram Protocol (UDP)** is a connectionless, unreliable transport protocol. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP.

User Datagram

UDP packets, called **user datagrams**, have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits). Figure shows the format of a user datagram. The first two fields define the source and destination port numbers. The third field defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes. The last field can carry the optional checksum.



UDP Services

Process-to-Process Communication

UDP provides process-to-process communication using **socket addresses**, a combination of IP addresses and port numbers.

Connectionless Services

UDP provides a *connectionless service*. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. There is no connection establishment and no connection termination. This means that each user datagram can travel on a different path.

Flow Control

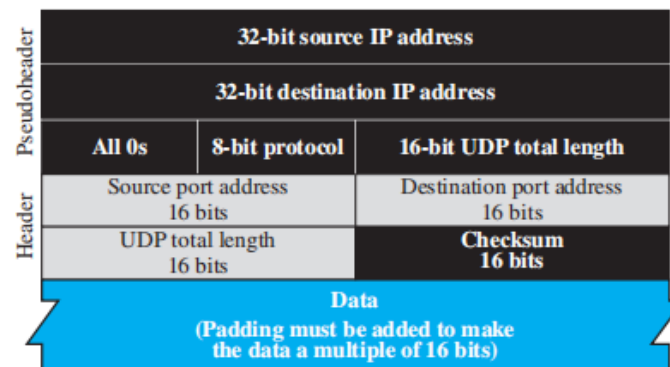
UDP is a very simple protocol. There is no *flow control*, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

Error Control

There is no *error control* mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.

Checksum

UDP checksum calculation includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer. The *pseudoheader* is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.



If the checksum does not include the pseudoheader, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host. The protocol field is added to ensure that the packet belongs to UDP, and not to TCP. The value of the protocol field for UDP is 17. If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol.

Congestion Control

Since UDP is a connectionless protocol, it does not provide congestion control. UDP assumes that the packets sent are small and sporadic and cannot create congestion in the network.

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.

Queuing

In UDP, queues are associated with ports. At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.

Multiplexing and Demultiplexing

In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexes and demultiplexes.

UDP Applications

Although UDP meets almost none of the criteria we mentioned earlier for a reliable transport-layer protocol, UDP is preferable for some applications. The reason is that some services may have some side effects that are either unacceptable or not preferable.

UDP Features

1. ***Connectionless Service*** : UDP is connectionless, with each packet independent of others. This can be an advantage or disadvantage depending on the application's needs. It's advantageous when a client sends a short request and receives a short response, as both can fit in a single datagram. In such cases, avoiding the overhead of establishing and closing a connection is beneficial. A connection-oriented service requires at least 9 packets exchanged between client and server, while a connectionless service only needs 2. The connectionless service reduces delay, making it preferable when minimizing delay is important.
2. ***Lack of Error Control***: UDP lacks error control and provides an unreliable service, while most applications expect reliability from a transport-layer protocol. Although reliable service is preferred, it may have drawbacks for certain applications. If a message part is lost or corrupted, it must be resent, causing delays in delivery to the application layer.

This leads to uneven delays between different parts of the message. While some applications may not notice these delays, they can be problematic for others.

3. **Lack of Congestion Control:** UDP does not provide congestion control. However, UDP does not create additional traffic in an error-prone network. TCP may resend a packet several times and thus contribute to the creation of congestion or worsen a congested situation. Therefore, in some cases, lack of error control in UDP can be considered an advantage when congestion is a big issue.

Typical Applications

The following shows some typical applications that can benefit more from the services of UDP than from those of TCP.

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data.
- UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP).
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.

TRANSMISSION CONTROL PROTOCOL

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitly defines connection establishment, data transfer, and connection tear down phases to provide a connection-oriented service. TCP uses a combination of GBN and SR protocols to provide reliability. To achieve this goal, TCP uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.

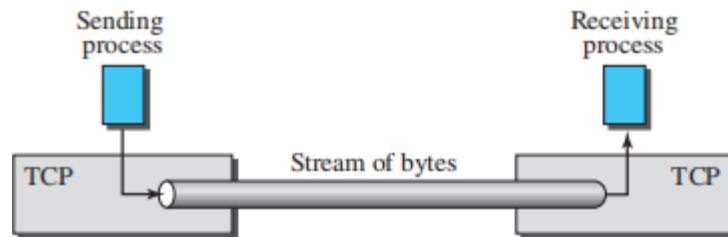
TCP Services

Process-to-Process Communication

As with UDP, TCP provides process-to-process communication using port numbers.

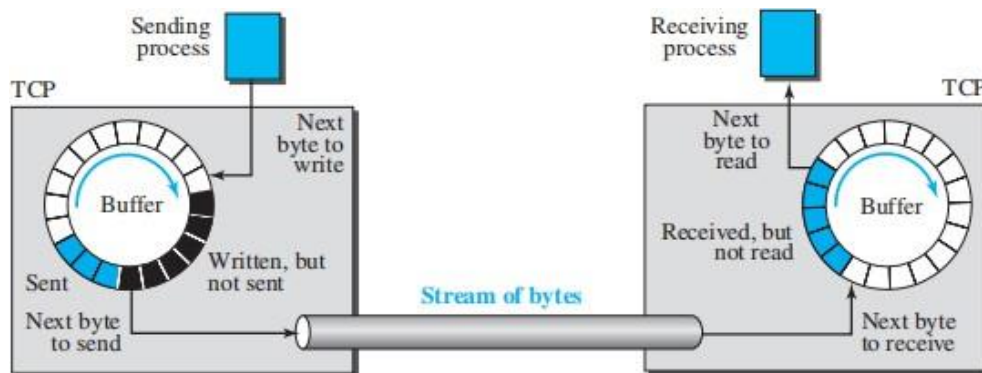
Stream Delivery Service

TCP, unlike UDP, is a stream-oriented protocol. TCP, allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary “tube” that carries their bytes across the Internet. This imaginary environment is depicted in Figure. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.



Sending and Receiving Buffers

Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction.



These buffers are essential for flow control and error control in TCP. A buffer can be implemented as a circular array of 1-byte locations, as shown in the figure, though typically they contain hundreds or thousands of bytes. The figure illustrates data movement in one direction, showing three sections in the sender's buffer:

1. **Empty chambers** (white) that can be filled by the sending process.
2. **Sent but unacknowledged bytes** (colored), which remain in the buffer until acknowledged.
3. **Bytes ready to be sent** (shaded), though TCP may only send part of this due to receiver slowness or network congestion.

Once acknowledged, the sent bytes' chambers are recycled, making them available for reuse, which is why a circular buffer is used.

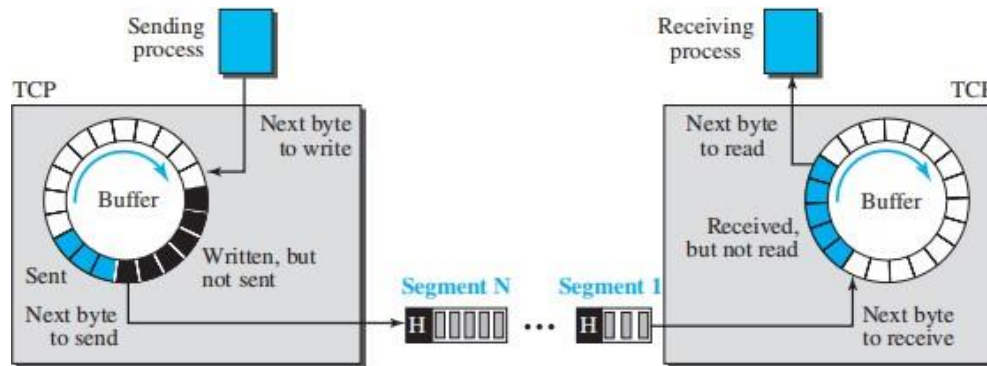
At the receiver, the buffer is simpler, divided into two areas:

1. **Empty chambers** (white), ready to be filled with received bytes.
2. **Filled chambers** (colored), containing received bytes.

Segments shown in the figure vary in size, but real segments typically carry hundreds or thousands of bytes.

Segments

Although buffering handles the speed difference between the producing and consuming processes, one more step is needed before sending data. The network layer transmits data in packets, not as a byte stream. TCP groups bytes into **segments**, adds a control header, and passes them to the network layer for transmission, where they are encapsulated in IP datagrams. This process is transparent to the receiving application, and TCP handles issues like out-of-order, lost, or corrupted segments without the receiving process being aware. Figure shows how segments are created from bytes in the buffers.



Full-Duplex Communication

TCP offers *full-duplex service*, where data can flow in both directions at the same time. Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.

Multiplexing and Demultiplexing

Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes.

Connection-Oriented Service

TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:

1. The two TCP's establish a logical connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

TCP Features

Numbering System Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields, called the *sequence number* and the *acknowledgment number*. These two fields refer to a byte number and not a segment number.

1. **Byte Number-** TCP numbers all data bytes transmitted in a connection, with independent numbering in each direction. The numbering doesn't start from 0 but from an arbitrary number between 0 and $2^{32} - 1$. For example, if the starting number is 1057 and 6000 bytes are to be sent, the bytes will be numbered from 1057 to 7056. This numbering is used for flow and error control.
2. **Sequence Number** -After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number, in each direction, is defined as follows:
 1. The sequence number of the first segment is the ISN (initial sequence number), which is a random number.
 2. The sequence number of any other segment is the sequence number of the previous segment plus the number of bytes (real or imaginary) carried by the previous segment. Later, we show that some control segments are thought of as carrying one imaginary byte.
3. **Acknowledgment Number** - In TCP's full duplex communication, both parties can send and receive data simultaneously. Each party numbers the bytes, typically starting with different numbers. The sequence number in each direction indicates the first byte in the segment, while the acknowledgment number confirms receipt of bytes and specifies the next expected byte. The acknowledgment number is cumulative, meaning if it's 5643, all bytes up to 5642 have been received, though the first byte may not be 0.

Segment

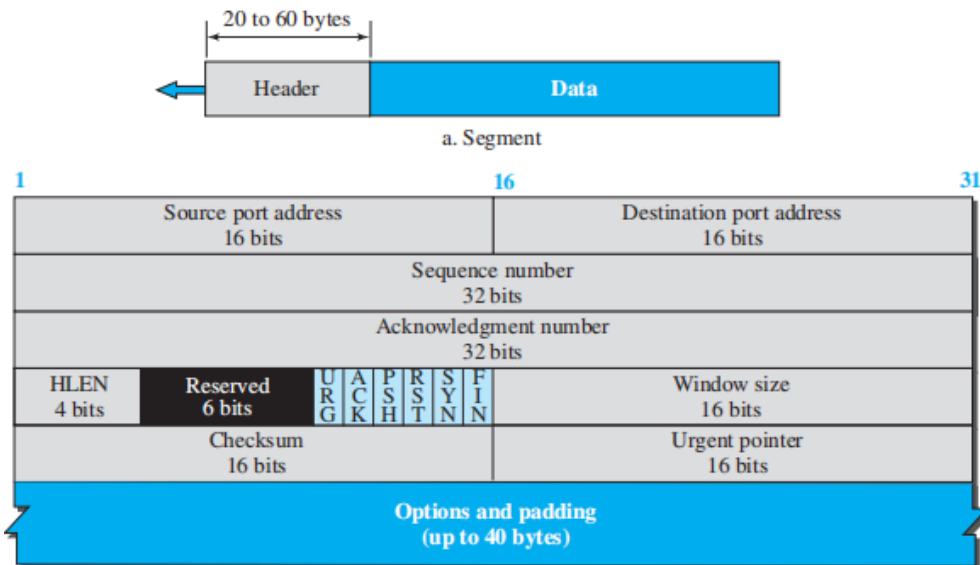
A packet in TCP is called a *segment*.

Format

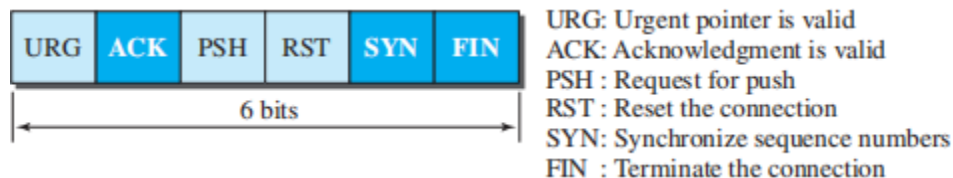
The format of a segment is shown in Figure . The segment consists of a header of 20 to 60 bytes, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

- **Source port address.** This is a 16-bit field that defines the port number of the application program in the host that is sending the segment.
- **Destination port address.** This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment.
- **Sequence number.** This 32-bit field defines the number assigned to the first byte of data contained in this segment. TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment

each party uses a random number generator to create an **initial sequence number** (ISN), which is usually different in each direction.

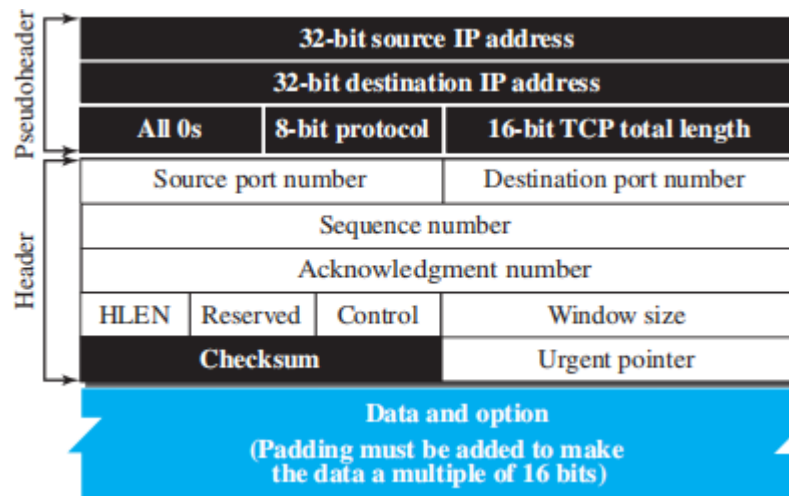


- **Acknowledgment number.** This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it returns $x + 1$ as the acknowledgment number. Acknowledgment and data can be piggybacked together.
- **Header length.** This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).
- **Control.** This field defines 6 different control bits or flags, as shown in Figure. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.



- **Window size.** This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (*rwnd*) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

- **Checksum.** This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment. For the TCP pseudoheader, the value for the protocol field is 6. See Figure



- **Urgent pointer.** This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.
- **Options.** There can be up to 40 bytes of optional information in the TCP header.

Encapsulation

A TCP segment encapsulates the data received from the application layer. The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer.

A TCP Connection

TCP is connection-oriented. A connection-oriented transport protocol establishes a logical path between the source and destination. All of the segments belonging to a message are then sent over this logical path. Using a single logical pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.

TCP is connection-oriented, but its connection is logical, not physical. TCP operates at a higher layer, using IP (a connectionless protocol) to send individual segments. If a segment is lost or corrupted, TCP retransmits it, whereas IP itself doesn't handle retransmissions.

If a segment arrives out of order, TCP holds it until the missing segments arrive; IP is unaware of this reordering.

In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

Connection Establishment

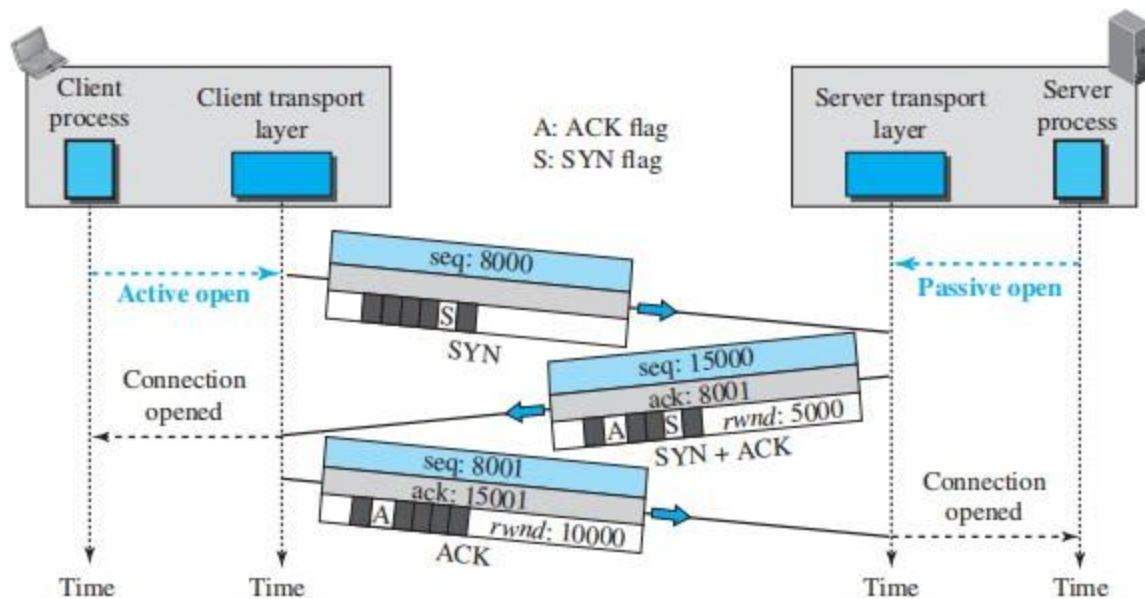
TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

Three-Way Handshaking

The connection establishment in TCP is called **three-way handshaking**. For example, an application program, called the *client*, wants to make a connection with another application program, called the *server*, using TCP as the transport-layer protocol.

The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a *passive open*. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an *active open*. A client that wishes to connect to an open server tells its TCP to connect to a particular server. TCP can now start the three-way handshaking process, as shown in Figure.



The three steps in this phase are as follows.

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. This sequence number is called the *initial sequence number (ISN)*. This segment does not contain an acknowledgment number. The SYN segment is a control segment and carries no data.
2. The server sends the second segment, a SYN + ACK segment with two flag bits set as: SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a

sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. Because the segment contains an acknowledgment, it also needs to define the receive.

3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field.

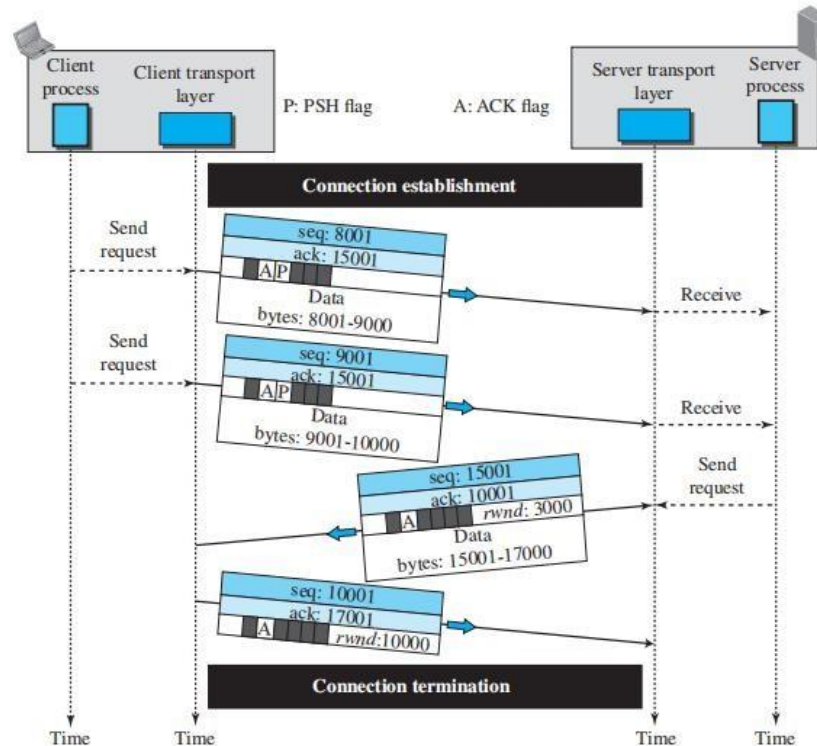
SYN Flooding Attack

The connection establishment procedure in TCP is susceptible to a serious security problem called *SYN flooding attack*. This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams. The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating transfer control block (TCB) tables and setting timers. The TCP server then sends the SYN + ACK segments to the fake clients, which are lost.

When the server waits for the third leg of the handshaking process, however, resources are allocated without being used. If, during this short period of time, the number of SYN segments is large, the server eventually runs out of resources and may be unable to accept connection requests from valid clients. This SYN flooding attack belongs to a group of security attacks known as a *denial-of-service attack*, in which an attacker monopolizes a system with so many service requests that the system overloads and denies service to valid requests.

Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions. The acknowledgment is piggybacked with the data. Figure shows an example.



Pushing Data

TCP uses a buffer at the sending side to store data from the application program, allowing flexibility in segment size selection. The receiving TCP also buffers incoming data, delivering it when convenient. This flexibility enhances TCP efficiency but may not suit applications requiring immediate interaction, where delays in transmission and delivery are unacceptable. In such cases, the sending application can request a push operation. This forces the sending TCP to create and send a segment immediately, setting the push bit (PSH) to ensure the receiving TCP delivers the data without waiting for additional data.

Urgent Data

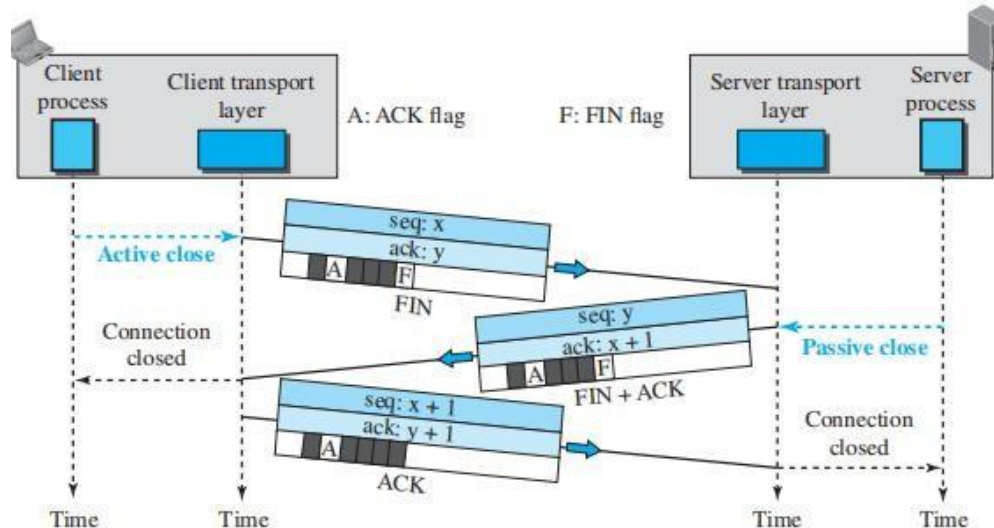
TCP is a stream-oriented protocol, meaning data is presented as a continuous stream of bytes, each with a specific position. However, when an application needs to send urgent data requiring special handling at the receiving end, it can use the URG bit. The sending TCP marks this data as urgent by creating a segment with the urgent data at the beginning and using the urgent pointer to indicate its end. For instance, if the sequence number is 15000 and the urgent pointer is 200, the urgent data covers bytes 15000 to 15200, while the rest of the segment contains non-urgent data. It's important to note that TCP's urgent data is not a priority or out-of-band service; instead, it flags a part of the byte stream for special attention by the receiving application. TCP delivers all bytes in sequence but informs the receiving application of the urgent data's start and end, leaving its handling to the application.

Connection Termination

Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

Three-Way Handshaking

Most implementations today allow *three-way handshaking* for connection termination.



1. In this situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set.
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN + ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction.
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

Half-Close

In TCP, one end can stop sending data while still receiving data. This is called a **half close**. Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin. A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data. The server,

after receiving the data, still needs time for sorting; its outbound direction must remain open. After half-closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server.

Connection Reset

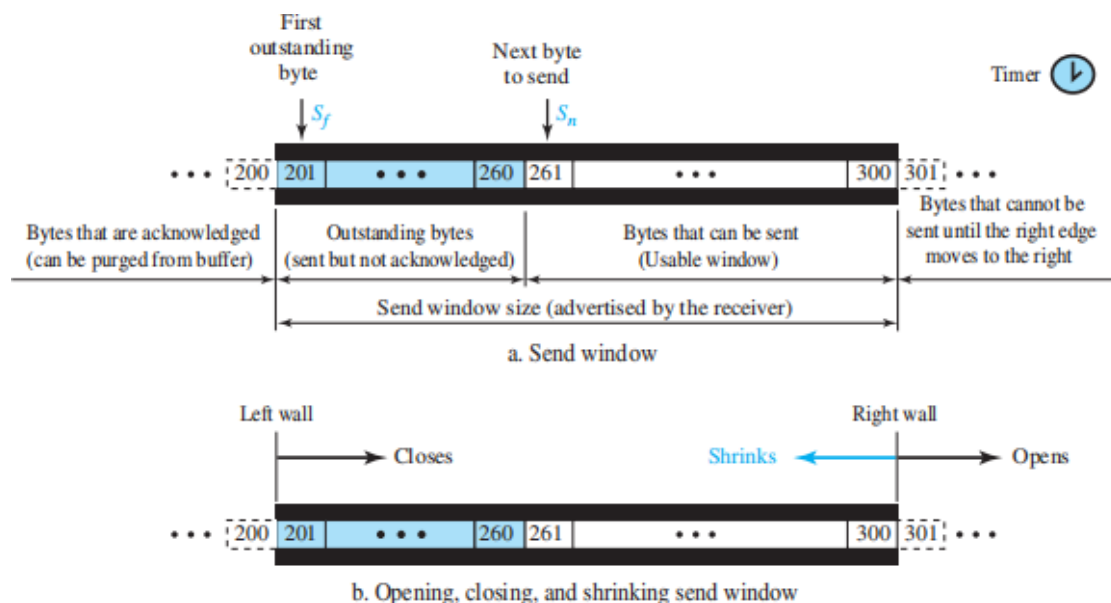
TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection. All of these are done with the RST (reset) flag.

Windows in TCP

TCP uses two windows (send window and receive window) for each direction of data transfer, which means four windows for a bidirectional communication.

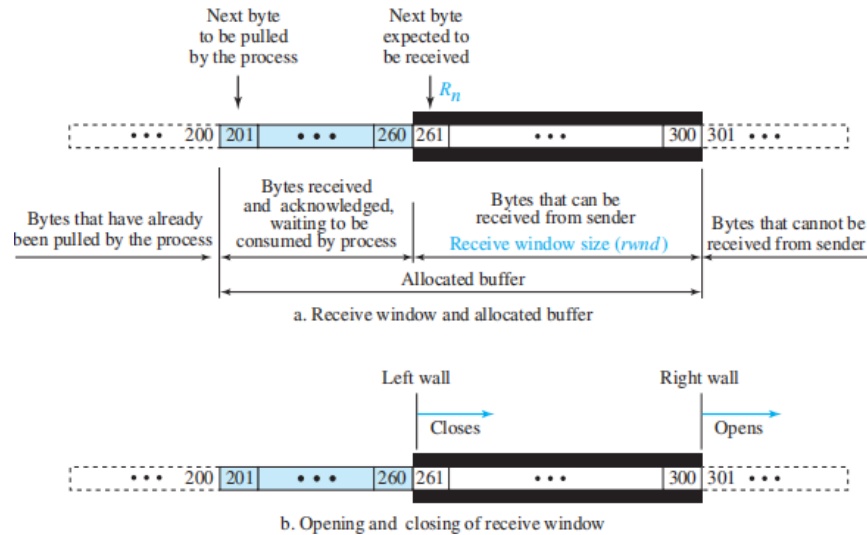
Send Window

Figure shows an example of a send window. The window size is 100 bytes, but later we see that the send window size is dictated by the receiver (flow control) and the congestion in the underlying network (congestion control). The figure shows how a send window *op*



Receive Window

Figure shows an example of a receive window. The window size is 100 bytes. The figure also shows how the receive window opens and closes; in practice, the window should never shrink.



Flow Control

The *flow control* balances the rate a producer creates data with the rate a consumer can use the data. TCP separates flow control from error control.

Opening and Closing Windows

To achieve flow control, TCP forces the sender and the receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established. The receive window closes (moves its left wall to the right) when more bytes arrive from the sender; it opens (moves its right wall to the right) when more bytes are pulled by the process.

Shrinking of Windows

The receive window cannot shrink. The send window, on the other hand, can shrink if the receiver defines a value for $rwnd$ that results in shrinking the window.

Window Shutdown

Shrinking the send window by moving its right wall to the left is strongly discouraged. However, there is one exception: the receiver can temporarily shut down the window by sending a $rwnd$ of 0. This can happen if for some reason the receiver does not want to receive any data from the sender for a while.

Silly Window Syndrome

A serious problem can arise in the sliding window operation when either the sending application program creates data slowly or the receiving application program consumes data slowly, or both. Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation. For example, if TCP sends segments containing only 1 byte of data, it means that a 41-byte datagram (20 bytes of TCP header and 20 bytes of IP header) transfers

only 1 byte of user data. Here the overhead is 41/1, which indicates that we are using the capacity of the network very inefficiently. The inefficiency is even worse after accounting for the data-link layer and physical-layer overhead. This problem is called the ***silly window syndrome***.

Error Control

TCP is a reliable transport-layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

Checksum

Each segment includes a checksum field, which is used to check for a corrupted segment. If a segment is corrupted, as detected by an invalid checksum, the segment is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment.

Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments. Control segments that carry no data, but consume a sequence number, are also acknowledged. ACK segments are never acknowledged.

Acknowledgment Type

In the past, TCP used only one type of acknowledgment: cumulative acknowledgment.

Today, some TCP implementations also use selective acknowledgment.

Cumulative Acknowledgment (ACK)

In TCP, segments are acknowledged cumulatively. The receiver advertises the next byte it expects, ignoring out-of-order segments. This method, called **positive cumulative acknowledgment**, provides no feedback for lost, discarded, or duplicate segments. The 32-bit ACK field in the TCP header is used for this, and it's valid only when the ACK flag is set.

Selective Acknowledgment (SACK)

Many TCP implementations now support **selective acknowledgment (SACK)**, which complements ACK by reporting out-of-order or duplicated byte blocks. Since the TCP header lacks space for this, SACK is implemented as an option at the end of the header.

Generating Acknowledgments

Common rules for generating ACK

1. When end A sends a data segment to end B, it must include (piggyback) an acknowledgment that gives the next sequence number it expects to receive. This rule decreases the number of segments needed and therefore reduces traffic.

2. When the receiver has no data to send and it receives an in-order segment (with expected sequence number) and the previous segment has already been acknowledged, the receiver delays sending an ACK segment until another segment arrives or until a period of time (normally 500 ms) has passed.
3. When a segment arrives with a sequence number that is expected by the receiver, and the previous in-order segment has not been acknowledged, the receiver immediately sends an ACK segment.
4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. This leads to the *fast retransmission* of missing segments.
5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected. This informs the receiver that segments reported missing have been received.
6. If a duplicate segment arrives, the receiver discards the segment, but immediately sends an acknowledgment indicating the next in-order segment expected. This solves some problems when an ACK segment itself is lost.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is sent, it is stored in a queue until it is acknowledged. When the retransmission timer expires or when the sender receives three duplicate ACKs for the first segment in the queue, that segment is retransmitted.

Retransmission after RTO

TCP maintains a retransmission time-out (RTO) for each connection. If the timer expires, the segment at the front of the queue (with the smallest sequence number) is retransmitted, and the timer is reset. The RTO is dynamic, adjusting based on the round-trip time (RTT), which is the time taken for a segment to reach its destination and receive an acknowledgment.

Retransmission after Three Duplicate ACK Segments

To avoid waiting for an RTO, many TCP implementations use **fast retransmission**. If three duplicate ACKs (the original and three identical copies) are received for a segment, TCP immediately retransmits the missing segment without waiting for the RTO.

Out-of-Order Segments

TCP implementations today do not discard out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segments arrive. However, that out-of-

order segments are never delivered to the process. TCP guarantees that data are delivered to the process in order.

Some Scenarios

Normal Operation

In normal bidirectional data transfer, the client sends a segment, and the server sends three. The client delays acknowledgments by 500 ms to check if more segments arrive. If no further segments arrive before the timer expires, the acknowledgment is sent immediately.

Lost Segment

If a segment is lost or corrupted, the receiver stores the out-of-order data in its buffer and sends an acknowledgment for the missing byte. The sender retransmits the lost segment after the RTO timer expires.

Fast Retransmission

If a segment is lost and the receiver sends three duplicate ACKs, the sender immediately retransmits the missing segment without waiting for the RTO timer, following the fast retransmission rule.

Delayed Segment

TCP segments may arrive with delays due to IP's connectionless nature. If a segment is delayed and arrives after its retransmission, it is considered a duplicate and discarded by the receiver.

Duplicate Segment

A duplicate segment occurs when a segment is delayed and the sender resends it, thinking it is lost. The receiver discards any duplicate segment and sends an acknowledgment for the expected sequence number.

Automatically Corrected Lost ACK

In cumulative acknowledgment, if an ACK is lost, the next ACK automatically corrects the loss. The sender continues normally because the lost ACK is not critical as long as the next one arrives.

Lost Acknowledgment Corrected by Resending a Segment

If an acknowledgment is lost and not corrected by the next one, the RTO timer triggers a retransmission of the segment. The receiver discards the duplicate segment and resends the last ACK to inform the sender.

Deadlock Created by Lost Acknowledgment

A deadlock can occur if an acknowledgment with a zero window size is lost. The sender waits for the window to reopen, while the receiver believes the sender is waiting for data. To prevent deadlock, a persistence timer is used.

TCP Congestion Control

TCP uses different policies to handle the congestion in the network.

Congestion Window

To control the number of segments to transmit, TCP uses another variable called a *congestion window*, *cwnd*, whose size is controlled by the congestion situation in the network. The *cwnd* variable and the *rwnd* variable together define the size of the send window in TCP. The first is related to the congestion in the middle (network); the second is related to the congestion at the end. The actual size of the window is the minimum of these two.

$$\text{Actual window size} = \text{minimum} (rwnd, cwnd)$$

Congestion Detection

TCP detects network congestion through two main events: time-outs and the receipt of three duplicate ACKs. A time-out occurs when the sender does not receive an ACK for a segment or group of segments before the timer expires, signaling the likelihood of severe congestion and possible segment loss. On the other hand, receiving three duplicate ACKs (four identical ACKs) indicates that one segment is missing, but others have been received, suggesting mild congestion or network recovery. While early TCP versions like **Tahoe** treated both events the same, later versions like **Reno** distinguish between the two, with time-outs indicating stronger congestion than duplicate ACKs. TCP relies on ACKs as the only feedback to detect congestion, where missing or delayed ACKs serve as indicators of network conditions.

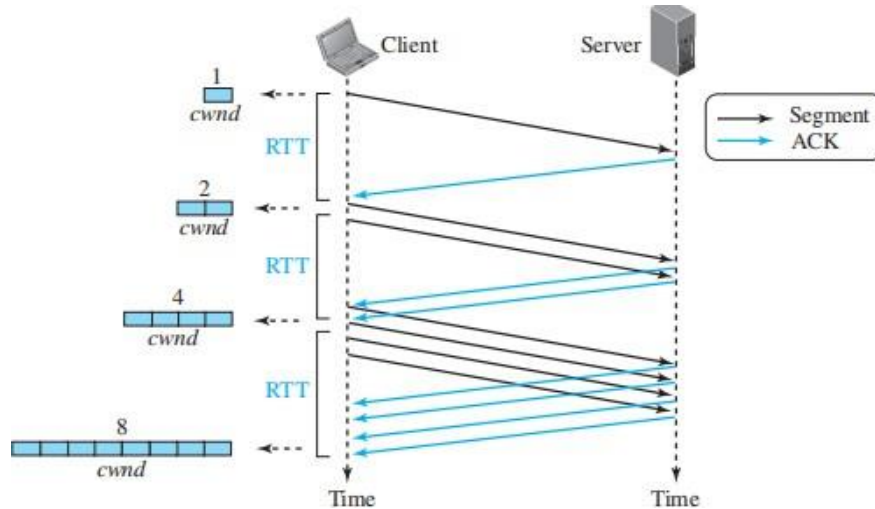
Congestion Policies

TCP's general policy for handling congestion is based on three algorithms: slow start, congestion avoidance, and fast recovery.

Slow Start: Exponential Increase

The slow-start algorithm begins with the congestion window (*cwnd*) set to one maximum segment size (MSS) and increases by one MSS for each received acknowledgment. The MSS is negotiated during connection establishment. Despite the name, the algorithm grows exponentially. Initially, the sender transmits one segment, and upon receiving the ACK, the *cwnd* increases by 1, allowing the sender to transmit two segments. Each acknowledgment further increases *cwnd*, doubling the number of segments the sender can transmit, resulting in rapid growth as long as no congestion is detected. The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined as follows.

$$\text{If an ACK arrives, } cwnd = cwnd + 1.$$



If we look at the size of the *cwnd* in terms of round-trip times (RTTs), we find that the growth rate is exponential in terms of each round trip time, which is a very aggressive approach:

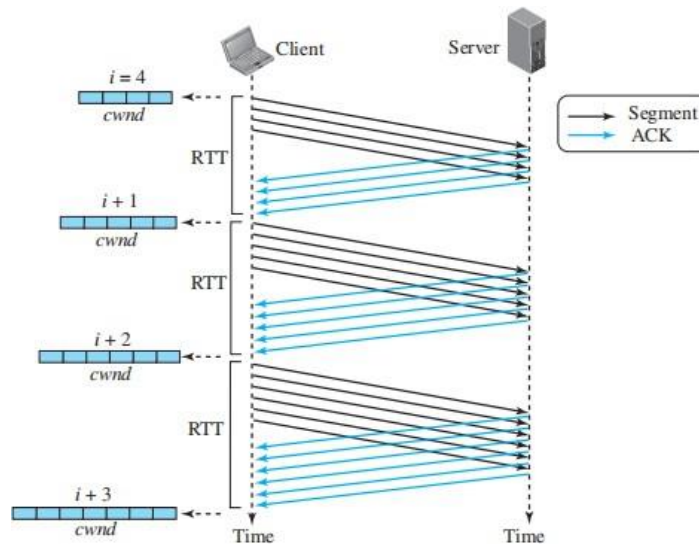
Start	→	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	→	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	→	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	→	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

A slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named *ssthresh* (slow-start threshold). When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.

Congestion Avoidance: Additive Increase

The slow-start algorithm in TCP increases the size of the congestion window (*cwnd*) exponentially, which can lead to congestion. To mitigate this, TCP employs the congestion avoidance algorithm, which increases *cwnd* additively rather than exponentially. When *cwnd* reaches the slow-start threshold (i), the slow-start phase ends, and the additive phase begins.

In this algorithm, for each set of acknowledged segments (the entire “window”), *cwnd* is incremented by one. A window represents the number of segments sent during a round-trip time (RTT).



For example, if the sender starts with $cwnd = 4$, it can send four segments. Upon receiving four ACKs, one segment slot opens up, increasing $cwnd$ to 5. After sending five segments and receiving five acknowledgments, $cwnd$ increases to 6, and so on.

The congestion window can be expressed as:

If an ACK arrives, $cwnd = cwnd + (1/cwnd)$.

The window increases by $(1/cwnd)$ portion of the Maximum Segment Size (MSS) in bytes.

Thus, all segments in the previous window must be acknowledged to increase $cwnd$ by 1 MSS byte. This results in a linear growth rate of $cwnd$ with each round-trip time (RTT), making it a more conservative approach than slow-start.

Start	→	$cwnd = i$
After 1 RTT	→	$cwnd = i + 1$
After 2 RTT	→	$cwnd = i + 2$
After 3 RTT	→	$cwnd = i + 3$

Fast Recovery

The **fast-recovery** algorithm is optional in TCP. The old version of TCP did not use it, but the new versions try to use it. It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network. Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm). We can say

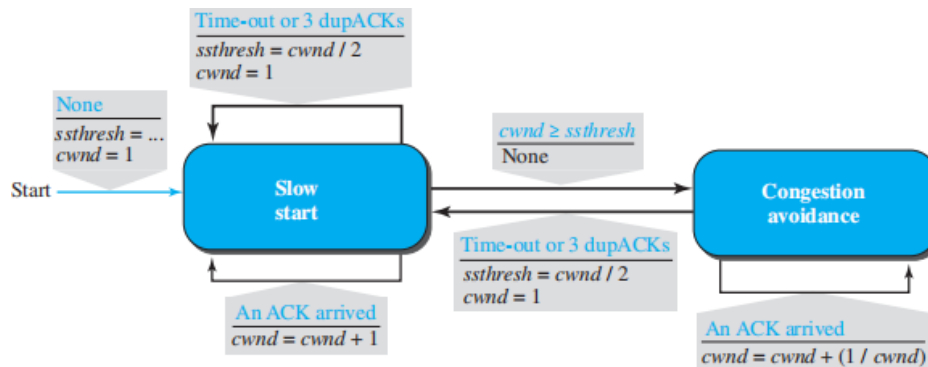
If a duplicate ACK arrives, $cwnd == cwnd + (1 / cwnd)$

Policy Transition

We discussed three congestion policies in TCP. Now the question is when each of these policies is used and when TCP moves from one policy to another. To answer these questions, we need to refer to three versions of TCP: Tahoe TCP, Reno TCP, and New Reno TCP.

Tahoe TCP

The early TCP, known as *Tahoe TCP*, used only two different algorithms in their congestion policy: *slow start* and *congestion avoidance*.



Tahoe TCP treats the two signs used for congestion detection, time-out and three duplicate ACKs, in the same way. In this version, when the connection is established, TCP starts the slow-start algorithm and sets the *ssthresh* variable to a pre-agreed value (normally a multiple of MSS) and the *cwnd* to 1 MSS.

If congestion is detected (occurrence of time-out or arrival of three duplicate ACKs), TCP immediately interrupts this aggressive growth and restarts a new slow start algorithm by limiting the threshold to half of the current *cwnd* and resetting the congestion window to 1.

If no congestion is detected while reaching the threshold, TCP learns that the ceiling of its ambition is reached; it should not continue at this speed. It moves to the congestion avoidance state and continues in that state.

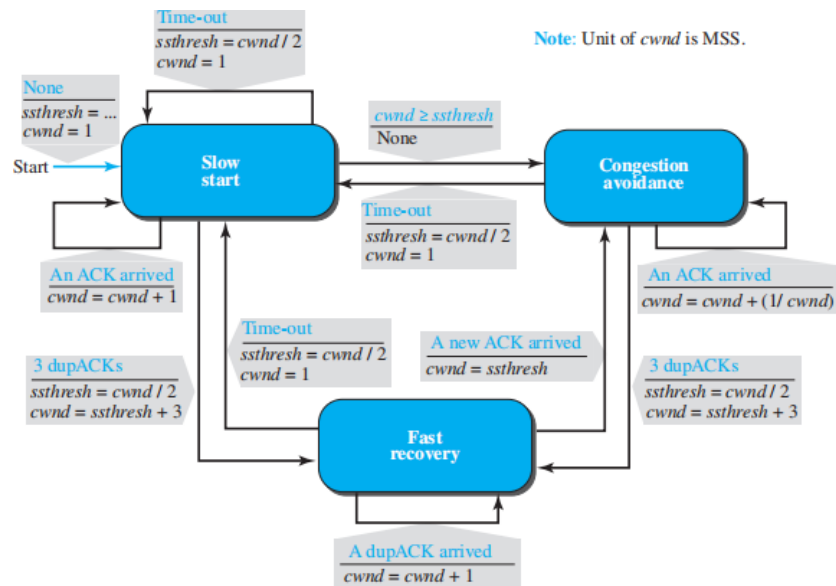
In the congestion-avoidance state, the size of the congestion window is increased by 1 each time a number of ACKs equal to the current size of the window has been received.

Reno TCP

A newer version of TCP, called *Reno TCP*, added a new state to the congestion-control FSM, called the fast-recovery state. This version treated the two signals of congestion, time-out and the arrival of three duplicate ACKs, differently. In this version, if a time-out occurs, TCP moves to the slow-start state (or starts a new round if it is already in this state); on the other hand, if three

duplicate ACKs arrive, TCP moves to the fast-recovery state and remains there as long as more duplicate ACKs arrive.

When TCP enters the fast-recovery state, three major events may occur. If duplicate ACKs continue to arrive, TCP stays in this state, but the *cwnd* grows exponentially. If a time-out occurs, TCP assumes that there is real congestion in the network and moves to the slow-start state. If a new (non duplicate) ACK arrives, TCP moves to the congestion-avoidance state, but deflates the size of the *cwnd* to the *ssthresh* value, as though the three duplicate ACKs have not occurred, and transition is from the slow-start state to the congestion-avoidance state.



NewReno TCP

A later version of TCP, called *NewReno TCP*, made an extra optimization on the Reno TCP. In this version, TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive. When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives. If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost. However, if the ACK number defines a position between the retransmitted segment and the end of the window, it is possible that the segment defined by the ACK is also lost. NewReno TCP retransmits this segment to avoid receiving more and more duplicate ACKs for it.

Additive Increase, Multiplicative Decrease

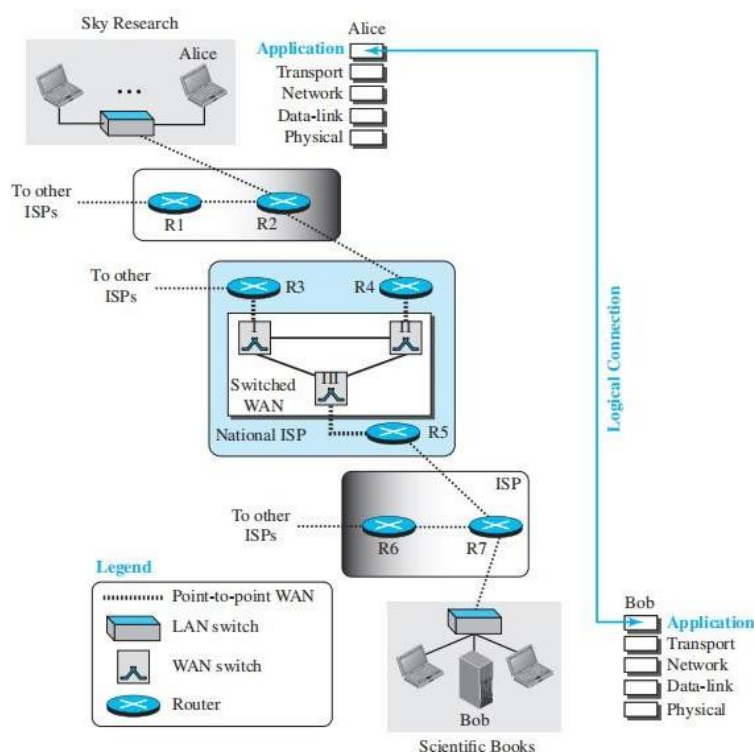
Out of the three versions of TCP, the Reno version is most common today. It has been observed that, in this version, most of the time the congestion is detected and taken care of by observing the three duplicate ACKs. Even if there are some time-out events, TCP recovers from them by aggressive exponential growth.

Module 5

Introduction to Application layer

Introduction

The application layer provides services to the user. Communication is provided using a logical connection, which means that the two application layers assume that there is an imaginary direct connection through which they can send and receive messages. Figure shows the idea behind this logical connection.



Providing Services

Before the Internet, communication networks like the telephone network were designed for specific services, such as voice communication. Over time, users adapted these networks to provide additional services, like fax, by adding extra hardware.

The Internet, similarly designed to serve global users, is more adaptable due to the layered structure of the TCP/IP protocol suite. Each layer in this suite can add or replace protocols, as

long as they work with the protocols in the layers below. This flexibility supports the evolving nature of the Internet.

The application layer, at the top, differs because it only receives services from the transport layer without needing to support other layers. This makes it easy to add or remove application protocols as long as they can work with the transport layer. This flexibility allows the Internet to continually expand with new application protocols, which has been a constant feature of its growth since its inception.

Standard and Nonstandard Protocols

For smooth Internet operation, the protocols in the first four layers of the TCP/IP suite need to be standardized and documented. These standard protocols are typically part of operating systems like Windows or UNIX. However, the application-layer protocols can be both standard and nonstandard for added flexibility.

Standard Application-Layer Protocols-Standard application-layer protocols are well-documented and standardized by Internet authorities. They are widely used in daily Internet interactions.

Nonstandard Application-Layer Protocols-Programmers can also create nonstandard (or proprietary) application-layer programs by developing two programs that provide services through the transport layer.

Application-Layer Paradigms

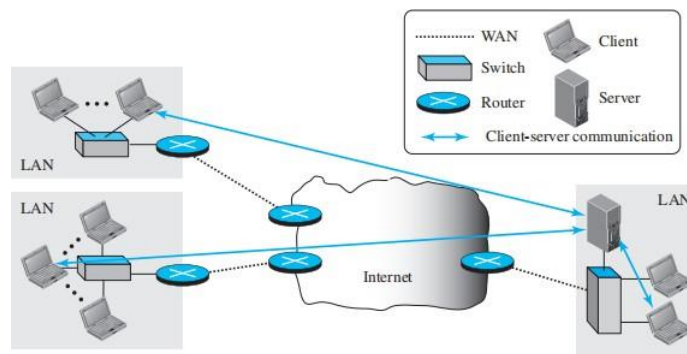
To use the Internet, two application programs must communicate: one on a computer somewhere in the world and the other on a different computer. These programs exchange messages over the Internet, but their relationship can vary. There are two main paradigms:

1. **Client-Server Paradigm:** One program (client) requests services, while the other (server) provides services.
2. **Peer-to-Peer (P2P) Paradigm:** Both programs can request and provide services.

Traditional Paradigm: Client-Server

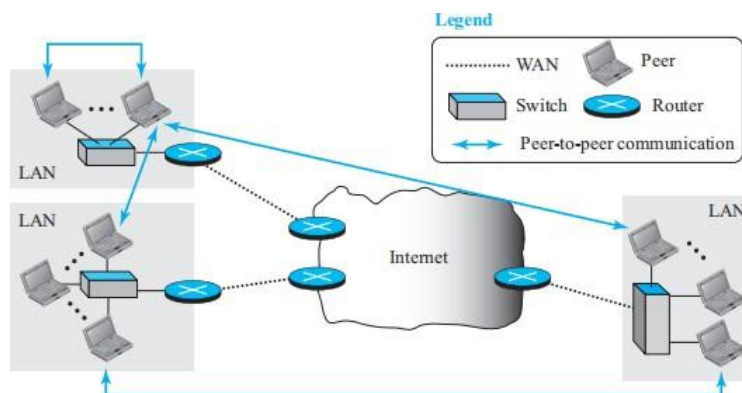
- ➔ In client-server architecture, there is an **always-on** host, called the **server**, which provides services when it receives requests from many other hosts, called **clients**.
- ➔ **Example:** In Web application Web server services requests from browsers running on client hosts. When a Web server receives a request for an object from a client host, it responds by sending the requested object to the client host.

- ➔ In client-server architecture, clients do not directly communicate with each other.
- ➔ The server has a fixed, well-known address, called an **IP address**. Because the server has a fixed, well-known address, and is always on, a client can always contact the server by sending a packet to the server's IP address.
- ➔ Some of the better-known applications with client-server architecture include the Web, FTP, Telnet, and e-mail.
- ➔ The most popular Internet services—such as search engines (e.g., Google and Bing), Internet commerce (e.g., Amazon and e-Bay), Web-based email (e.g., Gmail and Yahoo Mail), social networking (e.g., Facebook and Twitter)—employ one or more data centers.



New Paradigm: Peer-to-Peer

- ➔ In P2P architecture, there is minimal dependence on dedicated servers in data centers.
- ➔ The application employs direct communication between pairs of intermittently connected hosts, called peers.
- ➔ The peers are not owned by the service provider, but are instead desktops and laptops controlled by users, with most of the peers residing in homes, universities, and offices.
- ➔ Many of today's most popular and traffic-intensive applications are based on P2P architectures. These applications include file sharing (e.g., BitTorrent), Internet Telephony (e.g., Skype), and IPTV (e.g., Kankan and PPstream).



Mixed Paradigm

An application may choose to use a mixture of the two paradigms by combining the advantages of both. For example, a light-load client-server communication can be used to find the address of the peer that can offer a service. When the address of the peer is found, the actual service can be received from the peer by using the peer-to-peer paradigm.

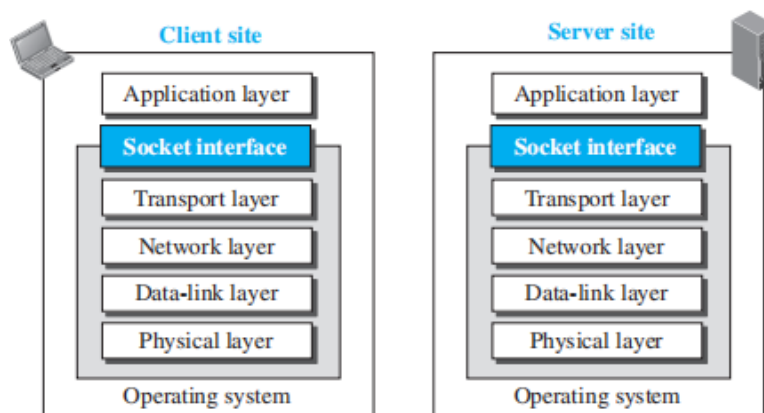
CLIENT-SERVER PROGRAMMING

In a client-server paradigm, communication at the application layer is between two running application programs called **processes**: a *client* and a *server*. The client is the program that initiates communication by sending a request, while the server is a program that waits for incoming requests, processes them, and returns a response. This setup requires that the server be running when a client request arrives, whereas the client only runs as needed. In practice, this means a client process can run on one computer and the server on another, but the server must be started first to be available for client requests.

Application Programming Interface

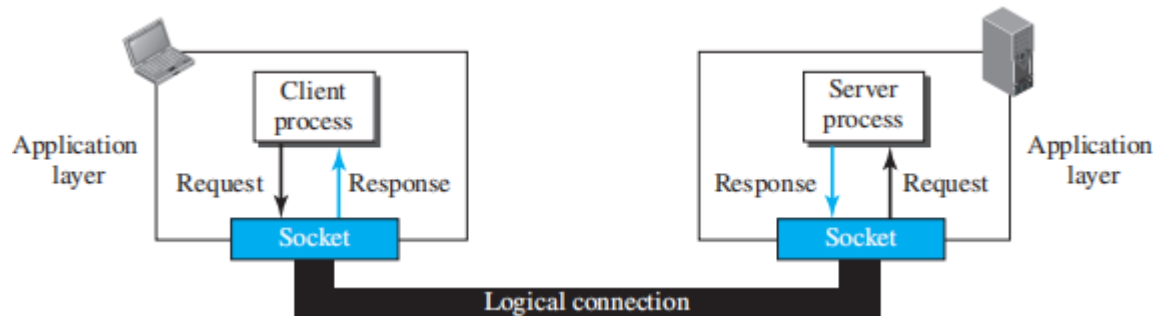
The interface between the application layer and the transport layer within a host which is also referred to as the **Application Programming Interface (API)**.

An API acts as a bridge between two parts: the application layer process (the program) and the operating system, which manages the first four layers of the TCP/IP protocol suite. Essentially, the computer's operating system includes these first four layers and an API so that applications can communicate through the Internet. This way, applications can easily send and receive messages by using the services of the operating system. te with the operating system when sending and receiving messages through the Internet. Several APIs have been designed for communication. Three among them are common: **socket interface**, **Transport Layer Interface (TLI)**, and **STREAM**.



Sockets

A process sends messages into, and receives messages from, the network through a software interface called a **socket**. The application at the sending side pushes messages through the socket. At the other side of the socket, the transport-layer protocol has the responsibility of getting the messages to the socket of the receiving process



Socket Addresses

The interaction between a client and a server is two-way communication. In a two-way communication, we need a pair of addresses: local (sender) and remote (receiver). The local address in one direction is the remote address in the other direction and vice versa. Since communication in the client-server paradigm is between two sockets, we need a pair of **socket addresses** for communication: a local socket address and a remote socket address.

To identify the receiving process, two pieces of information need to be specified:

The address of the host

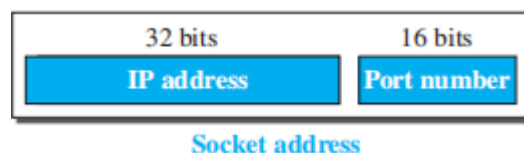
An identifier that specifies the receiving process in the destination host.

In the Internet, the host is identified by its **IP address**.

In addition to knowing the address of the host to which a message is destined, the sending process must also identify the receiving process running in the host.

A **destination port number** serves this purpose. Popular applications have been assigned specific port numbers.

For example, a Web server is identified by port number 80. A mail server process (using the SMTP protocol) is identified by port number 25.



Using Services of the Transport Layer

A pair of processes provide services to the users of the Internet, human or programs. A pair of processes, however, need to use the services provided by the transport layer for communication because there is no physical communication at the application layer. There are three common transport-layer protocols in the TCP/IP suite: UDP, TCP, and SCTP.

UDP Protocol

UDP is a connectionless, unreliable protocol that treats each message as an independent datagram with no connection between them. It doesn't guarantee delivery or retransmission of lost data but is fast and simple. UDP is ideal for applications that prioritize speed over reliability, like some management and multimedia apps.

TCP Protocol

TCP (Transmission Control Protocol) provides a connection-oriented, reliable, byte-stream service. It first establishes a logical connection between two ends through a handshaking process, setting parameters like packet size and buffer size. Data is then exchanged in segments, with byte numbering to ensure continuity. If bytes are lost or corrupted, TCP allows the receiver to request retransmission, making it reliable. TCP also supports flow and congestion control. However, TCP is not message-oriented and does not define boundaries in messages. Applications needing reliability and capable of handling long messages benefit from TCP.

SCTP Protocol

SCTP provides a service which is a combination of the two other protocols. Like TCP, SCTP provides a connection-oriented, reliable service, but it is not byte stream oriented. It is a message-oriented protocol like UDP. In addition, SCTP can provide multi-stream service by providing multiple network-layer connections. SCTP is normally suitable for any application that needs reliability and at the same time needs to remain connected, even if a failure occurs in one network-layer connection.

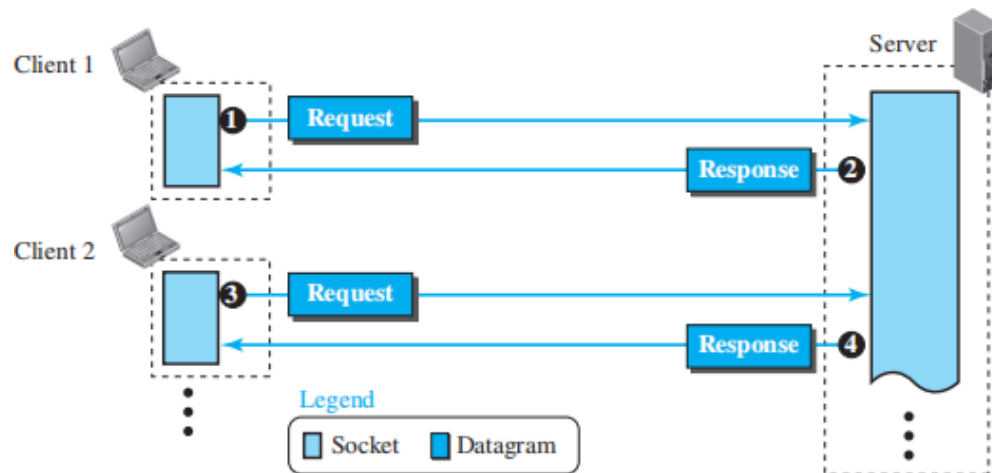
Iterative Communication Using UDP

Communication between a client program and a server program can occur iteratively or concurrently. Although several client programs can access the same server program at the same time, the server program can be designed to respond iteratively or concurrently. An iterative server can process one client request at a time; it receives a request, processes it, and sends the response to the requestor before handling another request. When the server is handling the request from a client, the requests from other clients, and even other requests from the same

client, need to be queued at the server site and wait for the server to be freed. The received and queued requests are handled in the first-in, first-out fashion.

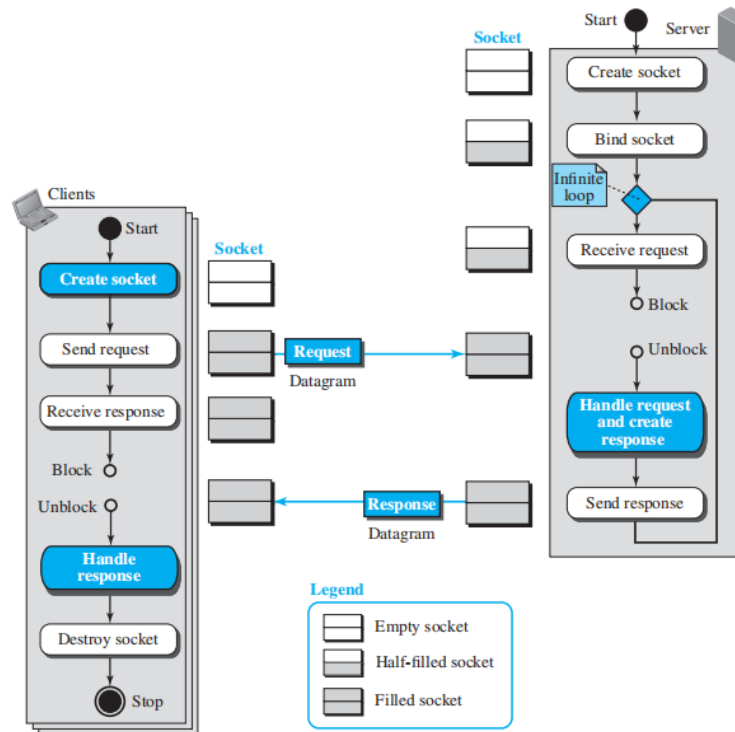
Sockets Used for UDP

In UDP communication, the client and server use only one socket each. The socket created at the server site lasts forever; the socket created at the client site is closed (destroyed) when the client process terminates. Figure shows the lifetime of the sockets in the server and client processes. This is logical, because the server does know its own socket address, but does not know the socket addresses of the clients who need its services; it needs to wait for the client to connect before filling this part of the socket address.



Flow Diagram

UDP provides a connectionless service, in which a client sends a request and the server sends back a response. Figure shows a simplified flow diagram for iterative communication. There are multiple clients, but only one server. Each client is served in each iteration of the loop in the server. If a client wants to send two datagrams, it is considered as two clients for the server. The second datagram needs to wait for its turn. The diagram also shows the status of the socket after each action.



Server Process

The server makes a *passive open*, in which it becomes ready for the communication, but it waits until a client process makes the connection. It creates an empty socket. It then binds the socket to the server and the well-known port, in which only part of the socket (the server socket address) is filled (binding can happen at the time of creation depending on the underlying language). The server then issues a receive request command, blocking it from doing anything else until a client request arrives.

When a client request is received, the server fills in the remaining socket information (the client's address), processes the request, and sends the response back to the client. After this, the server clears the client's address in the socket and waits for the next request in a continuous loop. Each time, the socket is only fully filled when a new request comes in.

Client Process

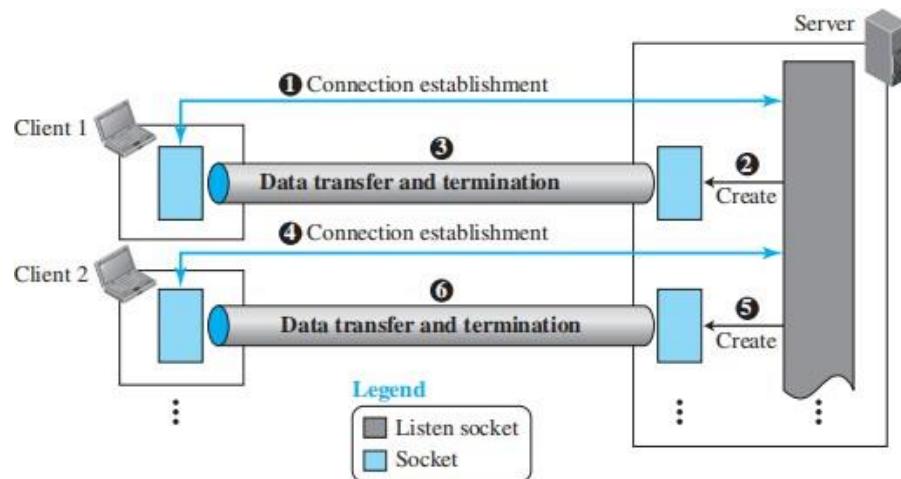
The client process makes an *active open*. In other words, it starts a connection. It creates an empty socket and then issues the send command, which fully fills the socket, and sends the request. The client then issues a receive command, which is blocked until a response arrives from the server. The response is then handled and the socket is destroyed.

Iterative Communication Using TCP

TCP is a connection-oriented protocol. Before sending or receiving data, a connection needs to be established between the client and the server. After the connection is established, the two parties can send and receive chunks of data as long as they have data to do so.

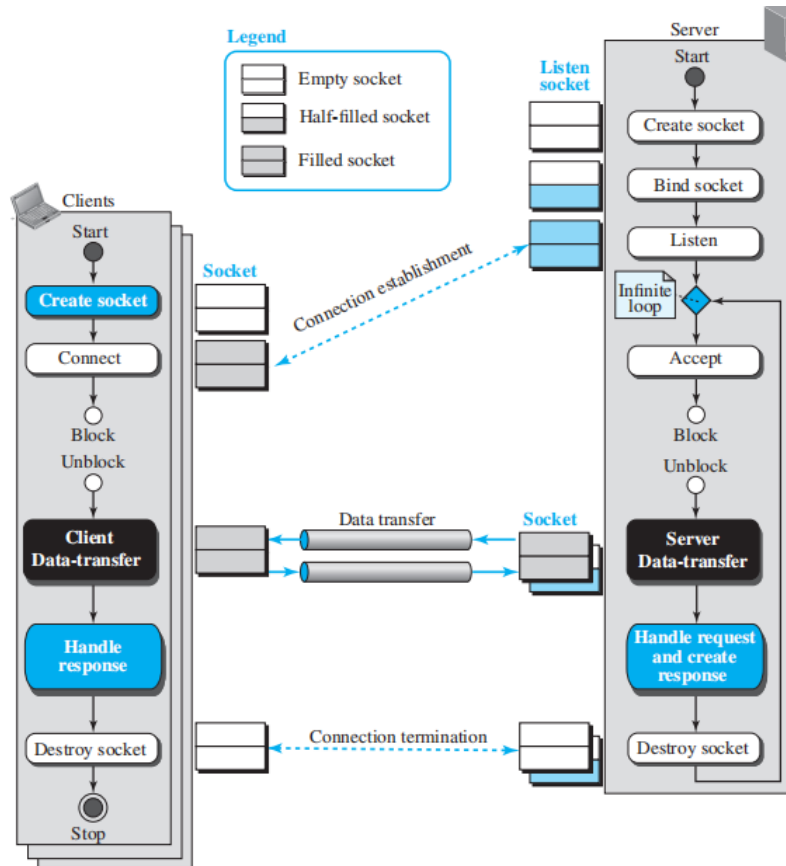
Sockets Used in TCP

The TCP server uses two different sockets, one for connection establishment and the other for data transfer. We call the first one the *listen socket* and the second the *socket*. The reason for having two types of sockets is to separate the connection phase from the data exchange phase. A server uses a listen socket to listen for a new client trying to establish connection. After the connection is established, the server creates a socket to exchange data with the client and finally to terminate the connection. The client uses only one socket for both connection establishment and data exchange.



Flow Diagram

Figure below shows a simplified flow diagram for iterative communication using TCP. There are multiple clients, but only one server. Each client is served in each iteration of the loop.



Server Process

- The TCP server creates a listening socket and binds it to a network address and port, used only for listening to incoming client connections.
- The server calls a "listen" function, allowing the operating system to accept connections and add them to a waiting list.
- The server enters a loop, handling each client one at a time.
- In each loop iteration, the server uses the "accept" function to pick a client from the waiting list; if no clients are waiting, the server blocks until a client arrives.
- Upon accepting a client, a new socket is created specifically for data exchange with that client.
- The server sets this new socket with the client's address for direct communication.
- The server and client can then exchange data using this client-specific socket.

Client Process

The client flow diagram is almost similar to the UDP version except that the *client data-transfer* box needs to be defined for each specific case.

Standard Client-Server Protocols

WORLD WIDE WEB AND HTTP

World Wide Web

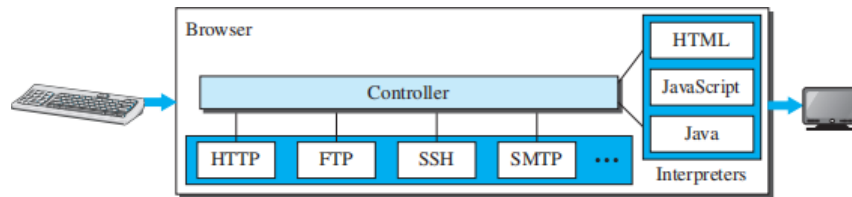
- The idea of the Web was first proposed by Tim Berners-Lee in 1989 at *CERN*[†], the European Organization for Nuclear Research, to allow several researchers at different locations throughout Europe to access each other's' researches.
- The commercial Web started in the early 1990s.
- Today, the Web is a global information space, where documents (called web pages) are spread across the world and connected through links.
- The Web's growth and popularity are tied to two key ideas: "distributed" and "linked."
 - **Distributed:** Any web server worldwide can add new pages, which helps expand the Web without overwhelming a few servers.
 - **Linked:** Web pages can reference other pages on different servers, creating a network of interconnected information.
- This linking is done through **hypertext**, a concept from before the Internet. It allows a document to automatically access another linked document. The Web made this possible electronically by letting users click a link to retrieve a connected document.
- The term **hypertext** evolved to **hypermedia**, meaning web pages can include not only text but also images, audio, and video.
- The Web's purpose has grown beyond sharing documents; it now supports online shopping, gaming, and on-demand access to radio and TV programs.

Architecture

The WWW today is a distributed client-server service, in which a client using a browser can access a service using a server. However, the service provided is distributed over many locations called **sites**. Each site holds one or more **web pages**. A web page can be **simple or composite**. A simple web page has no links to other web pages; a composite web page has one or more links to other web pages. Each web page is a file with a name and address.

Web Client (Browser)

A variety of vendors offer commercial **browsers** that interpret and display a web page, and all of them use nearly the same architecture. Each browser usually consists of three parts: a controller, client protocols, and interpreters.



- The controller receives input from the keyboard or the mouse and uses the client programs to access the document.
- After the document has been accessed, the controller uses one of the interpreters to display the document on the screen.
- The client protocol can be one of the protocols described later, such as HTTP or FTP.
- The interpreter can be HTML, Java, or JavaScript, depending on the type of document.
- Some commercial browsers include Internet Explorer, Netscape Navigator, and Firefox.

Web Server

- The web page is stored at the server. Each time a request arrives, the corresponding document is sent to the client.
- To improve efficiency, servers normally store requested files in a cache in memory; memory is faster to access than a disk.
- A server can also become more efficient through multithreading or multiprocessing. In this case, a server can answer more than one request at a time.
- Some popular web servers include Apache and Microsoft Internet Information Server.

Uniform Resource Locator (URL)

A web page, as a file, needs to have a unique identifier to distinguish it from other web pages.

To define a web page, four main parts are needed:

1. **Protocol:** This is like choosing the "vehicle" or method to reach the web page. Commonly, this is HTTP (HyperText Transfer Protocol), which lets the browser access and display web pages. However, other methods, like FTP (File Transfer Protocol), can also be used for different types of access.
2. **Host:** The host tells us where the web page is stored. It can be written as an IP address (e.g., 64.23.56.17) or as a domain name, like "example.com," which makes it easier to identify servers.
3. **Port:** The port number is a specific "entry point" on the server. For example, port 80 is usually used for HTTP, while HTTPS uses port 443. Most of the time, the port is

standard and doesn't need to be written, but if a different port is used, it can be specified in the address.

4. **Path:** The path gives the exact location of the web page file within the server. It shows the series of folders and the file name, like “/folder1/folder2/filename.” This tells the server exactly where to look to find and deliver the web page.

When we put these four elements together, we get a **URL (Uniform Resource Locator)**, which is the complete address that uniquely identifies and fetches a specific web page. Each part is separated to create a clear structure for finding the page.

<code>protocol://host/path</code>	Used most of the time
<code>protocol://host:port/path</code>	Used when port number is needed

Example

The URL <http://www.mhhe.com/compsci/forouzan/> defines the web page related to one of the authors of this book. The string **www.mhhe.com** is the name of the computer in the McGraw-Hill company (the three letters **www** are part of the host name and are added to the commercial host). The path is **compsci/forouzan/**, which defines Forouzan's web page under the directory **compsci** (computer science).

Web Documents

The documents in the WWW can be grouped into three broad categories: static, dynamic, and active

Static Documents

Static documents are fixed-content documents that are created and stored in a server. When a client accesses the document, a copy of the document is sent. The user can then use a browser to see the document. Static documents are prepared using one of several languages: *HyperText Markup Language* (HTML), *Extensible Markup Language* (XML), *Extensible Style Language* (XSL), and *Extensible Hypertext Markup Language* (XHTML).

Dynamic Documents

A **dynamic document** is created by a web server whenever a browser requests the document. When a request arrives, the web server runs an application program or a script that creates the dynamic document. The server returns the result of the program or script as a response to the browser that requested the document. Because a fresh document is created for each request, the contents of a dynamic document may vary from one request to another. A very simple example of a dynamic document is the retrieval of the time and date from a server.

Active Documents

Active documents are files that require a program or script to run on the client's device to enable interactive or animated content on the user's screen. When a browser requests an active

document, the server sends a copy of the file to the client. The document is then run by the client's browser, allowing animations and interactions to occur locally on the user's device.

Active Documents are created using two methods

1. **Java Applets:** Written in Java and compiled into bytecode (binary format). Ready to run on the client's device.
2. **JavaScript:** Scripts that are downloaded from the server and executed directly by the client's browser.

These methods enable dynamic, interactive content directly on the client's device, making it possible for applications like animations and user interactions to work smoothly at the client side.

HyperText Transfer Protocol (HTTP)

- The Hyper Text Transfer Protocol (HTTP), the Web's application-layer protocol, is at the heart of the Web.
- HTTP is implemented in two programs: a **client** program and a **server** program. The client program and server program, executing on different end systems, talk to each other by exchanging HTTP messages. HTTP defines the structure of these messages and how the client and server exchange the messages.
- A Web page consists of **objects**. An object is simply a file like HTML file, a JPEG image, a Java applet, or a video clip—that is addressable by a single URL.
- Most Web pages consist of a base HTML file and several referenced objects.
- The base HTML file references the other objects in the page with the objects' URLs.
- HTTP defines how Web clients request Web pages from Web servers and how servers transfer Web pages to clients.
- When a user requests a Web page (for example, clicks on a hyperlink), the browser sends HTTP request messages for the objects in the page to the server. The server receives the requests and responds with HTTP response messages that contain the objects.
- HTTP uses TCP as its underlying transport protocol. The HTTP client first initiates a TCP connection with the server. Once the connection is established, the browser and the server processes access TCP through their socket interfaces

Non-Persistent and Persistent Connections

If Separate TCP connection is used for each request and response, then the connection is said to be **non-persistent**.

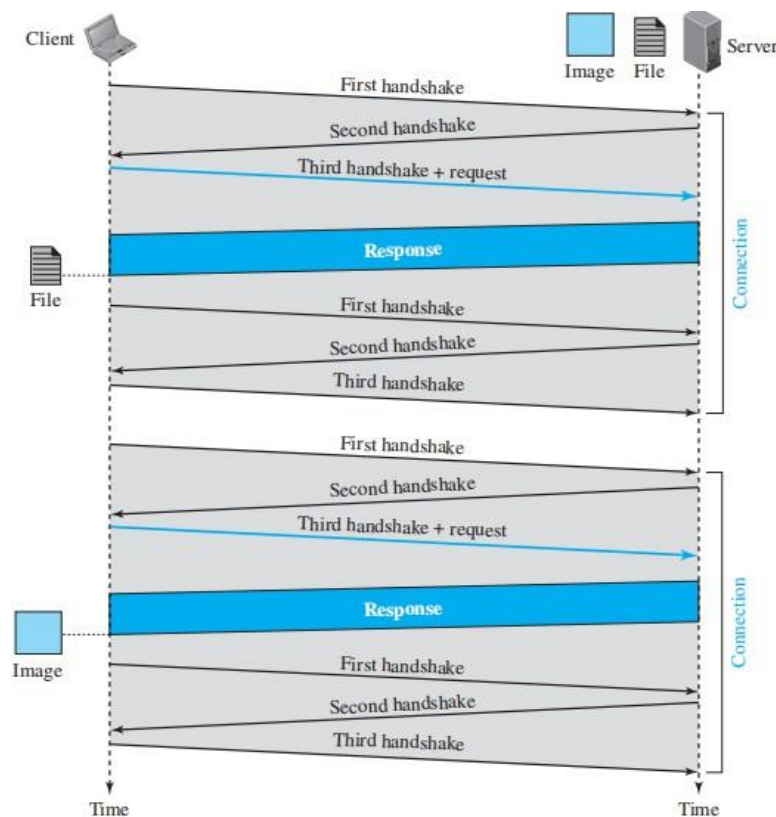
If same TCP connection is used for series of related request and response, then the connection is said to be **persistent**.

Nonpersistent Connections

In a **nonpersistent connection**, one TCP connection is made for each request/response. The following lists the steps in this strategy:

1. The client opens a TCP connection and sends a request.
2. The server sends the response and closes the connection.
3. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

In this strategy, if a file contains links to N different pictures in different files (all located on the same server), the connection must be opened and closed $N + 1$ times. The nonpersistent strategy imposes high overhead on the server because the server needs $N + 1$ different buffers each time a connection is opened.



- Round-trip time (RTT) is the time it takes for a small packet to travel from client to server and then back to the client.
- The RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.
- When a user clicks on a hyperlink, the browser initiates a TCP connection between the browser and the Web server; this involves a “three-way handshake”—the client sends a small TCP segment to the server, the server acknowledges and responds with a small TCP segment, and, finally, the client acknowledges back to the server.

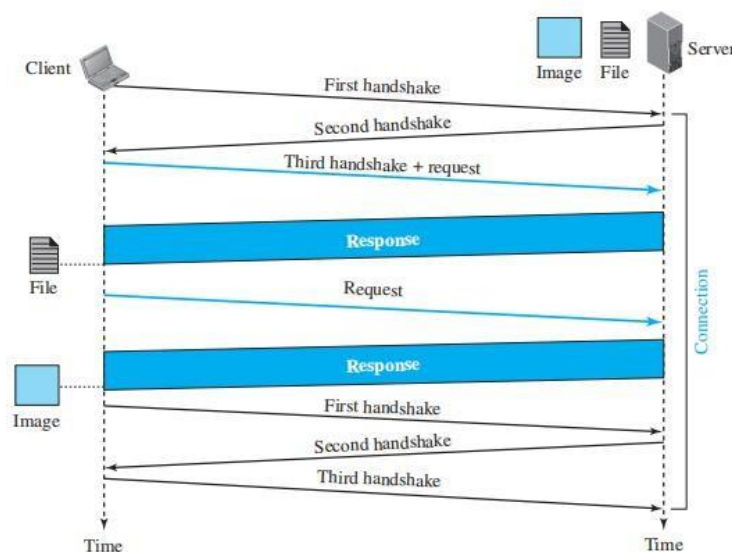
- The first two parts of the three-way handshake take one RTT.
- After completing the first two parts of the handshake, the client sends the HTTP request message combined with the third part of the three-way handshake (the acknowledgment) into the TCP connection.
- Once the request message arrives at the server, the server sends the HTML file into the TCP connection. This HTTP request/response eats up another RTT. Thus, roughly, the total response time is two RTTs plus the transmission time at the server of the HTML file.

Persistent Connections

HTTP version 1.1 specifies a persistent connection by default. Non-persistent connections have some shortcomings.

1. A brand-new connection must be established and maintained for each requested object. This can place a significant burden on the Web server, which may be serving requests from hundreds of different clients simultaneously.
2. Each object suffers a delivery delay of two RTTs— one RTT to establish the TCP connection and one RTT to request and receive an object.

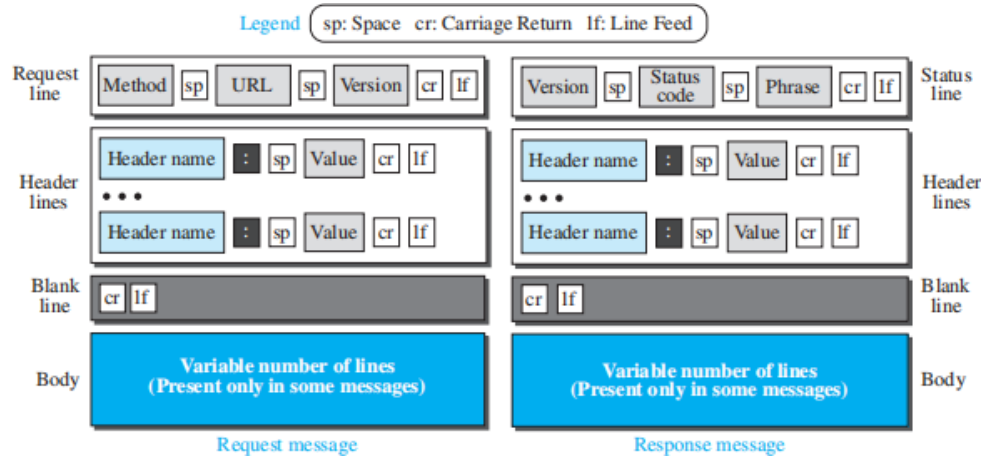
With persistent connections, the server leaves the TCP connection open after sending a response. Subsequent requests and responses between the same client and server can be sent over the same connection. In particular, an entire Web page can be sent over a single persistent TCP connection. Moreover, multiple Web pages residing on the same server can be sent from the server to the same client over a single persistent TCP connection.



Only one connection establishment and connection termination is used, but the request for the image is sent separately

Message Formats

The HTTP protocol defines the format of the request and response messages.



Request Message

Method: There are five HTTP methods:

- **GET:** The GET method is used when the browser requests an object, with the requested object identified in the URL field.
- **POST:** With a POST message, the user is still requesting a Web page from the server, but the specific contents of the Web page depend on what the user entered into the form fields. If the value of the method field is POST, then the entity body contains what the user entered into the form fields.
- **PUT:** The PUT method is also used by applications that need to upload objects to Web servers.
- **HEAD:** Used to retrieve header information. It is used for debugging purpose.
- **DELETE:** The DELETE method allows a user, or an application, to delete an object on a Web server.

URL: Specifies URL of the requested object

Version: This field represents HTTP version, usually HTTP/1.1

Header line: Ex:

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

The header line **Host:www.someschool.edu** specifies the host on which the object resides.

By including the **Connection:close** header line, the browser is telling the server that it doesn't want to bother with persistent connections; it wants the server to close the connection after sending the requested object.

The **User-agent:** header line specifies the user agent, that is, the browser type that is making the request to the server. Here the user agent is Mozilla/5.0, a Firefox browser.

The **Accept-language:** header indicates that the user prefers to receive a French version of the object, if such an object exists on the server; otherwise, the server should send its default version.

Response Message**EX**

HTTP/1.1 200 OK

Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

(data data data data data ...)

The **status line** has three fields: the protocol version field, a status code, and a corresponding status message.

Version is HTTP/1.1

The status code and associated phrase indicate the result of the request. Some common status codes and associated phrases include:

1. 200 OK: Request succeeded and the information is returned in the response.
2. 301 Moved Permanently: Requested object has been permanently moved; the new URL is specified in Location: header of the response message. The client software will automatically retrieve the new URL.
3. 400 Bad Request: This is a generic error code indicating that the request could not be understood by the server.
4. 404 Not Found: The requested document does not exist on this server.
5. 505 HTTP Version Not Supported: The requested HTTP protocol version is not supported by the server.

Header fields:

- The server uses the **Connection: close** header line to tell the client that it is going to close the TCP connection after sending the message.
- The **Date:** header line indicates the time and date when the HTTP response was created and sent by the server.
- The **Server:** header line indicates that the message was generated by an Apache Web server; it is analogous to the User-agent: header line in the HTTP request message.

- The **Last-Modified:** header line indicates the time and date when the object was created or last modified.
- The **Content-Length:** header line indicates the number of bytes in the object being sent.
- The **Content-Type:** header line indicates that the object in the entity body is HTML text.

Conditional Request

A client can add a condition in its request. In this case, the server will send the requested web page if the condition is met or inform the client otherwise. One of the most common conditions imposed by the client is the time and date the web page is modified. The client can send the header line *If-Modified-Since* with the request to tell the server that it needs the page only if it is modified after a certain point in time.

Example

The following shows how a client imposes the modification data and time condition on a request.

GET http://www.commonServer.com/information/file1 HTTP/1.1	Request line
If-Modified-Since: Thu, Sept 04 00:00:00 GMT	Header line
	Blank line

The status line in the response shows the file was not modified after the defined point in time. The body of the response message is also empty.

HTTP/1.1 304 Not Modified	Status line
Date: Sat, Sept 06 08 16:22:46 GMT	First header line
Server: commonServer.com	Second header line
	Blank line
(Empty Body)	Empty body

Cookies

An HTTP cookie (also called web cookie, Internet cookie, browser cookie, or simply cookie) is a small piece of data sent from a website and stored on the user's computer by the user's web browser while the user is browsing. Cookies were designed to be a reliable mechanism for websites to remember stateful information (such as items added in the shopping cart in an online store) or to record the user's browsing activity (including clicking particular buttons, logging in, or recording which pages were visited in the past). They can also be used to remember arbitrary pieces of information that the user previously entered into form fields such as names, addresses, passwords, and credit card numbers.

It is often desirable for a Web site to identify users, either because the server wishes to restrict

user access or because it wants to serve content as a function of the user identity. For these purposes, HTTP uses cookies.

Creating and Storing Cookies

The creation and storing of cookies depend on the implementation; however, the principle is the same.

1. When a server receives a request from a client, it stores information about the client in a file or a string. The information may include the domain name of the client, the contents of the cookie (information the server has gathered about the client such as name, registration number, and so on), a timestamp, and other information depending on the implementation.
2. The server includes the cookie in the response that it sends to the client.
3. When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the server domain name.

Using Cookies

When a client sends a request to a server, the browser looks in the cookie directory to see if it can find a cookie sent by that server. If found, the cookie is included in the request. When the server receives the request, it knows that this is an old client, not a new one. The contents of the cookie are never read by the browser or disclosed to the user. It is a cookie *made* by the server and *eaten* by the server.

- An e-commerce store uses a cookie to track client shopping. When an item is added to the cart, a cookie with the item number and unit price is sent to the browser. Each new selection updates the cookie. At checkout, the last cookie is retrieved to calculate the total charge.
- The site that restricts access to *registered clients* only sends a cookie to the client when the client registers for the first time. For any repeated access, only those clients that send the appropriate cookie are allowed.
- A web *portal* uses the cookie in a similar way. When a user selects her favourite pages, a cookie is made and sent. If the site is accessed again, the cookie is sent to the server to show what the client is looking for.
- Advertising agencies use cookies to track user behavior via banner ads on popular websites. When a user clicks a corporation's icon, a request goes to the agency, which sends the banner along with a cookie containing the user's ID. Future banner interactions build a profile of the user's web behavior. This data can be sold to third parties, making cookies controversial. New regulations are hoped to protect user privacy.

Ex:

Assume a shopper wants to buy a toy from an electronic store named BestToys.

Shopper initiates request: Shopper visits the BestToys website and requests to view toys.

Server creates cart: BestToys' server creates an empty shopping cart for this shopper, assigning a unique ID (e.g., 12343) to the cart.

Server sends response: Server sends a response with toy images, each with clickable links.

This response also includes a Set-Cookie header with the cart ID (12343), which is stored on the client's device in a file named BestToys.

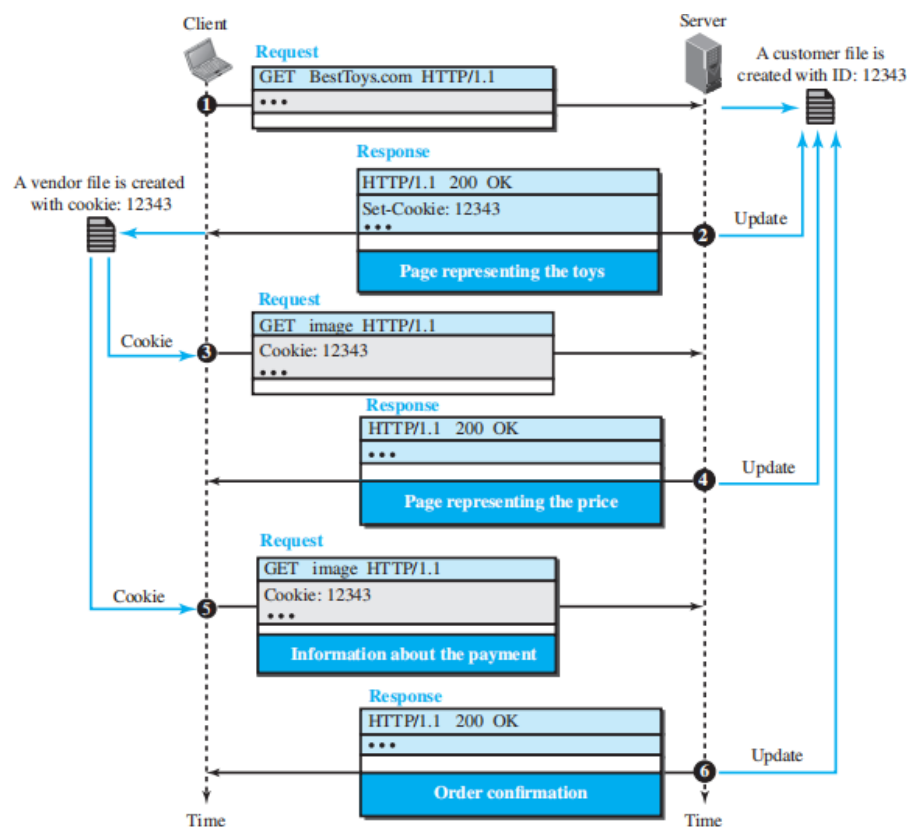
Shopper selects toy: Shopper clicks a toy to add to the cart; the client sends a request with the cookie (ID 12343) in the header, identifying the shopper.

Server retrieves cart: Server recognizes the returning ID 12343, retrieves the existing shopping cart, and adds the selected toy to the cart.

Server provides total price: Server responds with the updated total price and prompts the shopper for payment details.

Shopper makes payment: Shopper provides credit card information and submits it with the ID 12343 in the request.

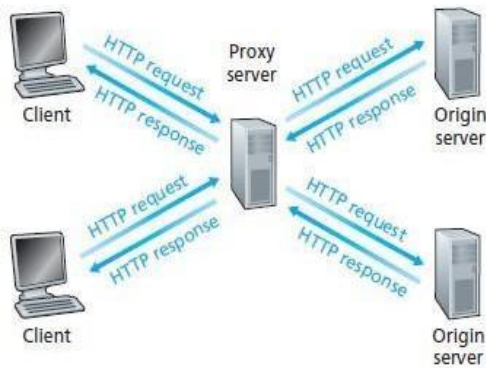
Order confirmation: Server confirms the ID, processes payment, completes the order, and sends a confirmation message to the shopper.



Web Caching

- A Web cache—also called a proxy server—is a network entity that satisfies HTTP requests on the behalf of an origin Web server.
- The Web cache has its own disk storage and keeps copies of recently requested objects in this storage.
- A user's browser can be configured so that all of the user's HTTP requests are first

directed to the Web cache.



Ex

Suppose a browser is requesting the object <http://www.someschool.edu/campus.gif>.

Here is what happens:

1. The browser establishes a TCP connection to the Web cache and sends an HTTP request for the object to the Web cache.
2. The Web cache checks to see if it has a copy of the object stored locally. If it does, the Web cache returns the object within an HTTP response message to the client browser
3. If the Web cache does not have the object, the Web cache opens a TCP connection to the origin server, that is, to www.someschool.edu. The Web cache then sends an HTTP request for the object into the cache-to-server TCP connection.
4. After receiving this request, the origin server sends the object within an HTTP response to the Web cache.
5. When the Web cache receives the object, it stores a copy in its local storage and sends a copy, within an HTTP response message, to the client browser (over the existing TCP connection between the client browser and the Web cache).

HTTP Security

HTTP per se does not provide security. HTTP can be run over the Secure Socket Layer (SSL). In this case, HTTP is referred to as HTTPS. HTTPS provides confidentiality, client and server authentication, and data integrity.

FTP

FTP (File Transfer Protocol) is used for transferring file from one host to another host. Transferring files between systems isn't always simple, as systems can have different file name rules, data formats, or directory structures. FTP (File Transfer Protocol) addresses these issues smoothly, making it easier to move files between different systems. While HTTP can also transfer files, FTP is generally better for large files or when handling various file formats.

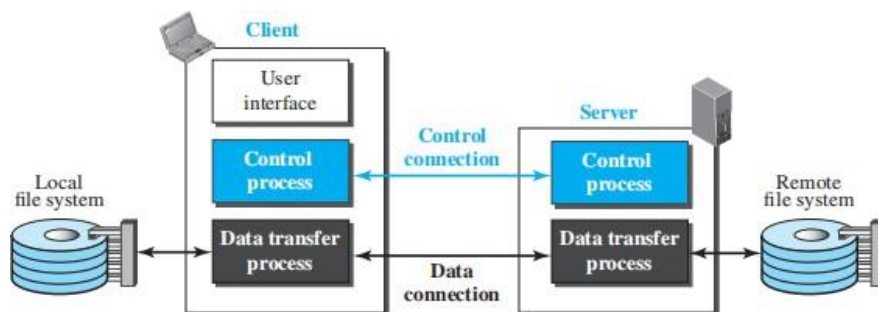


Figure shows the basic model of FTP. The client has three components: the user interface, the client control process, and the client data transfer process.

The server has two components: the server control process and the server data transfer process.

The control connection is made between the control processes. The data connection is made between the data transfer processes.

Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data types transferred.

Two Connections

The two connections in FTP have different lifetimes. The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transfer FTP uses two well-known TCP ports: port 21 is used for the control connection, and port 20 is used for the data connection.

Control Connection

Control communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (or response) at a time. Each line is terminated with a two-character (carriage return and line feed) end-of-line token. During this control connection, commands are sent from the client to the server and responses are sent from the server to the client. Commands, which are sent from the FTP client control process, are in the form of ASCII uppercase, which may or may not be followed by an argument. Some of the most common commands are shown in Table.

<i>Command</i>	<i>Argument(s)</i>	<i>Description</i>
ABOR		Abort the previous command
CDUP		Change to parent directory
CWD	Directory name	Change to another directory
DELE	File name	Delete a file
LIST	Directory name	List subdirectories or files
MKD	Directory name	Create a new directory
PASS	User password	Password
PASV		Server chooses a port
PORT	Port identifier	Client chooses a port
PWD		Display name of current directory
QUIT		Log out of the system
RETR	File name(s)	Retrieve files; files are transferred from server to client
RMD	Directory name	Delete a directory
RNFR	File name (old)	Identify a file to be renamed
RNTO	File name (new)	Rename the file
STOR	File name(s)	Store files; file(s) are transferred from client to server
STRU	F, R, or P	Define data organization (F : file, R : record, or P : page)
TYPE	A, E, I	Default file type (A : ASCII, E : EBCDIC, I : image)
USER	User ID	User information
MODE	S, B, or C	Define transmission mode (S : stream, B : block, or C : compressed)

Every FTP command generates at least one response. A response has two parts: a three-digit number followed by text. The numeric part defines the code; the text part defines needed parameters or further explanations. The first digit defines the status of the command. The second digit defines the area in which the status applies. The third digit provides additional information. Table shows some common responses.

<i>Code</i>	<i>Description</i>	<i>Code</i>	<i>Description</i>
125	Data connection open	250	Request file action OK
150	File status OK	331	User name OK; password is needed
200	Command OK	425	Cannot open data connection
220	Service ready	450	File action not taken; file not available
221	Service closing	452	Action aborted; insufficient storage
225	Data connection open	500	Syntax error; unrecognized command
226	Closing data connection	501	Syntax error in parameters or arguments
230	User login OK	530	User not logged in

Data Connection

The data connection uses the well-known port 20 at the server site. However, the creation of a data connection is different from the control connection. The following shows the steps:

1. The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.
2. Using the PORT command the client sends this port number to the server.
3. The server receives the port number and issues an active open using the well-known port 20 and the received ephemeral port number.

Communication over Data Connection

The purpose and implementation of the data connection are to transfer files through the data connection. The client must define the type of file to be transferred, the structure of the data, and the transmission mode.

Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communication: file type, data structure, and transmission mode.

File Type

FTP can transfer one of the following file types across the data connection: ASCII file, EBCDIC file, or image file.

Data Structure

FTP can transfer a file across the data connection using one of the following interpretations of the structure of the data: *file structure*, *record structure*, or *page structure*.

Transmission Mode

FTP can transfer a file across the data connection using one of the following three transmission modes: *stream mode*, *block mode*, or *compressed mode*. The stream mode is the

default mode; data are delivered from FTP to TCP as a continuous stream of bytes. In the block mode, data can be delivered from FTP to TCP in blocks.

File Transfer

File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things: *retrieving a file* (server to client), *storing a file* (client to server), and *directory listing* (server to client).

Security for FTP

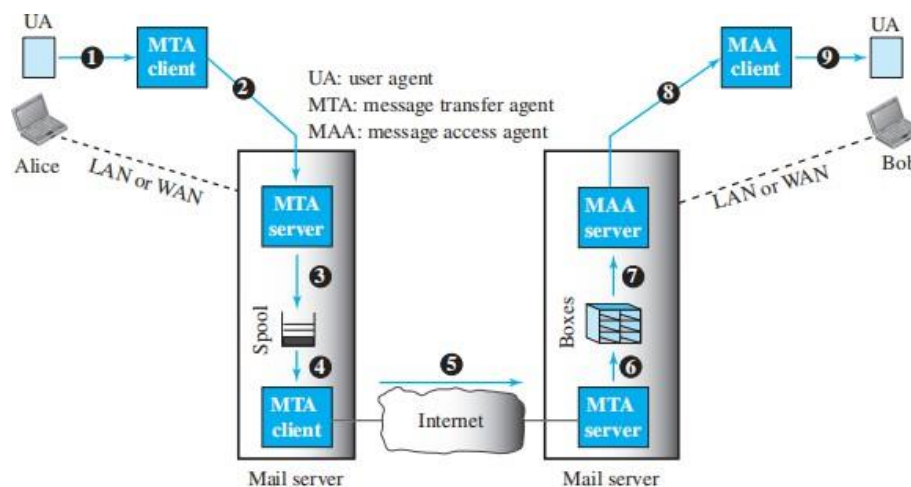
The FTP protocol was designed when security was not a big issue. Although FTP requires a password, the password is sent in plaintext (unencrypted), which means it can be intercepted and used by an attacker. The data transfer connection also transfers data in plain text, which is insecure. To be secure, one can add a Secure Socket Layer between the FTP application layer and the TCP layer. In this case FTP is called SSL-FTP.

ELECTRONIC MAIL

Electronic mail (or e-mail) allows users to exchange messages. In applications like HTTP or FTP, the server constantly awaits client requests to provide a response, whereas in email, messages are sent and stored until the recipient retrieves them. When Alice sends an email to Bob, a response is optional and, if given, is a separate transaction. Unlike typical client-server setups, Bob doesn't run a server to receive emails, as he might be offline. Instead, intermediate servers handle the client-server functions, allowing users to connect only when they wish to check or send messages.

Architecture

To explain the architecture of email, we present a typical scenario, as illustrated in Figure. An alternative scenario occurs when Alice or Bob is directly connected to their respective mail server, eliminating the need for a LAN or WAN connection; however, this variation does not affect the overall discussion.



In the common scenario, the sender and the receiver of the e-mail, Alice and Bob respectively, are connected via a LAN or a WAN to two mail servers. The administrator has created one mailbox for each user where the received messages are stored. A *mailbox* is part of a server hard drive, a special file with permission restrictions. Only the owner of the mailbox has access to it. The administrator has also created a queue (spool) to store messages waiting to be sent.

A simple e-mail from Alice to Bob takes nine different steps, as shown in the figure. Alice and Bob use three different *agents*: a **user agent (UA)**, a **message transfer agent (MTA)**, and a **message access agent (MAA)**.

1. Alice composes an email using her User Agent (UA).
2. The UA sends the message to the Message Transfer Agent (MTA) client on Alice's mail server.
3. The MTA client forwards the message to the MTA server on the same mail server.
4. The email is placed in a spool (queue) on Alice's mail server, awaiting transfer.
5. The MTA client sends the email across the Internet to Bob's mail server.
6. The MTA server on Bob's mail server receives the email.
7. The email is stored in Bob's mailbox on his mail server.
8. Bob's Message Access Agent (MAA) server retrieves the message from the mailbox.
9. Bob uses his MAA client to access the message through his User Agent (UA).

There are two important points we need to emphasize here.

1. Bob cannot directly use the MTA server to receive messages, as this would require him to run the MTA server continuously, keeping his computer and network connection on all the time—an impractical setup, especially with a WAN connection.
2. Unlike the MTA client-server, which "pushes" messages to the server, Bob requires "pull" programs to retrieve messages. This is why he needs Message Access Agent (MAA) programs, allowing him to pull messages from the server when needed.

User Agent

The first component of an electronic mail system is the **user agent (UA)**. A user agent is a software package (program) that composes, reads, replies to, and forwards messages. It also handles local mailboxes on the user computers.

There are two types of user agents: **command-driven and GUI-based**.

A command-driven user agent normally accepts a one-character command from the keyboard to perform its task. For example, a user can type the character *r*, at the command prompt, to reply to the sender of the message, or type the character *R* to reply to the sender and all recipients. Some examples of command driven user agents are *mail*, *pine*, and *elm*.

Modern user agents are GUI-based. They contain graphical user interface (GUI) components that allow the user to interact with the software by using both the keyboard and the mouse. Some examples of GUI-based user agents are *Eudora* and *Outlook*.

Sending Mail

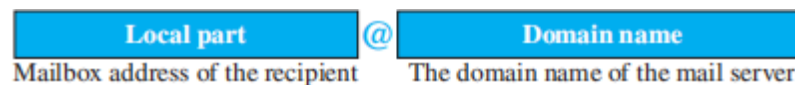
To send mail, the user, through the UA, creates mail that looks very similar to postal mail. It has an *envelope* and a *message*. The envelope usually contains the sender address, the receiver address, and other information. The message contains the *header* and the *body*. The header of the message defines the sender, the receiver, the subject of the message, and some other information. The body of the message contains the actual information to be read by the recipient.

Receiving Mail

The user agent is triggered by the user (or a timer). If a user has mail, the UA informs the user with a notice. If the user is ready to read the mail, a list is displayed in which each line contains a summary of the information about a particular message in the mailbox. The summary usually includes the sender mail address, the subject, and the time the mail was sent or received. The user can select any of the messages and display its contents on the screen.

Addresses

To deliver mail, a mail handling system must use an addressing system with unique addresses. In the Internet, the address consists of two parts: a *local part* and a *domain name*, separated by an @ sign.



The local part of an email address specifies the user's mailbox, while the domain part identifies the mail server or exchanger for sending and receiving mail.

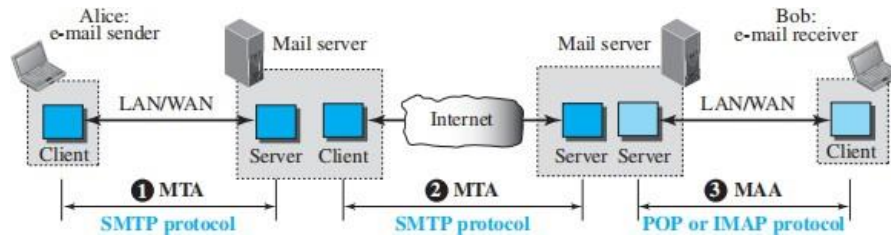
Mailing List or Group List

Electronic mail allows one name, an *alias*, to represent several different e-mail addresses; this is called a mailing list. Every time a message is to be sent, the system checks the recipient's name against the alias database; if there is a mailing list for the defined alias, separate messages, one for each entry in the list, must be prepared and handed to the MTA.

Message Transfer Agent: SMTP

An e-mail is an application that needs three uses of client-server paradigms to accomplish its task. It is important that we distinguish these three when we are dealing with e-mail. Figure shows these three client-server applications. We refer to the first and the second as Message Transfer Agents (MTAs), the third as Message Access Agent (MAA).

The formal protocol that defines the MTA client and server in the Internet is called ***Simple Mail Transfer Protocol (SMTP)***.



Commands and Responses

SMTP uses commands and responses to transfer messages between an MTA client and an MTA server. The command is from an MTA client to an MTA server; the response is from an MTA server to the MTA client. Each command or reply is terminated by a two character (carriage return and line feed) end-of-line token.

Mail Transfer Phases

The process of transferring a mail message occurs in three phases: connection establishment, mail transfer, and connection termination.

Connection Establishment

After a client has made a TCP connection to the well known port 25, the SMTP server starts the connection phase. This phase involves the following three steps:

1. The server sends code 220 (service ready) to tell the client that it is ready to receive mail. If the server is not ready, it sends code 421 (service not available).
2. The client sends the HELO message to identify itself, using its domain name address. This step is necessary to inform the server of the domain name of the client.
3. The server responds with code 250 (request command completed) or some other code depending on the situation.

Message Transfer

After connection has been established between the SMTP client and server, a single message between a sender and one or more recipients can be exchanged. This phase involves eight steps. Steps 3 and 4 are repeated if there is more than one recipient.

1. The client sends the MAIL FROM message to introduce the sender of the message. It includes the mail address of the sender (mailbox and the domain name). This step is needed to give the server the return mail address for returning errors and reporting messages.
2. The server responds with code 250 or some other appropriate code.
3. The client sends the RCPT TO (recipient) message, which includes the mail address of the recipient.
4. The server responds with code 250 or some other appropriate code.
5. The client sends the DATA message to initialize the message transfer.
6. The server responds with code 354 (start mail input) or some other appropriate message.

7. The client sends the contents of the message in consecutive lines. Each line is terminated by a two-character end-of-line token (carriage return and line feed). The message is terminated by a line containing just one period.
8. The server responds with code 250 (OK) or some other appropriate code.

Connection Termination

After the message is transferred successfully, the client terminates the connection. This phase involves two steps.

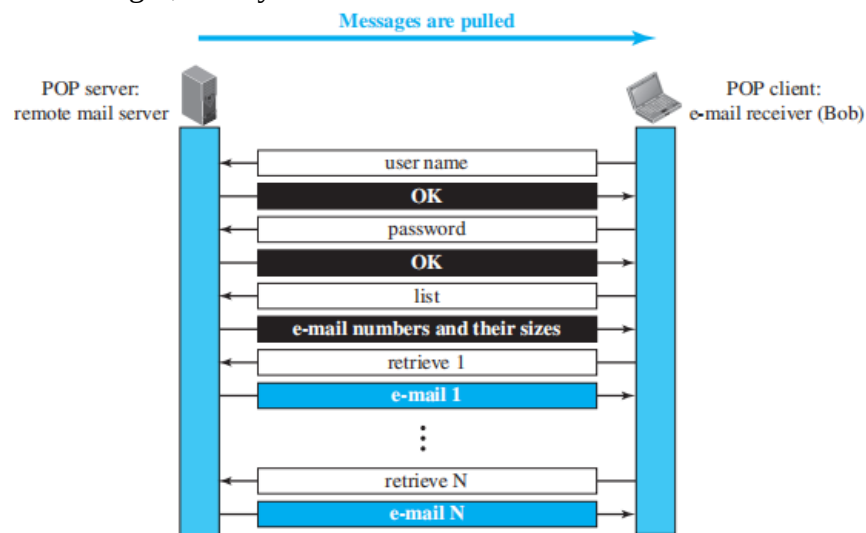
1. The client sends the QUIT command.
2. The server responds with code 221 or some other appropriate code.

Message Access Agent: POP and IMAP

SMTP protocol delivers the mail to the mail server (Push Protocol). To fetch the mail from mail server receiver used mail access protocols (Pull Protocol). There are currently a number of popular mail access protocols, including Post Office Protocol— Version 3 (POP3), Internet Mail Access Protocol (IMAP), and HTTP.

POP3

Post Office Protocol, version 3 (POP3) is simple but limited in functionality. The client POP3 software is installed on the recipient computer; the server POP3 software is installed on the mail server. Mail access starts with the client when the user needs to download its e-mail from the mailbox on the mail server. The client opens a connection to the server on TCP port 110. It then sends its user name and password to access the mailbox. The user can then list and retrieve the mail messages, one by one.



POP3 has two modes: the **delete** mode and the **keep** mode. In the delete mode, the mail is deleted from the mailbox after each retrieval. In the keep mode, the mail remains in the mailbox after retrieval.

IMAP4

Another mail access protocol is **Internet Mail Access Protocol, version 4 (IMAP4)**. IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more complex. POP3 is deficient in several ways. It does not allow the user to organize her mail on the server; the user cannot have different folders on the server. In addition, POP3 does not allow the user to partially check the contents of the mail before downloading. IMAP4 provides the following extra functions:

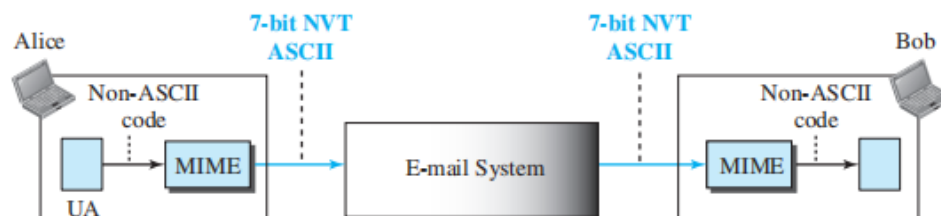
- A user can check the e-mail header prior to downloading.
- A user can search the contents of the e-mail for a specific string of characters prior to downloading.
- A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
- A user can create, delete, or rename mailboxes on the mail server.
- A user can create a hierarchy of mailboxes in a folder for e-mail storage.

MIME

Email has a simple structure, but it is limited to sending messages in 7-bit ASCII format, which restricts its use for non-English languages and prevents sending binary files, video, or audio data.

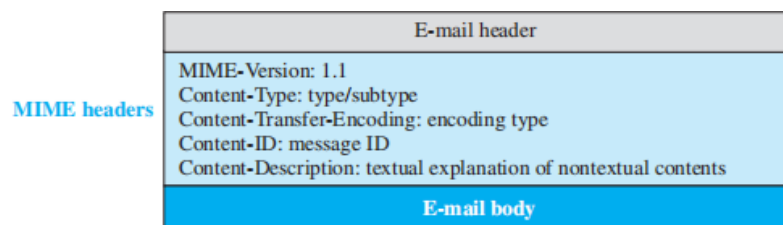
Multipurpose Internet Mail Extensions (MIME) is a supplementary protocol that allows non-ASCII data to be sent through e-mail. MIME transforms non-ASCII data at the sender site to NVT ASCII data and delivers it to the client MTA to be sent through the Internet. The message at the receiving site is transformed back to the original data.

MIME is a set of software functions that transforms non-ASCII data to ASCII data and vice versa.



MIME Headers

MIME defines five headers, which can be added to the original e-mail header section to define the transformation parameters:



MIME-Version -This header defines the version of MIME used. The current version is 1.1.

Content-Type -This header defines the type of data used in the body of the message. The content type and the content subtype are separated by a slash. Depending on the subtype, the header may contain other parameters. MIME allows seven different types of data, listed in Table

Type	Subtype	Description
Text	Plain	Unformatted
	HTML	HTML format (see Appendix C)
Multipart	Mixed	Body contains ordered parts of different data types
	Parallel	Same as above, but no order
	Digest	Similar to Mixed, but the default is message/RFC822
	Alternative	Parts are different versions of the same message
Message	RFC822	Body is an encapsulated message
	Partial	Body is a fragment of a bigger message
	External-Body	Body is a reference to another message
Image	JPEG	Image is in JPEG format
	GIF	Image is in GIF format
Video	MPEG	Video is in MPEG format
Audio	Basic	Single channel encoding of voice at 8 KHz
Application	PostScript	Adobe PostScript
	Octet-stream	General binary data (eight-bit bytes)

Content-Transfer-Encoding

This header defines the method used to encode the messages into 0s and 1s for transport. The five types of encoding methods are listed in Table.

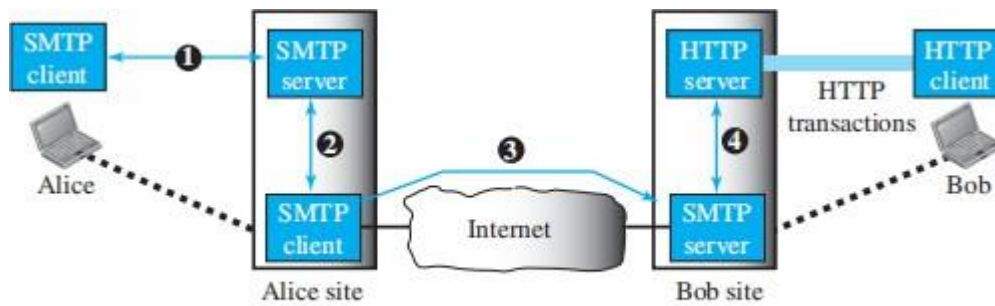
Type	Description
7-bit	NVT ASCII characters with each line less than 1000 characters
8-bit	Non-ASCII characters with each line less than 1000 characters
Binary	Non-ASCII characters with unlimited-length lines
Base64	6-bit blocks of data encoded into 8-bit ASCII characters
Quoted-printable	Non-ASCII characters encoded as an equal sign plus an ASCII code

Content-ID - This header uniquely identifies the whole message in a multiple message environment.

Content-Description -This header defines whether the body is image, audio, or video.

Web-Based Mail

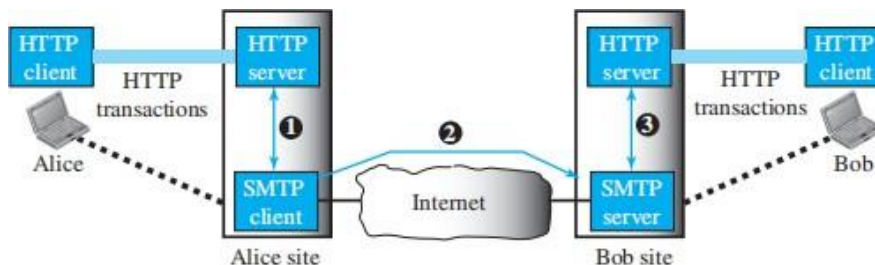
E-mail is such a common application that some websites today provide this service to anyone who accesses the site. Three common sites are Hotmail, Yahoo, and Google mail. The idea is very simple. Figure shows two cases:



Case 1: Only receiver uses HTTP

Case I

In this scenario, Alice uses a traditional mail server to send an email to Bob, who has an account on a web-based server. The email is transferred from Alice's browser to her mail server via SMTP, and from the sending mail server to the receiving mail server through SMTP as well. However, when the message reaches Bob's web server, it is transferred to Bob's browser via HTTP, not POP3 or IMAP4. Bob requests his emails through HTTP by logging into the website (e.g., Hotmail). Once logged in, the web server sends a list of emails in HTML format, and Bob can browse and retrieve his emails using more HTTP transactions.



Case 2: Both sender and receiver use HTTP

Case II

In this case, both Alice and Bob use web servers, but not necessarily the same one. Alice sends her email to her web server using HTTP, with Bob's mailbox address as the URL. The Alice server then forwards the message to Bob's server using SMTP. Bob receives the email through HTTP transactions, but the transfer between servers still occurs via SMTP.

E-Mail Security

The email protocols do not include built-in security features. However, email exchanges can be secured using two application-layer security protocols, Pretty Good Privacy (PGP) and Secure/Multipurpose Internet Mail Extensions (S/MIME).

TELNET

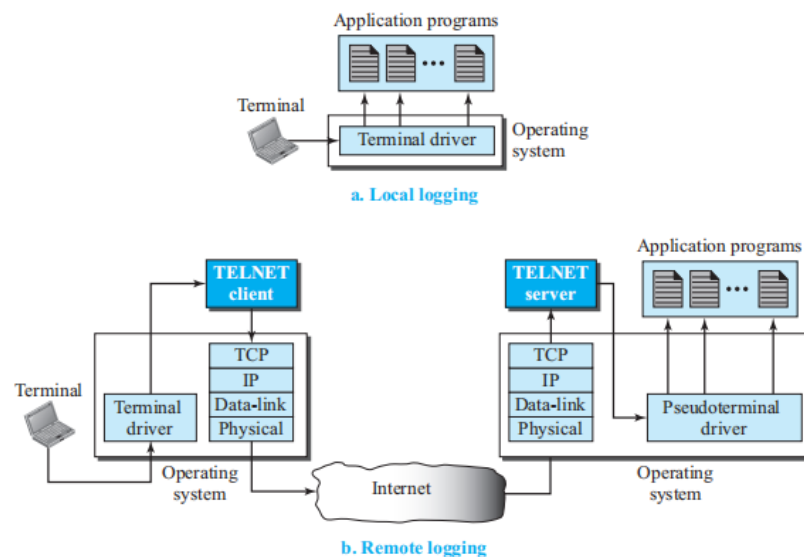
A specific client/server program is designed to provide a service, like FTP for file transfer. However, having a separate client/server pair for each service is not scalable. A solution is to use generic client/server programs that allow users to log into remote computers and use available services. For example, a student can log into a university server using a remote login program to access services like the Java compiler, without needing a separate client for each service. These generic programs are called **remote login applications**.

TELNET is an early remote login protocol that requires a username and password but sends all data, including the password, in plaintext, making it vulnerable to hacking. Due to security risks, TELNET has largely been replaced by **Secure Shell (SSH)**, which provides encrypted communication.

TELNET is briefly discussed because of its historical significance and for comparison with SSH.

1. The simple plaintext architecture of TELNET allows us to explain the issues and challenges related to the concept of remote logging, which is also used in SSH when it serves as a remote logging protocol.
2. Network administrators often use TELNET for diagnostic and debugging purposes.

Local versus Remote Logging



Local logging: When a user logs into their local system, keystrokes are processed by the terminal driver and passed to the operating system, which interprets them and invokes the appropriate application.

Remote logging: When accessing a program on a remote machine, TELNET is used. The user's keystrokes are sent to the terminal driver, but instead of being interpreted locally, they are passed to the TELNET client, which converts them into **Network Virtual Terminal**

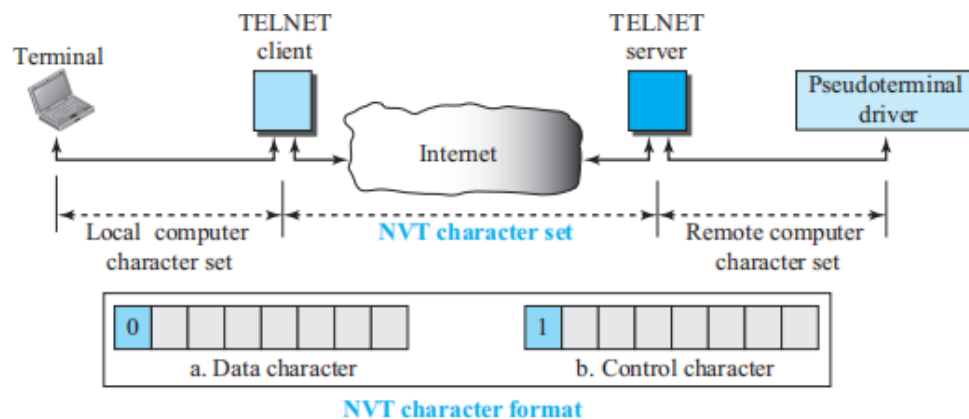
(NVT) characters. These NVT characters are sent through the network to the remote machine.

On the remote machine, the NVT characters are received by the TCP/IP stack and passed to the TELNET server. The server translates the NVT characters to ones the remote system can understand. Since the remote operating system can't directly process TELNET characters, a **pseudoterminal driver** is used to simulate a terminal input, allowing the characters to be passed to the correct application.

Network Virtual Terminal (NVT)

Accessing remote computers can be complex due to the different operating systems using unique character combinations as tokens (e.g., Ctrl+z for DOS and Ctrl+d for UNIX). To handle this complexity, TELNET provides a universal interface called the **Network Virtual Terminal (NVT)** character set.

- **NVT Character Set:** This allows TELNET to translate characters (data or commands) from the local terminal into NVT format for transmission across the network. On the remote side, TELNET translates the NVT characters back into a format understood by the remote computer.
- **Data and Control Characters:** NVT uses two sets of 8-bit characters:
 - **NVT ASCII:** For data, it uses a character set where the lowest 7 bits match US ASCII, and the highest bit is 0.
 - **Control Characters:** For control data, the highest bit is set to 1.



This system ensures that remote systems with different operating environments can communicate effectively.

Options

TELNET lets the client and server negotiate options before or during the use of the service. Options are extra features available to a user with a more sophisticated terminal. Users with simpler terminals can use default features.

User Interface

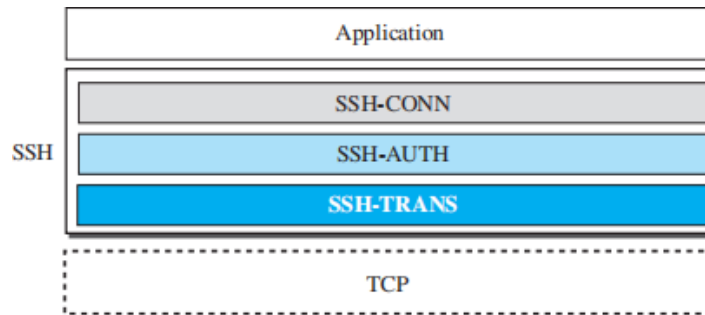
The operating system (UNIX, for example) defines an interface with user-friendly commands.

SECURE SHELL (SSH)

Although **Secure Shell (SSH)** is a secure application program that can be used today for several purposes such as remote logging and file transfer, it was originally designed to replace TELNET. There are two versions of SSH: SSH-1 and SSH-2, which are totally incompatible. The first version, SSH-1, is now deprecated because of security flaws in it.

Components

SSH is an application-layer protocol with three components.



SSH Transport-Layer Protocol (SSH-TRANS)

Since TCP is not a secure transport-layer protocol, **SSH** uses an additional protocol called **SSH-TRANS** to create a secure channel on top of TCP. The process begins with the client and server establishing an insecure connection via TCP. They then exchange security parameters to establish a secure channel using SSH-TRANS.

The services provided by this protocol:

1. Privacy or confidentiality of the message exchanged
2. Data integrity, which means that it is guaranteed that the messages exchanged between the client and server are not changed by an intruder.
3. Server authentication, which means that the client is now sure that the server is the one that it claims to be
4. Compression of the messages, which improves the efficiency of the system and makes attack more difficult.

SSH Authentication Protocol (SSH-AUTH)

After establishing a secure channel and authenticating the server, **SSH** proceeds to authenticate the client. The client sends a request to the server with the username, server name, authentication method, and required data. The server then responds with either a success message, confirming authentication, or a failure message, prompting the client to try again with a new request. This process is similar to client authentication in **SSL**.

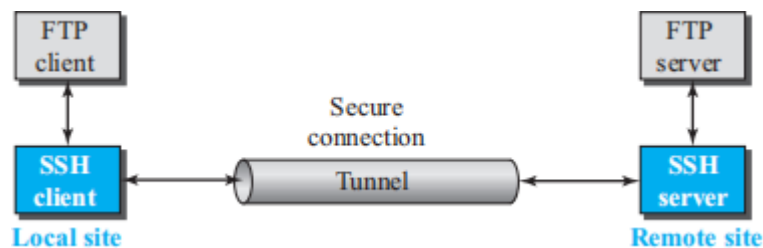
SSH Connection Protocol (SSH-CONN)

Once the secure channel is established and both the server and client are authenticated, **SSH** uses the **SSH-CONN** protocol. One of its key features is **multiplexing**, which allows the client to create multiple logical channels over the secure channel. Each channel can be used for different purposes, such as remote logging or file transfer.

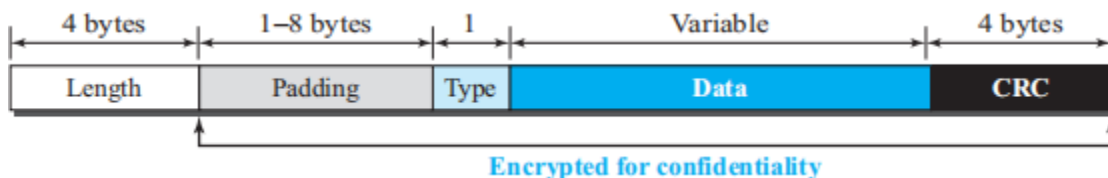
Applications

Although SSH is often thought of as a replacement for TELNET, SSH is, in fact, a general-purpose protocol that provides a secure connection between a client and server.

1. **SSH for Remote Logging** -Several free and commercial applications use SSH for remote logging.
2. **SSH for File Transfer** -One of the application programs that is built on top of SSH for file transfer is the Secure File Transfer Program (sftp).
3. **Port Forwarding** -One of the interesting services provided by the SSH protocol is port forwarding. We can use the secured channels available in SSH to access an application program that does not provide security services. Applications such as TELNET and Simple Mail Transfer Protocol (SMTP), can use the services of the SSH port forwarding mechanism. The SSH port forwarding mechanism creates a tunnel through which the messages belonging to other protocols can travel. For this reason, this mechanism is sometimes referred to as SSH tunnelling.



Format of the SSH Packets

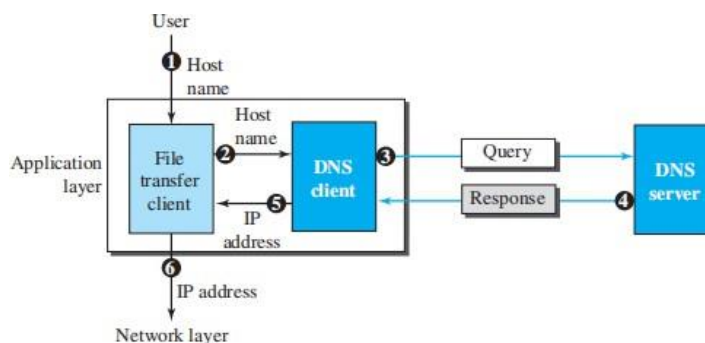


The length field defines the length of the packet but does not include the padding. One to eight bytes of padding is added to the packet to make the attack on the security provision more difficult. The cyclic redundancy check (CRC) field is used for error detection. The type field designates the type of the packet used in different SSH protocols. The data field is the data transferred by the packet in different protocols.

DOMAIN NAME SYSTEM (DNS)

The Domain Name System (DNS) was developed to simplify access to Internet resources by mapping human-friendly names to IP addresses, which are needed for network identification. Similar to how a telephone directory helps map names to numbers, DNS serves as a directory for the Internet, allowing users to remember domain names rather than numeric IP addresses. A central directory for the entire Internet would be impractical due to its vast scale and vulnerability to failures. Instead, DNS information is distributed across multiple servers worldwide. When a host requires name-to-IP mapping, it contacts the nearest DNS server with the necessary information. This distributed structure enhances reliability and efficiency.

Figure shows how TCP/IP uses a DNS client and a DNS server to map a name to an address. A user wants to use a file transfer client to access the corresponding file transfer server running on a remote host. The user knows only the file transfer server name, such as *aflesource.com*. The TCP/IP suite needs the IP address of the file transfer server to make the connection.



The following six steps map the host name to an IP address:

1. The user passes the host name to the file transfer client.
2. The file transfer client passes the host name to the DNS client.
3. Each computer, after being booted, knows the address of one DNS server. The DNS client sends a message to a DNS server with a query that gives the file transfer server name using the known IP address of the DNS server.
4. The DNS server responds with the IP address of the desired file transfer server.
5. The DNS server passes the IP address to the file transfer client.
6. The file transfer client now uses the received IP address to access the file transfer server.

Name Space

To be unambiguous, the names assigned to machines must be carefully selected from a name space with complete control over the binding between the names and IP addresses. A name space that maps each address to a unique name can be organized in two ways: flat or hierarchical

Flat Name Space:

- Each name is a sequence of characters without inherent structure or meaning.
- Names might share common segments, but this doesn't imply any relationship.
- Centralized control is needed to avoid duplication, making a flat name space impractical for large systems like the Internet.

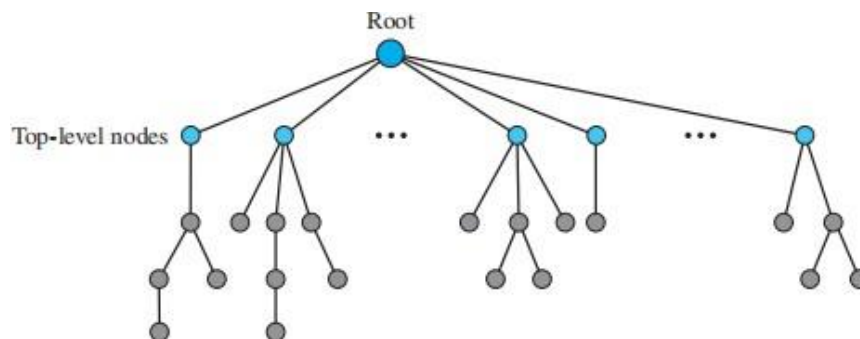
Hierarchical Name Space:

- Names consist of multiple parts, with each part representing a specific element, such as organization type, organization name, or department.
- Decentralization is possible since a central authority assigns only a portion of the name (e.g., organization type and name).
- Organizations manage the remaining parts, allowing unique identifiers even if different organizations use the same host name. For instance, "caesar.first.com" and "caesar.second.com" are distinct due to the organization names, ensuring *uniqueness* in large networks like the Internet.

This hierarchical structure is key to the *Domain Name System (DNS)*, enabling efficient, scalable, and decentralized name resolution across the Internet.

Domain Name Space

To have a hierarchical name space, a **domain name space** was designed. In this design the names are defined in an inverted-tree structure with the root at the top. The tree can have only 128 levels: level 0 (root) to level 127 .

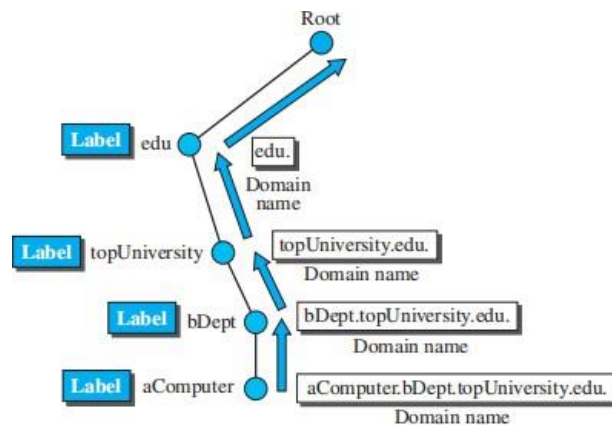


Label

Each node in the tree has a **label**, which is a string with a maximum of 63 characters. The root label is a null string (empty string). DNS requires that children of a node (nodes that branch from the same node) have different labels, which guarantees the uniqueness of the domain names.

Domain Name

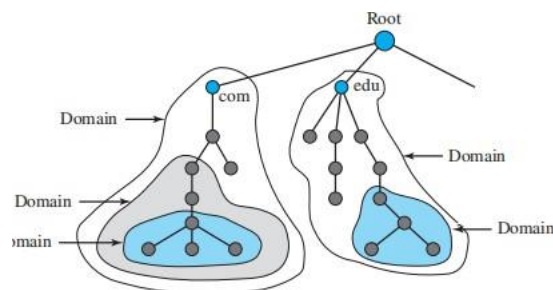
Each node in the tree has a domain name. A full **domain name** is a sequence of labels separated by dots (.). The domain names are always read from the node up to the root. The last label is the label of the root (null).



If a label is terminated by a null string, it is called a **fully qualified domain name (FQDN)**. The name must end with a null label, but because null means nothing, the label ends with a dot. If a label is not terminated by a null string, it is called a **partially qualified domain name (PQDN)**. A PQDN starts from a node, but it does not reach the root. It is used when the name to be resolved belongs to the same site as the client. Here the resolver can supply the missing part, called the *suffix*, to create an FQDN.

Domain

A **domain** is a subtree of the domain name space. The name of the domain is the name of the node at the top of the subtree. Note that a domain may itself be divided into domains.

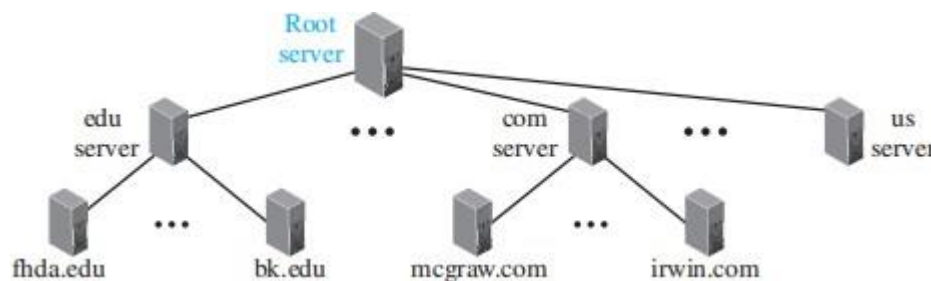


Distribution of Name Space

The information contained in the domain name space must be stored. However, it is very inefficient and also not reliable to have just one computer store such a huge amount of information. It is inefficient because responding to requests from all over the world places a heavy load on the system. It is not reliable because any failure makes the data inaccessible.

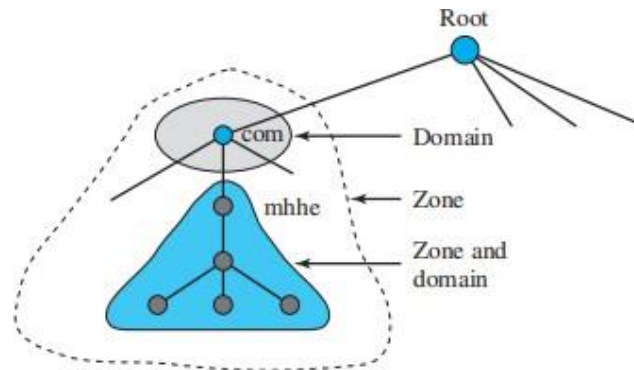
Hierarchy of Name Servers

The solution to these problems is to distribute the information among many computers called **DNS servers**. One way to do this is to divide the whole space into many domains based on the first level. In other words, we let the root stand alone and create as many domains (subtrees) as there are first-level nodes. Because a domain created this way could be very large, DNS allows domains to be divided further into smaller domains (subdomains). Each server can be responsible (authoritative) for either a large or small domain.



Zone

In DNS, a *zone* represents the part of the DNS hierarchy for which a server holds authority, facilitating distributed management. If a server is responsible for an entire domain without subdivisions, then the *zone* and *domain* are effectively the same, and the server keeps all relevant data in a *zone file*. However, if the server divides its domain into smaller *subdomains* and delegates authority for these to other servers, the *zone* differs from the *domain*. In this case, the main server retains responsibility for the broader domain but references lower-level servers that manage data for the specific subdomains. This hierarchical structure enables scalable and efficient distribution of DNS data across multiple servers.



Root Server

A **root server** is a server whose zone consists of the whole tree. A root server usually does not store any information about domains but delegates its authority to other servers, keeping references to those servers. There are several root servers, each covering the whole domain name space. The root servers are distributed all around the world.

Primary and Secondary Servers

DNS defines two types of servers: primary and secondary. A *primary server* is a server that stores a file about the zone for which it is an authority. It is responsible for creating, maintaining, and updating the zone file. It stores the zone file on a local disk.

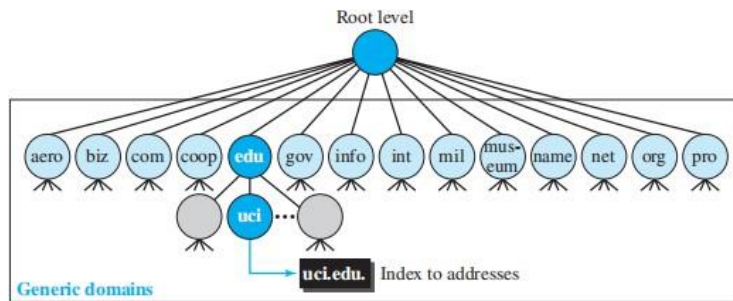
A *secondary server* is a server that transfers the complete information about a zone from another server (primary or secondary) and stores the file on its local disk. The secondary server neither creates nor updates the zone files. If updating is required, it must be done by the primary server, which sends the updated version to the secondary.

DNS in the Internet

DNS is a protocol that can be used in different platforms. In the Internet, the domain name space (tree) was originally divided into three different sections: generic domains, country domains, and the inverse domains. However, due to the rapid growth of the Internet, it became extremely difficult to keep track of the inverse domains, which could be used to find the name of a host when given the IP address. The inverse domains are now deprecated.

Generic Domains

The **generic domains** define registered hosts according to their generic behaviour. Each node in the tree defines a domain, which is an index to the domain name space database.



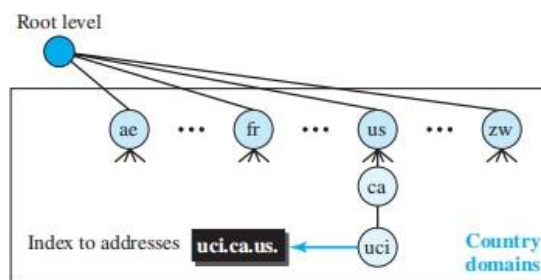
Looking at the tree, we see that the first level in the **generic domains** section allows 14 possible labels. These labels describe the organization types as listed in Table

Table 26.12 Generic domain labels

Label	Description	Label	Description
aero	Airlines and aerospace	int	International organizations
biz	Businesses or firms	mil	Military groups
com	Commercial organizations	museum	Museums
coop	Cooperative organizations	name	Personal names (individuals)
edu	Educational institutions	net	Network support centers
gov	Government institutions	org	Nonprofit organizations
info	Information service providers	pro	Professional organizations

Country Domains

The **country domains** section uses two-character country abbreviations (e.g., us for United States). Second labels can be organizational, or they can be more specific national designations. The United States, for example, uses state abbreviations as a sub division of us (e.g., ca.us.). The address **uci.ca.us.** can be translated to University of California, Irvine, in the state of California in the United States.

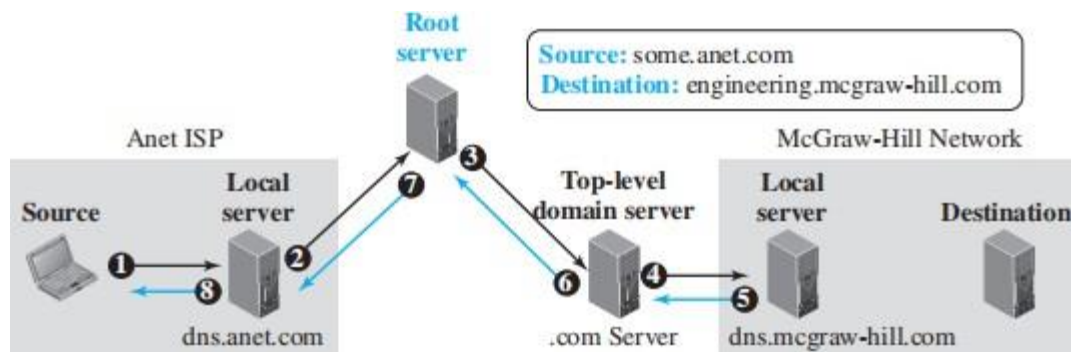


Resolution

Mapping a name to an address is called *name-address resolution*. DNS is designed as a client-server application. A host that needs to map an address to a name or a name to an address calls a DNS client called a **resolver**. The resolver accesses the closest DNS server with a mapping request. If the server has the information, it satisfies the resolver; otherwise, it either refers the resolver to other servers or asks other servers to provide the information. After the resolver receives the mapping, it interprets the response to see if it is a real resolution or an error, and finally delivers the result to the process that requested it. A resolution can be either recursive or iterative.

Recursive Resolution

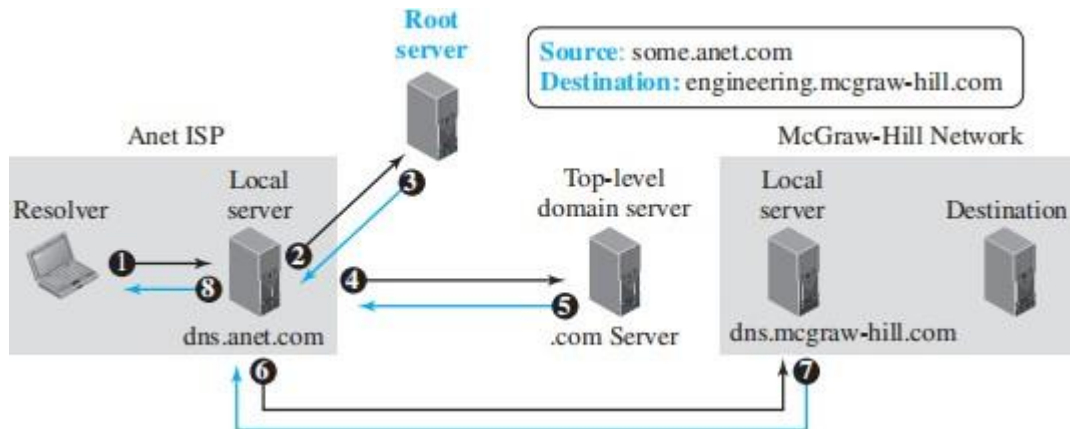
In a *recursive DNS resolution*, an application on a host (e.g., some.anet.com) needs the IP address of another host (e.g., engineering.mcgraw-hill.com) to send a message. The source host's *DNS resolver* (client) queries its local DNS server (dns.anet.com), which doesn't have the address and forwards the request to a root DNS server. The root server, lacking the specific mapping, knows the relevant top-level domain server (e.g., for .com) and forwards the query there. This server, not having the exact address, directs the query to McGraw-Hill's local DNS server (dns.mcgraw-hill.com), which finally provides the destination IP. This IP address is sent back step-by-step—from McGraw-Hill's DNS server to the top-level DNS server, then to the root server, the ISP's DNS server (which caches it), and ultimately back to the source host.



Iterative Resolution

In **iterative resolution**, each server that does not know the mapping sends the IP address of the next server back to the one that requested it. Figure shows the flow of information in an iterative resolution. Normally the iterative resolution takes place between two local servers; the original resolver gets the final answer from the local server. Note that the

messages shown by events 2, 4, and 6 contain the same query. However, the message shown by event 3 contains the IP address of the top-level domain server, the message shown by event 5 contains the IP address of the McGraw-Hill local DNS server, and the message shown by event 7 contains the IP address of the destination. When the Anet local DNS server receives the IP address of the destination, it sends it to the resolver (event 8).



Caching

To improve efficiency, DNS servers use *caching*. When a server requests a name-to-IP mapping from another server and receives a response, it stores the information in its *cache memory*. If the same or another client requests that mapping again, the server can quickly retrieve it from the cache, marked as *unauthoritative* to indicate it's not from the original source.

While caching speeds up DNS resolution, it can lead to issues if a mapping becomes outdated. To manage this, each authoritative server includes a *time-to-live (TTL)* value with the mapping, specifying how long the receiving server can cache it before it becomes invalid. DNS servers also keep a TTL counter for each cached mapping, regularly checking and purging entries with expired TTLs to maintain up-to-date data.

Resource Records

The zone information associated with a server is implemented as a set of *resource records*. A *resource record* is a 5-tuple structure, as shown below:

(Domain Name, Type, Class, TTL, Value)

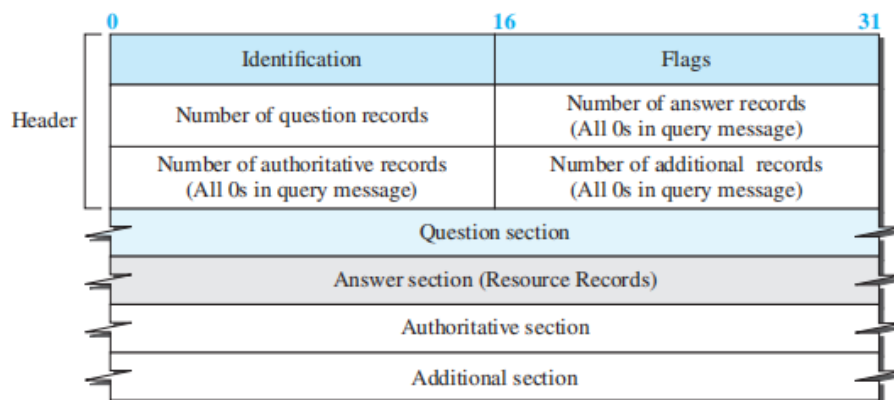
The domain name field is what identifies the resource record. The value defines the information kept about the domain name. The TTL defines the number of seconds for

which the information is valid. The class defines the type of network; we are only interested in the class IN (Internet). The type defines how the value should be interpreted. Table lists the common types and how the value is interpreted for each type.

Type	Interpretation of value
A	A 32-bit IPv4 address (see Chapter 18)
NS	Identifies the authoritative servers for a zone
CNAME	Defines an alias for the official name of a host
SOA	Marks the beginning of a zone
MX	Redirects mail to a mail server
AAAA	An IPv6 address (see Chapter 22)

DNS Messages

To retrieve information about hosts, DNS uses two types of messages: *query* and *response*. Both types have the same format.



Note:

The query message contains only the question section. The response message includes the question section, the answer section, and possibly two other sections.

- The identification field is used by the client to match the response with the query.
- The flag field defines whether the message is a query or response. It also includes status of error.
- The next four fields in the header define the number of each record type in the message.
- The question section consists of one or more question records. It is present in both query and response messages.
- The answer section consists of one or more resource records. It is present only in response messages.

- The authoritative section gives information (domain name) about one or more authoritative servers for the query.
- The additional information section provides additional information that may help the resolver.

Registrars

New domains are added to the DNS through *registrars*, commercial entities accredited by *ICANN*. Registrars verify that the requested domain name is unique, enter it into the DNS database, and charge a fee. Today, there are many registrars; their names and addresses can be found at

<http://www.intenic.net>

To register, the organization needs to give the name of its server and the IP address of the server. For example, a new commercial organization named wonderful with a server named ws and IP address 200.200.200.5 needs to give the following information to one of the registrars:

Domain name: ws.wonderful.com **IP address:** 200.200.200.5

DDNS

The *Dynamic Domain Name System (DDNS)* was created to automate DNS updates, accommodating frequent address changes on today's Internet. Instead of manual updates, DDNS allows changes like adding or removing hosts or updating IP addresses to be made dynamically. Typically, when a new name-to-address binding is established (e.g., via DHCP), it's sent to the *primary DNS server*, which updates the *zone*. *Secondary servers* learn of these changes through *active* or *passive notifications*—either the primary server sends an alert, or secondary servers periodically check for updates. After notification, secondary servers perform a *zone transfer* to receive updated information. To prevent unauthorized changes, DDNS can implement an *authentication mechanism*.

Security of DNS

DNS is one of the most important systems in the Internet infrastructure; it provides crucial services to Internet users. Applications such as Web access or e-mail are heavily dependent on the proper operation of DNS. DNS can be attacked in several ways including:

1. The attacker may read the response of a DNS server to find the nature or names of sites the user mostly accesses. This type of information can be used to find the user's profile. To prevent this attack, DNS messages need to be confidential

2. The attacker may intercept the response of a DNS server and change it or create a totally new bogus response to direct the user to the site or domain the attacker wishes the user to access. This type of attack can be prevented using message origin authentication and message integrity
3. The attacker may flood the DNS server to overwhelm it or eventually crash it. This type of attack can be prevented using the provision against denial-of-service attack.

To protect DNS, IETF has devised a technology named *DNS Security* (DNSSEC) that provides message origin authentication and message integrity using a security service called *digital signature*. DNSSEC, however, does not provide confidentiality for the DNS messages. There is no specific protection against the denial-of service attack in the specification of DNSSEC. However, the caching system protects the upper-level servers against this attack to some extent.