Cndc (4361101) - Problem Solver

Antigravity AI

June 4, 2026

Cndc

First Edition: 2026

Author: Antigravity AI
Website: milav.in
YouTube: @milav.dabgar

To my students,
who constantly inspire me to find new analogies,
break down complex theories, and make learning
an engineered science.

Contents

Preface

About this Book

Welcome to **Computer Networks**! This textbook is meticulously designed to serve as a comprehensive problem solver and study companion. It provides a robust and intuitive foundation in the core principles of this subject, bridging the gap between theoretical models and practical applications. A unique feature of this book is its focus on decoding complex concepts using clear, step-by-step methodologies and real-world analogies.

Target Audience

This book is intended for students and professionals looking to master the concepts of Computer Networks. Whether you are revising for exams, seeking to solidify your fundamental understanding, or looking for practical examples to clarify abstract concepts, this book has been structured to support your learning journey. It serves as an accessible guide designed to remove the fear of complex topics, replacing it with an appreciation for the underlying mechanics.

Scope and Coverage

The coverage is divided logically to follow standard academic curriculums. Each chapter focuses on key topics, breaking them down into digestible sections:

- **Theoretical Insights** – Core concepts explained intuitively.
- **Method Blocks** – Standardized, step-by-step procedures for solving specific types of problems.
- **Worked Examples** – Fully solved problems that apply the methods in practice.

We hope this resource proves invaluable in your studies and helps you achieve excellence in **Computer Networks**. Happy learning!

Acknowledgments

Writing a comprehensive book that connects the fundamental forces of Chemistry with practical engineering applications requires more than just academic knowledge—it requires a community of support. I would like to express my sincere gratitude to everyone who contributed to the realization of this project.

Specially, I extend my heartfelt gratitude to the **Department of Technical Education (DTE), Government of Gujarat**, and **Government Polytechnic, Palanpur**, for providing an environment that fosters innovation, academic rigor, and continuous professional development.

I am deeply thankful to my family for their unwavering patience and support during the countless hours spent researching, formatting, and refining these chapters.

Finally, my deepest appreciation goes to my students. Your curiosity, challenging questions, and continuous feedback are the true driving force behind the unique analogies and streamlined structure of this book. This textbook was engineered for you.

Antigravity AI

About the Author

Milav Jayeshkumar Dabgar is an engineering educator and R&D professional with over 9 years of experience in electronics hardware development, embedded systems, AI/ML, and full-stack software engineering. He currently serves as a **Lecturer (Class - II) & Innovation Leader** at Government Polytechnic, Palanpur, under the Education Department of the Government of Gujarat.

Milav holds a Master of Engineering (ME) in Communication Systems from L.D. College of Engineering and a Bachelor of Engineering (BE) in Electronics & Communication. He is also currently pursuing a **BS in Data Science and Applications from IIT Madras**, where he has completed both the **Diploma in Programming** and the **Diploma in Data Science** with distinction.

Most recently, he completed the **AICTE QIP PG Certificate Program in Deep Learning from SVNIT Surat** with a **perfect 10/10 CGPA**, topping his batch.

A dedicated mentor, Milav has guided numerous student teams in securing over **INR 45 lakhs** in innovation funding, including INR 25 lakhs from Shark Tank India and INR 20 lakhs through iHub Gujarat, resulting in two student patents. He is recognized as an **NPTEL Evangelist** and **NPTEL Discipline Star** in Computer Science, reflecting his commitment to continuous learning and academic excellence.

His technical expertise spans from embedded systems (8051, STM32, Arduino) to modern web technologies (Next.js, Python, PostgreSQL) and Data Science (TensorFlow, PyTorch). Through this textbook, he aims to simplify complex Engineering Chemistry concepts by integrating relatable, real-world analogies drawn from modern tech and engineering landscapes.

Milav is also an active content creator, running a popular **YouTube channel** (@milav.dabgar) where he shares technical tutorials and academic resources. He maintains a personal blog and resource hub at milav.in and can be reached for professional collaborations on LinkedIn.

CHAPTER 1

Fundamentals of Networking and Data Communication

This chapter focuses on the fundamental computational aspects of data communication and computer networks. We will explore network topology link calculations, data rate limits using Nyquist and Shannon theorems, signal attenuation, and network latency components.

1.1 Network Topology Link Calculations

Methodology:

Mesh Topology Links and Ports In a fully connected mesh topology, every device has a dedicated point-to-point link to every other device. For a network of n devices:

1. **Number of Links:** Calculate the total number of physical channels using the combination formula $L = \frac{n(n-1)}{2}$.
2. **Number of Ports:** Calculate the number of input/output (I/O) ports required per device as $P = n - 1$.

Worked Example:

Calculating Links in a Mesh Network A fully connected mesh network consists of 8 devices. Calculate the total number of cable links required and the number of ports needed for each device.
Solution:

1. **Identify the number of devices:** $n = 8$
2. **Calculate total number of links:**

$$L = \frac{n(n-1)}{2} = \frac{8(8-1)}{2} = \frac{8 \times 7}{2} = 28 \text{ links}$$

3. **Calculate the number of ports per device:**

$$P = n - 1 = 8 - 1 = 7 \text{ ports}$$

1.2 Data Rate Limits

Data rate refers to the speed at which data is successfully transferred through a communication channel. Two fundamental theorems determine the maximum data rate of a channel: Nyquist Bit Rate for a noiseless channel and Shannon Capacity for a noisy channel.

Methodology:

Nyquist Bit Rate for Noiseless Channel The Nyquist theorem defines the maximum bit rate for a noiseless channel.

1. **Identify the parameters:** Bandwidth B (in Hz) and the number of signal levels L.
2. **Apply the formula:**
$$\text{BitRate} = 2 \times B \times \log_2(L) \text{ bps}$$
where bps is bits per second.

Worked Example:

Calculating Nyquist Bit Rate Consider a noiseless channel with a bandwidth of 3000 Hz transmitting a signal with 4 signal levels. Determine the maximum bit rate.
Solution:

1. **Identify given parameters:** Bandwidth $B = 3000$ Hz
Signal levels $L = 4$
2. **Calculate the bit rate:**
$$\text{BitRate} = 2 \times B \times \log_2(L)$$
$$\text{BitRate} = 2 \times 3000 \times \log_2(4)$$
Since $\log_2(4) = 2$:
$$\text{BitRate} = 6000 \times 2 = 12000 \text{ bps or } 12 \text{ kbps}$$

Methodology:

Shannon Capacity for Noisy Channel The Shannon capacity formula determines the theoretical highest data rate for a noisy channel.

1. **Identify the parameters:** Bandwidth B (in Hz) and the Signal-to-Noise Ratio (SNR).
2. **Convert SNR to a ratio if given in dB:**
$$\text{SNR}_{\text{dB}} = 10 \log_{10}(\text{SNR}) \implies \text{SNR} = 10^{\frac{\text{SNR}_{\text{dB}}}{10}}$$
3. **Apply the capacity formula:**
$$C = B \times \log_2(1 + \text{SNR}) \text{ bps}$$

Worked Example:

Calculating Shannon Capacity A noisy channel has a bandwidth of 1 MHz and a Signal-to-Noise Ratio (SNR) of 63. Determine the maximum channel capacity.
Solution:

1. **Identify given parameters:** Bandwidth $B = 1,000,000$ Hz
SNR $= 63$
2. **Apply the Shannon formula:**
$$C = B \times \log_2(1 + \text{SNR})$$
$$C = 10^6 \times \log_2(1 + 63) = 10^6 \times \log_2(64)$$
Since $\log_2(64) = 6$:
$$C = 10^6 \times 6 = 6,000,000 \text{ bps} = 6 \text{ Mbps}$$

Worked Example:

Shannon Capacity with SNR in Decibels A telephone line normally has a bandwidth of 3000 Hz assigned for data communications. The signal-to-noise ratio is usually 3162. Determine the capacity of this channel.
Solution:

1. **Identify given parameters:** Bandwidth $B = 3000$ Hz
   SNR $= 3162$
2. **Calculate Capacity:**

$$C = B \log_2(1 + \text{SNR}) = 3000 \log_2(1 + 3162) = 3000 \log_2(3163)$$

$$C \approx 3000 \times 11.62 \approx 34860 \text{ bps}$$

1.3 Transmission Impairment

Signals lose power as they travel through a medium. This loss of energy is called attenuation. To show that a signal has lost or gained strength, engineers use the unit of decibel (dB).

Methodology:

Calculating Signal Attenuation in Decibels The decibel measures the relative strengths of two signals or one signal at two different points.

1. **Identify the power levels:** Initial power P_1 at the source and final power P_2 at the destination.
2. **Calculate the change in dB:**

$$\text{dB} = 10 \log_{10}\left(\frac{P_2}{P_1}\right)$$

3. **Interpret the result:** A negative dB value indicates attenuation (loss of power), while a positive dB value indicates amplification (gain).

Worked Example:

Calculating Attenuation of a Signal Suppose a signal travels through a transmission medium and its power is reduced to one-half of its original value. Calculate the attenuation in decibels.
Solution:

1. **Determine the relationship between P_1 and P_2:**

$$P_2 = \frac{1}{2}P_1 \implies \frac{P_2}{P_1} = 0.5$$

2. **Calculate the decibel value:**

$$\text{dB} = 10 \log_{10}(0.5)$$

$$\text{dB} = 10 \times (-0.301) \approx -3.01 \text{ dB}$$

   A loss of approximately 3 dB indicates the signal has lost half its power.

1.4 Network Latency Components

Latency or delay defines how long it takes for an entire message to completely arrive at the destination from the time the first bit is sent out from the source. Latency is made of four components: propagation time, transmission time, queuing time and processing delay.

Methodology:

Calculating Propagation and Transmission Delay The two major computational components of latency in a basic link are propagation delay and transmission delay.

1. **Calculate Propagation Delay:** The time required for a bit to travel from the source to the destination.
$$\text{Propagation Delay} = \frac{\text{Distance}}{\text{Propagation Speed}}$$
2. **Calculate Transmission Delay:** The time required to put the entire message onto the transmission medium.
$$\text{Transmission Delay} = \frac{\text{Message Size (in bits)}}{\text{Bandwidth (in bps)}}$$
3. **Calculate Total Delay:** Assuming processing and queuing delays are zero.
$$\text{Total Delay} = \text{Propagation Delay} + \text{Transmission Delay}$$

Worked Example:

Total Delay in a Point to Point Link A message of size 5 MB is sent across a network link of distance 10000 km. The bandwidth of the link is 1 Gbps and the propagation speed is 2×10^8 m/s. Calculate the propagation time and transmission time for the message.
Solution:

1. **Convert units to standard forms:** Message Size = 5 MB = $5 \times 10^6 \times 8$ bits = 40×10^6 bits
Distance = 10000 km = 10000×10^3 meters = 10^7 m
Bandwidth = 1 Gbps = 10^9 bps
2. **Calculate Propagation Delay:**
$$\text{Propagation Delay} = \frac{\text{Distance}}{\text{Speed}} = \frac{10^7 \text{ m}}{2 \times 10^8 \text{ m/s}} = 0.05 \text{ seconds} = 50 \text{ ms}$$
3. **Calculate Transmission Delay:**
$$\text{Transmission Delay} = \frac{\text{Message Size}}{\text{Bandwidth}} = \frac{40 \times 10^6 \text{ bits}}{10^9 \text{ bps}} = 0.04 \text{ seconds} = 40 \text{ ms}$$
4. **Total Delay:**
$$\text{Total Delay} = 50 \text{ ms} + 40 \text{ ms} = 90 \text{ ms}$$

CHAPTER 2
Network Devices

This chapter focuses on the computational and analytical techniques associated with network devices at various OSI layers. We will explore collision and broadcast domain calculations, switch MAC address table operations, Longest Prefix Match routing algorithms, and Spanning Tree Protocol root bridge elections.

2.1 Collision and Broadcast Domains

Understanding how different devices segment a network is critical for performance evaluation and security analysis.

Methodology:

Domain Calculation Rules To calculate the total number of collision and broadcast domains in a network architecture, apply the following device-specific rules:

1. **Repeater/Hub (Layer 1):** Does not segment collision or broadcast domains. All connected ports belong to a single shared collision domain and a single broadcast domain.
2. **Bridge/Switch (Layer 2):** Breaks collision domains. Each individual active port on a switch forms its own separate collision domain. However, switches do not break broadcast domains; all ports belong to a single broadcast domain.
3. **Router (Layer 3):** Breaks both collision and broadcast domains. Each router interface forms a separate collision domain and a separate broadcast domain.

Worked Example:

Topology Domain Count **Problem:** Consider a network topology containing 1 Router, 2 Switches and 1 Hub.

- Router Port 1 connects to Switch A.
- Router Port 2 connects to Hub A.
- Switch A has 4 PCs directly connected to its ports.
- Hub A has 3 PCs connected to it and also connects to Switch B via an uplink.
- Switch B has 2 PCs connected to its ports.

Calculate the total number of collision domains and broadcast domains in this network.
Solution:
Step 1: Calculate Broadcast Domains (Layer 3 boundaries) Routers segment broadcast domains. The router has 2 active interfaces in this topology.

- Broadcast Domain 1: Emanates from Router Port 1, encompassing Switch A and its 4 PCs.
- Broadcast Domain 2: Emanates from Router Port 2, encompassing Hub A, its 3 PCs, Switch B, and its 2 PCs.

Total Broadcast Domains = 2.
Step 2: Calculate Collision Domains (Layer 2 and 3 boundaries) Examine each broadcast domain independently to count Layer 2 segments.

- **Network segment on Router Port 1:** Switch A connects to the router (1 port) and to 4 PCs (4 ports). Each active switch port represents a unique collision domain. Total = $1 + 4 = 5$ collision domains.
- **Network segment on Router Port 2:** The Hub acts as a single large collision domain. The router port, the 3 PCs, and the uplink to Switch B all share this single domain. Total = 1 collision domain.
- **Switch B (connected via Hub):** Switch B provides 2 active ports connecting to PCs. Each is a separate collision domain. Total = 2 collision domains.

Total Collision Domains = $5 + 1 + 2 = 8$.

2.2 Switch Forwarding and MAC Learning

A Layer 2 switch makes isolated forwarding decisions by building and maintaining a MAC Address Table (Content-Addressable Memory or CAM table).

Methodology:

MAC Learning and Forwarding Algorithm When a switch receives a frame on a specific ingress port, it executes the following logic sequentially:

1. **Source Address Learning:** Extract the Source MAC address. If it is not in the MAC table, insert it along with the ingress port number and reset the aging timer. If it is already present, update the entry and reset the aging timer.
2. **Destination Address Lookup:** Extract the Destination MAC address.
3. **Decision Phase:**
   - **Flood:** If the Destination MAC is a Broadcast address (FF:FF:FF:FF:FF:FF) or is NOT found in the MAC table (Unknown Unicast), forward the frame out of all active ports EXCEPT the ingress port.
   - **Forward:** If the Destination MAC is found in the table and maps to a port different from the ingress port, forward the frame ONLY out of that specific mapped port.
   - **Filter:** If the Destination MAC is found in the table but maps to the SAME port it was received on, discard (filter) the frame.

Worked Example:

Switch Operation Trace **Problem:** A 4-port switch is freshly powered on (its MAC table is empty). PC A (MAC AA) is on Port 1, PC B (MAC BB) is on Port 2, and PC C (MAC CC) is on Port 3. Trace the MAC table updates and the switch's forwarding actions for the following sequential frame transmissions:

1. PC A sends a frame to PC B.
2. PC B sends a frame to PC A.
3. PC C sends a frame to PC A.
4. PC A sends a frame to PC C.

Solution:
Transmission 1: Source AA to Destination BB arriving on Port 1

- **Learning:** Switch learns MAC AA is on Port 1.
- **Lookup:** Destination BB is NOT in the table.
- **Action: Flood** the frame out of Ports 2, 3, and 4.

Transmission 2: Source BB to Destination AA arriving on Port 2

- **Learning:** Switch learns MAC BB is on Port 2.
- **Lookup:** Destination AA IS in the table (mapped to Port 1).
- **Action: Forward** the frame ONLY out of Port 1.

Transmission 3: Source CC to Destination AA arriving on Port 3

- **Learning:** Switch learns MAC CC is on Port 3.
- **Lookup:** Destination AA IS in the table (mapped to Port 1).
- **Action: Forward** the frame ONLY out of Port 1.

Transmission 4: Source AA to Destination CC arriving on Port 1

- **Learning:** Switch sees MAC AA is already on Port 1. Aging timer refreshed.
- **Lookup:** Destination CC IS in the table (mapped to Port 3).
- **Action: Forward** the frame ONLY out of Port 3.

2.3 Routing Table Lookups

Network routers rely on the Longest Prefix Match (LPM) algorithm to determine the optimal outbound path for an incoming IP packet when multiple routing entries might apply.

Methodology:

Longest Prefix Match Algorithm Given an incoming IP packet and a router's routing table containing Destination Subnets and their respective Masks:

1. **Subnet Calculation:** Iterate over each entry in the routing table. Perform a bitwise AND operation between the packet's Destination IP address and the Subnet Mask of the entry.
2. **Match Verification:** Compare the result of the bitwise AND operation with the Network Address of that routing table entry. If they are identical, the route is marked as a valid candidate.
3. **Prefix Length Selection:** Among all valid candidate routes, select the one possessing the longest subnet mask (the largest /CIDR prefix length or most contiguous 1-bits). This represents the most specific route.
4. **Default Route:** If no specific route matches, check for a default route (0.0.0.0/0).
5. **Forwarding:** Forward the packet to the Outgoing Interface or Next Hop indicated by the selected best match.

Worked Example:

IP Route Selection **Problem:** A router receives a packet with the destination IP address `192.168.1.130`. Its routing table contains the following entries:

Network ID	Subnet Mask (CIDR)	Next Hop
192.168.1.0	255.255.255.0 (/24)	Interface A
192.168.1.128	255.255.255.128 (/25)	Interface B
192.168.1.128	255.255.255.192 (/26)	Interface C
0.0.0.0	0.0.0.0 (/0)	Interface D

Determine the selected outgoing interface for this packet.
Solution: We must evaluate the destination IP `192.168.1.130` against every route. For calculations in the 4th octet, `130` in binary is `10000010`.

1. **Test Entry 1: 192.168.1.0/24 (Mask: 255.255.255.0)**
   - Bitwise AND: `192.168.1.130` AND `255.255.255.0` = `192.168.1.0`.
   - Match? Yes. (Prefix length = 24).
2. **Test Entry 2: 192.168.1.128/25 (Mask: 255.255.255.128)**
   - Bitwise AND in 4th octet: `10000010` (130) AND `10000000` (128) = `10000000` (128).
   - Result IP: `192.168.1.128`.
   - Match? Yes. (Prefix length = 25).
3. **Test Entry 3: 192.168.1.128/26 (Mask: 255.255.255.192)**
   - Bitwise AND in 4th octet: `10000010` (130) AND `11000000` (192) = `10000000` (128).
   - Result IP: `192.168.1.128`.
   - Match? Yes. (Prefix length = 26).
4. **Test Default Route: 0.0.0.0/0**
   - Match? Yes, matches all IPs. (Prefix length = 0).

Selection: The valid candidates are /24, /25, /26, and /0. The longest prefix is /26 (Entry 3). Therefore, the router forwards the packet out of **Interface C**.

2.4 Spanning Tree Protocol Selection

Redundant Layer 2 links create debilitating broadcast storms. The Spanning Tree Protocol (STP) computationally resolves this by electing a central root and blocking redundant port pathways.

Methodology:

Root Bridge Election The algorithm starts by electing a Root Bridge, serving as the central reference node for the spanning tree topology:

1. **Bridge ID Construction:** Every switch creates an 8-byte Bridge ID, concatenating a 2-byte Bridge Priority and its 6-byte base MAC Address.
2. **Priority Evaluation:** Switches exchange Bridge Protocol Data Units (BPDUs). Compare the Bridge Priority of all active switches. The switch with the mathematically lowest priority becomes the Root Bridge. (Default priority is generally 32768).
3. **MAC Address Tie-Breaker:** If multiple switches share the same lowest priority, compare their MAC Addresses sequentially from left to right. The switch with the numerically lowest MAC Address (assessed in hexadecimal) wins the election and becomes the Root Bridge.

Worked Example:

Determining the Root Bridge **Problem:** Three switches form a redundant ring. Their STP parameters are defined below:

- Switch A: Priority 32768, MAC `00:11:22:AA:BB:CC`
- Switch B: Priority 28672, MAC `00:11:22:DD:EE:FF`
- Switch C: Priority 28672, MAC `00:11:22:99:88:77`

Execute the STP election algorithm to determine the Root Bridge.
Solution:
Step 1: Compare Bridge Priorities

- Priority A = 32768
- Priority B = 28672
- Priority C = 28672

Switch A has a larger priority value (32768 > 28672), so it is eliminated. Switches B and C tie for the lowest priority.
Step 2: Apply MAC Address Tie-Breaker Compare the MAC addresses of the tied switches, B and C.

- Switch B: `00:11:22:DD:EE:FF`
- Switch C: `00:11:22:99:88:77`

The first three hex octets (`00:11:22`) are identical. Moving to the fourth octet:

- Switch B: Hex `DD` (Decimal 221)
- Switch C: Hex `99` (Decimal 153)

Because Hex `99` < Hex `DD`, Switch C holds the lower MAC address.
Conclusion: Switch C is elected as the Root Bridge for this network.

CHAPTER 3

Hardware Layer: Network Protocols and Computations

The Hardware Layer (spanning Physical, Data Link, and Network layers in this context) involves fundamental algorithms for addressing, error control, flow control, and routing. This chapter covers the computational techniques essential for designing and managing these layer protocols.

3.1 IP Addressing and Subnetting

IP addressing assigns logical addresses to devices on a network. We will examine both Classful and Classless (CIDR) addressing schemes.

Methodology:

Classful IP Addressing In Classful addressing, an IPv4 address is divided into classes based on the first octet.

1. **Class A:** First octet 1 to 126. Default Mask: `255.0.0.0`
2. **Class B:** First octet 128 to 191. Default Mask: `255.255.0.0`
3. **Class C:** First octet 192 to 223. Default Mask: `255.255.255.0`
4. **Network Address:** Bitwise AND of the IP address and the Default Subnet Mask.
5. **Broadcast Address:** Set all Host ID bits of the IP address to 1.

Worked Example:

Identify Network and Broadcast Addresses Given the IP address 140.25.60.100, find its class, network address, and broadcast address.

Step 1: Identify the Class

The first octet is 140. Since 140 lies between 128 and 191, this is a **Class B** address.

Step 2: Determine Default Subnet Mask

The default mask for Class B is `255.255.0.0`.

Step 3: Calculate Network Address

Perform bitwise AND of IP and Mask:

$$\begin{array}{l} 140.25.60.100 \\ \text{AND } 255.255.0.0 \\ \hline 140.25.0.0 \end{array}$$

The Network Address is `140.25.0.0`.

Step 4: Calculate Broadcast Address

For Class B, the last two octets represent the Host ID. Set these host bits to 1 (which is 255 in decimal): Broadcast Address = `140.25.255.255`.

Methodology:

CIDR Subnetting and Host Calculation Classless Inter-Domain Routing (CIDR) uses a prefix length notation, e.g., `/n`, where n is the number of network bits.

1. **Subnet Mask:** The first n bits are 1s, and the remaining $(32 - n)$ bits are 0s.
2. **Number of Hosts per Subnet:** $2^{32-n} - 2$ (subtract 2 for network and broadcast addresses).
3. **Block Size (Magic Number):** $256 -$ interesting octet value in the mask. The interesting octet is the one containing both 1s and 0s.
4. **Network Address:** Find the multiple of the block size that is less than or equal to the IP's value in the interesting octet.
5. **Broadcast Address:** The network address of the *next* subnet minus 1.

Worked Example:

CIDR Network Address and Valid Host Range Given the IP address 192.168.10.75/28, find the subnet mask, network address, broadcast address, and the range of valid host IP addresses.
Step 1: Determine the Subnet Mask
The prefix is /28. This means 28 bits of 1s and 4 bits of 0s.
In binary: `11111111.11111111.11111111.11110000`
In decimal: `255.255.255.240`
Step 2: Find the Block Size
The interesting octet is the fourth octet (value is 240).
Block Size $= 256 - 240 = 16$.
Step 3: Calculate the Network Address
The multiples of the block size (16) are: $0, 16, 32, 48, 64, 80, \ldots$
The IP's 4th octet is 75. The largest multiple less than or equal to 75 is 64.
Network Address: `192.168.10.64`
Step 4: Calculate the Broadcast Address
The next subnet starts at $64 + 16 = 80$.
The broadcast address is one less than the next subnet's network address $(80 - 1 = 79)$.
Broadcast Address: `192.168.10.79`
Step 5: Determine the Valid Host Range
First valid host: Network Address + 1 = `192.168.10.65`
Last valid host: Broadcast Address - 1 = `192.168.10.78`
Range: `192.168.10.65` to `192.168.10.78`

3.2 Error Detection: Cyclic Redundancy Check

CRC is used at the Data Link Layer to detect bit errors.

Methodology:

Cyclic Redundancy Check Calculation

1. **Given Data:** Let the data word be $D(x)$ and the generator polynomial be $G(x)$.
2. **Append Zeros:** Let k be the degree of the generator polynomial $G(x)$ (i.e., length of $G(x)$ minus 1). Append k zeros to the right of the data $D(x)$.
3. **Binary Division:** Perform modulo-2 division (XOR operation) of the padded data by the generator $G(x)$.
4. **Remainder (CRC):** The remainder obtained from the division is the CRC bits.

5. **Codeword:** Replace the appended zeros with the CRC to form the transmitted codeword.

Worked Example:

Compute CRC Codeword Find the transmitted codeword if the data word is 1101011011 and the generator polynomial is $x^4 + x + 1$.
Step 1: Identify Generator and Append Zeros
Generator polynomial $G(x) = x^4 + x + 1$ corresponds to the binary string 10011.
The degree of $G(x)$ is 4.
Original Data = 1101011011.
Appended Data = 11010110110000.
Step 2: Perform Modulo-2 Division
We divide the appended data by 10011 using XOR.
Detailed XOR steps:

1. 11010 XOR 10011 = 01001. Bring down 1 → 10011.
2. 10011 XOR 10011 = 00000. Bring down next bits until length is 5: 10110.
3. 10110 XOR 10011 = 00101. Bring down 0 → 01010 (need one more 0) → 10100.
4. 10100 XOR 10011 = 00111. Length is now 4.

The remainder (CRC) is 1110.
Step 3: Form Transmitted Codeword
Codeword = Data + CRC = 11010110111110.

3.3 Flow Control Protocols

Flow control prevents a fast sender from overwhelming a slow receiver. Key computational metrics are efficiency (utilization) and throughput.

Methodology:

Efficiency of Flow Control Protocols Let:

- L = Length of the frame (bits)
- R = Bandwidth or Data rate (bps)
- $T_t = L/R$ = Transmission Time
- T_p = Propagation Delay
- $a = T_p/T_t$
- W = Window Size (number of frames)

Formulas:

1. **Stop-and-Wait ARQ:** Efficiency $\eta = \frac{1}{1+2a}$
   Throughput $= \eta \times R$
2. **Go-Back-N and Selective Repeat ARQ:** Efficiency $\eta = \frac{W}{1+2a}$ (if $W < 1 + 2a$)
   Efficiency $\eta = 1$ (if $W \geq 1 + 2a$)
3. **Window Size constraints:**
   For Go-Back-N: $W \leq 2^m - 1$
   For Selective Repeat: $W \leq 2^{m-1}$
   where m is the number of bits in the sequence number.

Worked Example:

Calculate Stop-and-Wait Efficiency A channel has a bit rate of 4 Kbps and a propagation delay of 20 ms. For what range of frame sizes does Stop-and-Wait give an efficiency of at least 50%?

Step 1: Identify Given Values

$R = 4000$ bps

$T_p = 20$ ms $= 0.02$ s

Target Efficiency $\eta \geq 0.5$

Step 2: Apply the Efficiency Formula

$\eta = \frac{1}{1+2a} \geq 0.5$

$1 \geq 0.5(1+2a) \implies 1 \geq 0.5 + a \implies 0.5 \geq a \implies a \leq 0.5$

Step 3: Substitute the definition of a

$a = \frac{T_p}{T_t} \leq 0.5$

$T_t \geq \frac{T_p}{0.5} = \frac{0.02}{0.5} = 0.04$ s

Step 4: Calculate the Frame Size L

$T_t = \frac{L}{R} \geq 0.04$

$L \geq 0.04 \times 4000 = 160$ bits.

The frame size must be at least 160 bits.

3.4 CSMACD Protocol

In Carrier Sense Multiple Access with Collision Detection, the sender must still be transmitting the frame when the collision signal reaches it.

Methodology:

Minimum Frame Size in CSMACD To detect collisions properly, the transmission time must be at least twice the propagation delay.

1. **Condition:** $T_t \geq 2 \times T_p$
2. **Formula:** $\frac{L_{min}}{R} = 2 \times \frac{d}{v}$ where L_{min} is the minimum frame size, R is bandwidth, d is distance, and v is propagation speed.
3. **Minimum Frame Size:** $L_{min} = 2 \times T_p \times R$

Worked Example:

Determine Minimum Frame Length A network using CSMA/CD has a bandwidth of 10 Mbps. The maximum length of the cable is 2500 m and the signal propagation speed is 2×10^8 m/s. Calculate the minimum frame size.

Step 1: Calculate Propagation Delay (T_p)

$T_p = \frac{\text{Distance}}{\text{Speed}} = \frac{2500}{2 \times 10^8} = 12.5 \times 10^{-6}$ s $= 12.5$ μs

Step 2: Apply the Condition

$L_{min} = 2 \times T_p \times R$

Step 3: Calculate L_{min}

$L_{min} = 2 \times (12.5 \times 10^{-6}) \times (10 \times 10^6)$

$L_{min} = 25 \times 10^{-6} \times 10^7 = 250$ bits

The minimum frame size required to guarantee collision detection is 250 bits.

3.5 Routing Algorithms

Routing algorithms determine the optimal path for packets from source to destination.

Methodology:

Dijkstra Shortest Path Algorithm Used in Link-State Routing protocols like OSPF.

1. **Initialization:** Set the distance to the source node to 0 and all other nodes to ∞. Mark all nodes as unvisited.
2. **Node Selection:** Select the unvisited node with the smallest known distance (initially, this is the source node).
3. **Update Neighbors:** For the current node, examine all its unvisited neighbors. Calculate their tentative distances through the current node.
4. **Relaxation:** If the calculated tentative distance is less than the current assigned value, update the distance.
5. **Mark as Visited:** Once all neighbors are considered, mark the current node as visited. It will not be checked again.
6. **Repeat:** Repeat Steps 2-5 until all nodes are visited.

Worked Example:

Find Shortest Path using Dijkstra Consider a network with nodes A, B, C, D, and E with the following link costs: A-B: 2, A-C: 5, B-C: 2, B-D: 4, C-D: 1, C-E: 3, D-E: 1.
Find the shortest path tree starting from source node A.

Step 1: Initialization

Distances: A=0, B=∞, C=∞, D=∞, E=∞.
Unvisited: {A, B, C, D, E}

Step 2: Iteration 1 (Current Node: A)

Neighbors of A: B (cost 2), C (cost 5).
Update B: $\min(\infty, 0+2) = 2$.
Update C: $\min(\infty, 0+5) = 5$.
Visited: {A}. Unvisited: {B, C, D, E}. Distances: A=0, B=2, C=5, D=∞, E=∞.

Step 3: Iteration 2 (Current Node: B)

Smallest unvisited distance is B (2).
Neighbors of B (unvisited): C (cost 2), D (cost 4).
Update C: $\min(5, 2+2) = 4$. (Distance to C is updated via B)
Update D: $\min(\infty, 2+4) = 6$.
Visited: {A, B}. Unvisited: {C, D, E}. Distances: A=0, B=2, C=4, D=6, E=∞.

Step 4: Iteration 3 (Current Node: C)

Smallest unvisited distance is C (4).
Neighbors of C (unvisited): D (cost 1), E (cost 3).
Update D: $\min(6, 4+1) = 5$. (Distance to D is updated via C)
Update E: $\min(\infty, 4+3) = 7$.
Visited: {A, B, C}. Unvisited: {D, E}. Distances: A=0, B=2, C=4, D=5, E=7.

Step 5: Iteration 4 (Current Node: D)

Smallest unvisited distance is D (5).
Neighbors of D (unvisited): E (cost 1).
Update E: $\min(7, 5+1) = 6$. (Distance to E is updated via D)
Visited: {A, B, C, D}. Unvisited: {E}. Distances: A=0, B=2, C=4, D=5, E=6.

Step 6: Iteration 5 (Current Node: E)

Smallest unvisited distance is E (6). No unvisited neighbors.
Visited: {A, B, C, D, E}.

Final Shortest Paths from A:

- to B: Cost 2 (Path: A $\rightarrow$ B)

- to C: Cost 4 (Path: A $\rightarrow$ B $\rightarrow$ C)
- to D: Cost 5 (Path: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D)
- to E: Cost 6 (Path: A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D $\rightarrow$ E)

CHAPTER 4
Software Layer

This chapter focuses on the computational aspects of the Transport and Application layers. It covers numerical problems related to TCP/UDP mechanisms, Application Layer protocols (DNS, HTTP, FTP, Email), and Voice over IP (VoIP) transmission metrics.

4.1 Transport Layer Problems

4.1.1 UDP and TCP Checksum Calculation

Methodology:

UDP and TCP Checksum Calculation The checksum is used for error detection in transport layer segments.

1. Divide the data (including the pseudo-header, header, and payload) into 16-bit words. If the total number of bytes is odd, pad with a zero byte.
2. Add all 16-bit words together using ones' complement arithmetic.
3. If a carry bit overflows from the most significant bit (the 16th bit), add it back to the least significant bit of the sum.
4. Take the ones' complement (invert all bits: change 0 to 1 and 1 to 0) of the final sum. This result is the Checksum.

Worked Example:

Calculate Checksum for 16-bit Words Calculate the 16-bit checksum for the following three 16-bit words: `0110011001100000`, `0101010101010101`, and `1000111100001100`.

Step 1: Add the first two words.

$$\begin{array}{r} 0110\ 0110\ 0110\ 0000 \\ +\ 0101\ 0101\ 0101\ 0101 \\ \hline 1011\ 1011\ 1011\ 0101 \end{array}$$

Step 2: Add the third word to the result of Step 1.

$$\begin{array}{r} 1011\ 1011\ 1011\ 0101 \\ +\ 1000\ 1111\ 0000\ 1100 \\ \hline 1\ 0100\ 1010\ 1100\ 0001 \end{array}$$

Step 3: Handle the carry bit. A carry of 1 is generated from the most significant bit. Add it to the least significant bit:

$$\begin{array}{r} 0100\ 1010\ 1100\ 0001 \\ +\ 1 \\ \hline 0100\ 1010\ 1100\ 0010 \end{array}$$

Step 4: Take the ones' complement. Invert all bits of the sum to generate the checksum: Sum: `0100 1010 1100 0010` Checksum: `1011 0101 0011 1101`

4.1.2 TCP Round Trip Time Estimation

Methodology:

TCP Round Trip Time Estimation TCP uses an exponential weighted moving average to estimate the RTT for its retransmission timer.

1. **Estimated RTT:**

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT}_{\text{prev}} + \alpha \times \text{SampleRTT}$$

   where typically $\alpha = 0.125$.

2. **DevRTT (Deviation in RTT):**

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT}_{\text{prev}} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}_{\text{prev}}|$$

   where typically $\beta = 0.25$.

3. **Timeout Interval:**

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

Worked Example:

Calculate TCP Timeout Interval A TCP connection has a current EstimatedRTT of 30 ms and a DevRTT of 5 ms. A new SampleRTT of 40 ms is received. Calculate the new EstimatedRTT, DevRTT, and the TCP Timeout Interval. Use $\alpha = 0.125$ and $\beta = 0.25$.

Step 1: Calculate new EstimatedRTT.

$$\text{EstimatedRTT} = (1 - 0.125) \times 30 + 0.125 \times 40$$

$$\text{EstimatedRTT} = 0.875 \times 30 + 0.125 \times 40 = 26.25 + 5 = 31.25 \text{ ms}$$

Step 2: Calculate new DevRTT. Note: The DevRTT formula uses the EstimatedRTT before the update.

$$\text{DevRTT} = (1 - 0.25) \times 5 + 0.25 \times |40 - 30|$$

$$\text{DevRTT} = 0.75 \times 5 + 0.25 \times 10 = 3.75 + 2.5 = 6.25 \text{ ms}$$

Step 3: Calculate Timeout Interval.

$$\text{TimeoutInterval} = 31.25 + 4 \times 6.25 = 31.25 + 25 = 56.25 \text{ ms}$$

4.1.3 TCP Congestion Window Size Calculation

Methodology:

TCP Congestion Window Size Calculation TCP uses an additive-increase multiplicative-decrease (AIMD) algorithm.

1. **Slow Start:** Starts with a Congestion Window (*cwnd*) of 1 Maximum Segment Size (MSS). *cwnd* doubles every RTT until it reaches the slow start threshold (*ssthresh*).
2. **Congestion Avoidance:** Once $cwnd \geq ssthresh$, *cwnd* increases by 1 MSS every RTT.
3. **Timeout (Loss):** If a timeout occurs, *ssthresh* becomes $cwnd/2$ and *cwnd* drops to 1 MSS. The phase returns to Slow Start.

Worked Example:

Calculate TCP Congestion Window Size A TCP connection has a current $cwnd = 16$ KB and $ssthresh = 32$ KB. The Maximum Segment Size (MSS) is 2 KB. The connection experiences a sequence of events. Determine the $cwnd$ at each stage.

1. 1 RTT passes with all ACKs received.
2. Another RTT passes with all ACKs received.
3. A timeout occurs.
4. 1 RTT passes with all ACKs received.

Step 1: Initial state. $cwnd = 16$ KB (which is 8 MSS). Since $cwnd < ssthresh$ ($16 < 32$), TCP is in Slow Start.
Step 2: After 1 RTT. In Slow Start, $cwnd$ doubles. $cwnd = 16 \text{ KB} \times 2 = 32$ KB. Now $cwnd = ssthresh$. The phase shifts to Congestion Avoidance.
Step 3: After another 1 RTT. In Congestion Avoidance, $cwnd$ increases by 1 MSS (2 KB). $cwnd = 32 \text{ KB} + 2 \text{ KB} = 34$ KB.
Step 4: A timeout occurs. $ssthresh$ becomes half of the current $cwnd$. $ssthresh = 34 \text{ KB}/2 = 17$ KB. $cwnd$ drops to 1 MSS. $cwnd = 2$ KB. The phase returns to Slow Start.
Step 5: After 1 RTT. In Slow Start, $cwnd$ doubles. $cwnd = 2 \text{ KB} \times 2 = 4$ KB.

4.2 Application Layer Problems

4.2.1 DNS Resolution Delay Calculation

Methodology:

DNS Resolution Delay Calculation To calculate the total time taken to resolve a domain name and establish a connection:

1. Identify the Round Trip Time (RTT) to the Local DNS server, Root DNS server, TLD DNS server, and Authoritative DNS server.
2. For iterative queries, the local DNS server makes individual requests to the Root, TLD, and Authoritative servers. Add the RTT of each required lookup.
3. If the IP address is cached, the delay is only the RTT to the local DNS server.
4. Add the RTT between the client and the target server to establish a TCP connection (if requested).

Worked Example:

Calculate Iterative DNS Delay A client wants to connect to a web server. The RTT between the client and its local DNS server is $RTT_1 = 5$ ms. The RTT between the local DNS server and the Root DNS server is $RTT_2 = 20$ ms, to the TLD server is $RTT_3 = 15$ ms, and to the Authoritative server is $RTT_4 = 10$ ms. Assume no DNS records are cached. Calculate the total time elapsed before the client receives the IP address of the web server.
Step 1: Client to Local DNS Server. The client sends a query to the local DNS server. Time taken is $RTT_1 = 5$ ms.
Step 2: Local DNS Iterative Queries. The local DNS server sequentially queries:

- Root Server: $RTT_2 = 20$ ms
- TLD Server: $RTT_3 = 15$ ms
- Authoritative Server: $RTT_4 = 10$ ms

Total time spent by local DNS $= 20 + 15 + 10 = 45$ ms.
Step 3: Total Resolution Time. Total Time = Client $\leftrightarrow$ Local DNS + Local DNS iterative lookups Total Time $= RTT_1 + (RTT_2 + RTT_3 + RTT_4) = 5$ ms $+ 45$ ms $= 50$ ms. The client receives the IP address after 50 ms.

4.2.2 Web Object Transmission Time Calculation

Methodology:

HTTP Object Fetching Time Calculation Total time to fetch web objects depends on the type of HTTP connection. Let RTT be the round trip time between client and server, and T_{tx} be the transmission time of the object.

1. **Non-Persistent HTTP (HTTP 1.0):** Requires a new TCP connection for every object.

$$\text{Time per object} = 2 \times RTT + T_{tx}$$

   For N objects, total time $= N \times (2 \times RTT + T_{tx})$.

2. **Persistent HTTP without Pipelining (HTTP 1.1):** Reuses TCP connection. Next request is sent only after receiving the previous object.

$$\text{Total Time} = (\text{TCP Setup}) + \text{Request for Base HTML} + \sum \text{Requests for objects}$$

$$\text{Total Time} = (1 \times RTT) + (1 \times RTT + T_{tx,\text{base}}) + N \times (1 \times RTT + T_{tx,\text{object}})$$

3. **Persistent HTTP with Pipelining:** Reuses TCP connection. Requests are sent back-to-back.

$$\text{Total Time} = 2 \times RTT + T_{tx,\text{base}} + 1 \times RTT + \sum T_{tx,\text{object}}$$

Worked Example:

Calculate HTTP Transfer Time A client requests a web page containing a base HTML file and 3 small images. The RTT between the client and server is 50 ms. The transmission time T_{tx} for all objects is negligible (assume $T_{tx} = 0$). Calculate the total time to fetch the page and all images using: 1) Non-Persistent HTTP 2) Persistent HTTP without pipelining
Step 1: Determine total objects. Total objects = 1 (base HTML) + 3 (images) = 4 objects.
Step 2: Non-Persistent HTTP Calculation. Each object requires $2 \times RTT$ (1 for TCP setup, 1 for HTTP request/response). Total time = 4 objects $\times (2 \times RTT)$ Total time $= 4 \times (2 \times 50$ ms$) = 4 \times 100$ ms $= 400$ ms.
Step 3: Persistent HTTP without Pipelining Calculation. TCP connection setup $= 1 \times RTT = 50$ ms. Base HTML fetch $= 1 \times RTT = 50$ ms. Fetching 3 images sequentially $= 3 \times RTT = 150$ ms. Total time $= 50 + 50 + 150 = 250$ ms.

4.2.3 Web Caching and Proxy Server Calculation

Methodology:

Web Caching Delay Calculation A proxy server (web cache) stores copies of recently requested web objects to reduce response time.

1. **Hit Rate (h):** The fraction of requests satisfied by the proxy cache.
2. **Miss Rate ($1 - h$):** The fraction of requests that must be fetched from the origin server.
3. **Average Response Time:**

$$\text{Avg Time} = h \times (\text{Delay to Cache}) + (1 - h) \times (\text{Delay to Origin Server})$$

Worked Example:

Calculate Average Web Response Time A local college has a proxy server. The delay to fetch a blog post from the local proxy cache is 10 ms. If the post is not in the cache, it must be fetched from the origin server, which takes 800 ms. If the cache hit rate is 0.6 (or 60%), calculate the average response time for a web request.
Step 1: Identify values. Delay to cache = 10 ms. Delay to origin = 800 ms. Hit rate $h = 0.6$. Miss rate $(1 - h) = 1 - 0.6 = 0.4$.
Step 2: Calculate Average Response Time. Avg Time $= (0.6 \times 10) + (0.4 \times 800)$ Avg Time $= 6 + 320 =$ 326 ms.
Thus, using the proxy cache reduces the average delay from 800 ms to 326 ms.

4.2.4 File Transfer Time Calculation

Methodology:

File Transfer Time Calculation To calculate the time to transfer a file using FTP or an email attachment via SMTP:

1. Identify the file size F in bits (1 Byte = 8 bits).
2. Identify the link bandwidth R in bits per second (bps).
3. **Transmission Delay (T_{tx}):** $T_{tx} = \frac{F}{R}$
4. **Total Transfer Time:** Includes connection setup, propagation delay (T_{prop}), and transmission delay.

$$\text{Total Time} = \text{Setup Time} + T_{tx} + T_{prop}$$

(If setup time and propagation delay are negligible, Total Time $\approx T_{tx}$).

Worked Example:

Calculate FTP File Transfer Time A user uploads a 15 MB file to an FTP server over a 10 Mbps link. The FTP control connection setup takes 100 ms, and data connection setup takes 100 ms. The propagation delay is negligible. Calculate the total time to complete the file transfer.
Step 1: Convert file size to bits. $F = 15$ MB $= 15 \times 10^6$ Bytes $\times$ 8 bits/Byte $= 120 \times 10^6$ bits.
Step 2: Convert bandwidth to bps. $R = 10$ Mbps $= 10 \times 10^6$ bps.
Step 3: Calculate Transmission Delay. $T_{tx} = \frac{F}{R} = \frac{120 \times 10^6 \text{ bits}}{10 \times 10^6 \text{ bps}} = 12$ seconds.
Step 4: Calculate Total Time. Total Time = Control Setup + Data Setup + T_{tx} Total Time = 0.1 s + 0.1 s + 12 s = 12.2 seconds.

4.3 Voice over IP Calculations

4.3.1 VoIP Packetization Delay and Bandwidth

Methodology:

VoIP Packetization and Bandwidth Calculation Voice over IP digitizes audio and sends it as packets.

1. **Data Generation Rate (R_c):** The rate at which the codec generates voice data (e.g., PCM generates 64 kbps).
2. **Packetization Delay (T_{pack}):** Time taken to fill one packet payload.

$$T_{pack} = \frac{\text{Payload Size (bits)}}{R_c}$$

3. **Total Packet Size:**

$$\text{Total Size} = \text{Payload Size} + \text{Headers (RTP + UDP + IP + Ethernet)}$$

4. **Required Network Bandwidth:**

$$\text{Bandwidth} = \frac{\text{Total Packet Size (bits)}}{T_{pack}}$$

Worked Example:

Calculate VoIP Packetization Delay and Bandwidth A VoIP application uses a PCM codec that generates data at 64 kbps. The application gathers 20 ms of audio data into a single chunk. The headers added are RTP (12 Bytes), UDP (8 Bytes), and IP (20 Bytes). 1) Calculate the size of the audio payload. 2) Calculate the total packet size. 3) Calculate the required network bandwidth for this stream.

Step 1: Calculate Payload Size. $R_c = 64$ kbps $= 64,000$ bps. $T_{pack} = 20$ ms $= 0.020$ s. Payload (bits) $= R_c \times T_{pack} = 64,000 \times 0.020 = 1,280$ bits. Payload (Bytes) $= 1,280/8 = 160$ Bytes.

Step 2: Calculate Total Packet Size. Total Header Size $= 12$ (RTP) $+ 8$ (UDP) $+ 20$ (IP) $= 40$ Bytes. Total Packet Size $=$ Payload $+$ Headers $= 160$ Bytes $+ 40$ Bytes $= 200$ Bytes. Total Packet Size in bits $= 200 \times 8 = 1,600$ bits.

Step 3: Calculate Required Network Bandwidth. The sender generates one $1,600$-bit packet every 20 ms. Bandwidth $= \frac{1,600 \text{ bits}}{0.020 \text{ s}} = 80,000$ bps $= 80$ kbps.

CHAPTER 5
Network Security

Network security relies heavily on cryptographic algorithms to ensure data confidentiality, integrity, and authentication. This chapter focuses on the computational methods used in both symmetric and asymmetric cryptography, providing practical techniques to encrypt and decrypt data.

5.1 Substitution Ciphers

Substitution ciphers replace each letter in the plaintext with another letter or symbol according to a fixed system.

5.1.1 Caesar Cipher

The Caesar cipher is one of the simplest substitution ciphers where each letter in the plaintext is shifted by a fixed number of positions down the alphabet.

Methodology:

Caesar Cipher Algorithm **1. Preparation:** Assign a numerical value to each letter of the alphabet: $A = 0, B = 1, \ldots, Z = 25$. Let k be the shift key (an integer).
2. Encryption Formula: For each plaintext letter P, the corresponding ciphertext letter C is computed as:

$$C = (P + k) \pmod{26}$$

3. Decryption Formula: To recover the plaintext letter P from the ciphertext letter C:

$$P = (C - k) \pmod{26}$$

If $(C - k)$ is negative, add 26 to the result before finding the corresponding letter.

Worked Example:

Encrypting and Decrypting with Caesar Cipher **Problem:** Encrypt the plaintext `HELLO` using a Caesar cipher with a shift key of $k = 3$. Then demonstrate the decryption process.
Solution: First, assign numerical values to each letter of the plaintext:

Plaintext	H	E	L	L	O
P Value	7	4	11	11	14

Apply the encryption formula $C = (P + 3) \pmod{26}$:

- H: $C = (7 + 3) \pmod{26} = 10 \implies$ K
- E: $C = (4 + 3) \pmod{26} = 7 \implies$ H
- L: $C = (11 + 3) \pmod{26} = 14 \implies$ O
- L: $C = (11 + 3) \pmod{26} = 14 \implies$ O
- O: $C = (14 + 3) \pmod{26} = 17 \implies$ R

The encrypted ciphertext is **KHOOR**.
Decryption: Convert the ciphertext back to numerical values (C):

Ciphertext	K	H	O	O	R
C Value	10	7	14	14	17

Apply the decryption formula $P = (C - 3) \pmod{26}$:

- K: $P = (10 - 3) \pmod{26} = 7 \implies$ H
- H: $P = (7 - 3) \pmod{26} = 4 \implies$ E
- O: $P = (14 - 3) \pmod{26} = 11 \implies$ L
- O: $P = (14 - 3) \pmod{26} = 11 \implies$ L
- R: $P = (17 - 3) \pmod{26} = 14 \implies$ O

The decrypted text is **HELLO**, which matches the original plaintext.

5.1.2 Vigenère Cipher

The Vigenère cipher uses a keyword to determine multiple shift values, making it a polyalphabetic substitution cipher.

Methodology:

Vigenere Cipher Algorithm **1. Preparation:** Assign numerical values 0 to 25 to letters A to Z. Obtain the plaintext and the keyword. Repeat the keyword until it matches the length of the plaintext.
2. Encryption Formula: Let P_i be the i-th letter of the plaintext and K_i be the i-th letter of the repeated keyword. The ciphertext letter C_i is:

$$C_i = (P_i + K_i) \pmod{26}$$

3. Decryption Formula: The plaintext letter P_i is recovered by:

$$P_i = (C_i - K_i) \pmod{26}$$

If the result is negative, add 26.

Worked Example:

Encryption using Vigenere Cipher **Problem:** Encrypt the plaintext `ATTACK` using the keyword `LEMON`.
Solution: Step 1: Repeat the keyword. Length of plaintext is 6. Repeat `LEMON` to match the length: `LEMONL`.
Step 2: Map to numerical values.

Plaintext	A	T	T	A	C	K
P_i Value	0	19	19	0	2	10
Keyword	L	E	M	O	N	L
K_i Value	11	4	12	14	13	11

Step 3: Apply the encryption formula $C_i = (P_i + K_i) \pmod{26}$**.**

- $C_1 = (0 + 11) \pmod{26} = 11 \implies$ L
- $C_2 = (19 + 4) \pmod{26} = 23 \implies$ X
- $C_3 = (19 + 12) \pmod{26} = 31 \pmod{26} = 5 \implies$ F
- $C_4 = (0 + 14) \pmod{26} = 14 \implies$ O

- $C_5 = (2 + 13) \pmod{26} = 15 \implies \text{P}$
- $C_6 = (10 + 11) \pmod{26} = 21 \implies \text{V}$

The encrypted ciphertext is **LXFOPV**.

5.2 Transposition Ciphers

Transposition ciphers do not change the characters in the plaintext; instead, they rearrange their order based on a defined algorithm.

5.2.1 Rail Fence Cipher

The Rail Fence cipher writes the plaintext in a zigzag pattern across multiple rows (rails) and reads off each row sequentially to form the ciphertext.

Methodology:

Rail Fence Cipher Algorithm **1. Encryption Process:**

- Choose the number of rails (rows), say k.
- Write the plaintext characters downwards diagonally until the bottom rail is reached, then upwards diagonally until the top rail is reached, forming a zigzag pattern.
- Read the characters row by row from top to bottom to form the ciphertext.

2. Decryption Process:

- Construct an empty grid with k rails and length equal to the ciphertext.
- Mark the zigzag path with placeholders (e.g., asterisks).
- Fill the placeholders row by row with the letters from the ciphertext.
- Read the plaintext following the zigzag path.

Worked Example:

Encrypting using Rail Fence Cipher **Problem:** Encrypt the message `NETWORKSECURITY` using a Rail Fence cipher with 3 rails.
Solution: Step 1: Write the text in a zigzag pattern across 3 rails.

N	.	.	.	O	.	.	.	E	.	.	.	I	.	.
.	E	.	W	.	R	.	S	.	C	.	R	.	T	.
.	.	T	.	.	.	K	.	.	.	U	.	.	.	Y

Step 2: Read row by row.

- Rail 1: N O E I
- Rail 2: E W R S C R T
- Rail 3: T K U Y

The final ciphertext is **NOEIEWRSCRTTKUY**.

5.2.2 Columnar Transposition Cipher

The Columnar Transposition cipher writes the plaintext in a grid and reads the columns out of order, determined by a keyword.

Methodology:

Columnar Transposition Algorithm **1. Encryption Process:**

- Select a keyword. The alphabetical order of the letters in the keyword determines the column reading order.
- Write the plaintext row by row into a grid whose number of columns matches the length of the keyword.
- If the grid is incomplete, pad the last row with filler characters (e.g., X).
- Read the columns vertically according to the alphabetical order of the keyword's letters to form the ciphertext.

2. Decryption Process:

- Determine the number of columns (length of the keyword) and the number of rows (length of ciphertext divided by columns).
- Write the ciphertext vertically into columns following the alphabetical order of the keyword.
- Read the grid row by row to recover the plaintext.

Worked Example:

Encryption with Columnar Transposition **Problem:** Encrypt the message `DEFENDTHEEASTWALLOFTHECASTLE` using the keyword `GERMAN`.
Solution: Step 1: Determine the column order based on the keyword. The keyword is `GERMAN` (6 letters). Alphabetical order: A(1), E(2), G(3), M(4), N(5), R(6).

G	E	R	M	A	N
3	2	6	4	1	5

Step 2: Write the plaintext into the grid. Length of plaintext is 28. Since 28 is not a multiple of 6, we pad it with two 'X' characters to make it 30.

G (3)	E (2)	R (6)	M (4)	A (1)	N (5)
D	E	F	E	N	D
T	H	E	E	A	S
T	W	A	L	L	O
F	T	H	E	C	A
S	T	L	E	X	X

Step 3: Read columns in numerical order (1 to 6).

- Column 1 (A): NALCX
- Column 2 (E): EHWTT
- Column 3 (G): DTTFS
- Column 4 (M): EELEE
- Column 5 (N): DSOAX
- Column 6 (R): FEAHL

The final ciphertext is **NALCXEHWTTDTTFSEELEEDSOAXFEAHL**.

5.3 Public Key Cryptography

Public key cryptography (asymmetric cryptography) uses a pair of keys: a public key for encryption and a private key for decryption.

5.3.1 RSA Algorithm

The Rivest-Shamir-Adleman (RSA) algorithm relies on the mathematical difficulty of factoring large prime numbers.

Methodology:

RSA Algorithm **1. Key Generation:**

- Choose two distinct prime numbers, p and q.
- Compute $n = p \times q$. The value n is part of the public key and determines the modulus.
- Compute Euler's totient function: $\phi(n) = (p-1) \times (q-1)$.
- Choose an integer e such that $1 < e < \phi(n)$ and e is coprime to $\phi(n)$ (i.e., $\gcd(e, \phi(n)) = 1$). e is the public exponent.
- Compute the private exponent d such that $d \times e \equiv 1 \pmod{\phi(n)}$. This is calculated using the Extended Euclidean Algorithm.

Public Key: (e, n)
Private Key: (d, n)
2. Encryption: Given a plaintext message M (where $M < n$), compute the ciphertext C:

$$C = M^e \pmod{n}$$

3. Decryption: Given the ciphertext C, compute the original message M:

$$M = C^d \pmod{n}$$

Worked Example:

Key Generation and Encryption using RSA **Problem:** Perform RSA encryption and decryption given primes $p = 11$, $q = 3$, public exponent $e = 3$, and plaintext message $M = 4$. Find the private key d, the ciphertext C, and verify decryption.
Solution: Step 1: Compute n and $\phi(n)$.

$$n = p \times q = 11 \times 3 = 33$$

$$\phi(n) = (p-1) \times (q-1) = 10 \times 2 = 20$$

Step 2: Verify e and compute d. Given $e = 3$. We verify $\gcd(3, 20) = 1$, which is correct. We need to find d such that $d \times 3 \equiv 1 \pmod{20}$.

- Trying values for d:
- $d = 7 \implies 7 \times 3 = 21 \equiv 1 \pmod{20}$.
- Thus, $d = 7$.

Public Key: $(e, n) = (3, 33)$
Private Key: $(d, n) = (7, 33)$
Step 3: Encrypt the message $M = 4$.

$$C = M^e \pmod{n} = 4^3 \pmod{33}$$

$$C = 64 \pmod{33} = 31$$

The ciphertext is **31**.
Step 4: Decrypt the ciphertext $C = 31$.

$$M = C^d \pmod{n} = 31^7 \pmod{33}$$

Instead of computing 31^7 directly, use modular exponentiation properties:

$$31 \equiv -2 \pmod{33}$$

$$M = (-2)^7 \pmod{33} = -128 \pmod{33}$$

Divide -128 by 33: $-128 = 33 \times (-4) + 4$. Thus, $-128 \equiv 4 \pmod{33}$. The decrypted message is **4**, which matches the original plaintext.

5.3.2 Diffie-Hellman Key Exchange

The Diffie-Hellman (DH) algorithm allows two parties to securely establish a shared secret key over an insecure channel, based on the difficulty of computing discrete logarithms.

Methodology:

Diffie Hellman Key Exchange **1. Public Parameters:** Alice and Bob publicly agree on a large prime modulus p and a generator base g, where g is a primitive root modulo p.
2. Key Exchange Process:

- **Alice:** Chooses a secret integer a. Computes her public key $A = g^a \pmod{p}$. Sends A to Bob.
- **Bob:** Chooses a secret integer b. Computes his public key $B = g^b \pmod{p}$. Sends B to Alice.

3. Shared Secret Computation:

- **Alice:** Receives B and computes the shared secret $S = B^a \pmod{p}$.
- **Bob:** Receives A and computes the shared secret $S = A^b \pmod{p}$.

Both calculations result in the same value: $S = g^{ab} \pmod{p}$.

Worked Example:

Computing Shared Secret Key **Problem:** Alice and Bob wish to establish a shared secret using Diffie-Hellman. They agree on a prime $p = 23$ and a base $g = 5$. Alice chooses her secret key $a = 6$, and Bob chooses his secret key $b = 15$. Compute their public keys and the final shared secret key.
Solution: Step 1: Compute Public Keys. Alice's public key A:

$$A = g^a \pmod{p} = 5^6 \pmod{23}$$

$$5^2 = 25 \equiv 2 \pmod{23}$$

$$5^6 = (5^2)^3 \equiv 2^3 \equiv 8 \pmod{23}$$

Alice sends $A = 8$ to Bob.
Bob's public key B:

$$B = g^b \pmod{p} = 5^{15} \pmod{23}$$

Using the previous result $5^6 \equiv 8 \pmod{23}$:

$$5^{12} = (5^6)^2 \equiv 8^2 = 64 \equiv 18 \equiv -5 \pmod{23}$$

$$5^{15} = 5^{12} \times 5^3 \equiv -5 \times (2 \times 5) \equiv -5 \times 10 = -50 \pmod{23}$$

$-50 = 23 \times (-3) + 19$. Thus, $B = 19$. Bob sends $B = 19$ to Alice.

Step 2: Compute Shared Secret S. Alice computes:

$$S = B^a \pmod{p} = 19^6 \pmod{23}$$

Since $19 \equiv -4 \pmod{23}$:

$$S \equiv (-4)^6 = 4^6 = (4^3)^2 = 64^2 \pmod{23}$$

$64 = 23 \times 2 + 18 \equiv -5 \pmod{23}$.

$$S \equiv (-5)^2 = 25 \equiv 2 \pmod{23}$$

Bob computes:

$$S = A^b \pmod{p} = 8^{15} \pmod{23}$$

$8 = 2^3$, so $8^{15} = (2^3)^{15} = 2^{45}$. By Fermat's Little Theorem, $a^{p-1} \equiv 1 \pmod{p}$, so $2^{22} \equiv 1 \pmod{23}$.

$$2^{45} = 2^{22 \times 2 + 1} = (2^{22})^2 \times 2^1 \equiv 1^2 \times 2 \equiv 2 \pmod{23}$$

Both Alice and Bob successfully compute the identical shared secret key **2**.

Formulae

Appendix: Key Formulae

This appendix provides a quick reference to the key formulae and mathematical relationships discussed in this book.

Unit 1: Physical Layer and Network Basics

Network Topologies

Number of Links in a Fully Connected Mesh Topology:

$$L = \frac{n(n-1)}{2}$$

Where n is the number of nodes (or ports).

Delay and Performance

Total Delay:

$$\text{Total Delay} = \text{Propagation Delay} + \text{Transmission Delay}$$

Transmission Delay:

$$\text{Transmission Delay} = \frac{\text{Message Size}}{\text{Bandwidth}}$$

Propagation Delay:

$$\text{Propagation Delay} = \frac{\text{Distance}}{\text{Propagation Speed}}$$

Signal and Channel Capacity

Signal-to-Noise Ratio (SNR) in Decibels:

$$\text{SNR}_{\text{dB}} = 10\log_{10}(\text{SNR}) \implies \text{SNR} = 10^{\frac{\text{SNR}_{\text{dB}}}{10}}$$

Power Ratio in Decibels (Attenuation/Gain):

$$\text{dB} = 10\log_{10}\left(\frac{P_2}{P_1}\right)$$

Where P_1 and P_2 are the input and output powers, respectively.

Shannon Capacity (for Noisy Channels):

$$C = B \times \log_2(1 + \text{SNR})$$

Where C is the capacity in bps and B is the bandwidth in Hz.

Nyquist Bit Rate (for Noiseless Channels):

$$\text{BitRate} = 2 \times B \times \log_2(L)$$

Where L is the number of signal levels.

Unit 3: Data Link Layer

Transmission and Propagation

Transmission Time (T_t):

$$T_t = \frac{L}{R}$$

Where L is the frame length and R is the data rate.

Propagation to Transmission Time Ratio (a):

$$a = \frac{T_p}{T_t}$$

Where T_p is the propagation delay.

Protocol Efficiency

Efficiency of Sliding Window Protocol (η):

$$\eta = \frac{W}{1 + 2a}$$

Where W is the window size. For Stop-and-Wait, $W = 1$.

Unit 4: Transport and Application Layers

TCP Timer Management

Estimated Round Trip Time (RTT):

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT}_{\text{prev}} + \alpha \times \text{SampleRTT}$$

Typically, $\alpha = 0.125$.

RTT Deviation (DevRTT):

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT}_{\text{prev}} + \beta \times |\text{SampleRTT} - \text{EstimatedRTT}_{\text{prev}}|$$

Typically, $\beta = 0.25$.

TCP Timeout Interval:

$$\text{TimeoutInterval} = \text{EstimatedRTT} + 4 \times \text{DevRTT}$$

HTTP Performance

Total Time for Non-Persistent HTTP:

$$\text{Total Time} = (2 \times \text{RTT}) + T_{\text{tx, base}} + \sum (2 \times \text{RTT} + T_{\text{tx, object}})$$

Total Time for Persistent HTTP:

$$\text{Total Time} = (2 \times \text{RTT}) + T_{\text{tx, base}} + 1 \times \text{RTT} + \sum T_{\text{tx, object}}$$

Average Web Cache Response Time:

$$\text{Avg Time} = h \times (\text{Delay to Cache}) + (1 - h) \times (\text{Delay to Origin Server})$$

Where h is the cache hit rate.

VoIP and Multimedia

Packetization Delay (T_{pack}):

$$T_{\text{pack}} = \frac{\text{Payload Size}}{R_c}$$

Where R_c is the encoding rate.

Total Packet Size:

$$\text{Total Size} = \text{Payload Size} + \text{Headers (RTP + UDP + IP + Ethernet)}$$

Required Bandwidth:

$$\text{Bandwidth} = \frac{\text{Total Size}}{T_{\text{pack}}}$$

Unit 5: Cryptography and Network Security

Classical Ciphers

Caesar Cipher Encryption & Decryption:

$$C = (P + k) \pmod{26}$$
$$P = (C - k) \pmod{26}$$

Where P is plaintext, C is ciphertext, and k is the shift key.

Vigenère/Polyalphabetic Cipher:

$$C_i = (P_i + K_i) \pmod{26}$$
$$P_i = (C_i - K_i) \pmod{26}$$

RSA Algorithm

Modulus (n) and Euler's Totient ($\phi(n)$):

$$n = p \times q$$
$$\phi(n) = (p-1) \times (q-1)$$

Where p and q are prime numbers.

RSA Encryption & Decryption:

$$C = M^e \pmod{n}$$
$$M = C^d \pmod{n}$$

Where M is the message, e is the public exponent, and d is the private exponent.

Diffie-Hellman Key Exchange

Public Keys (A and B):

$$A = g^a \pmod{p}$$
$$B = g^b \pmod{p}$$

Where p is a prime, g is a primitive root modulo p, and a, b are private keys.

Shared Secret (S):

$$S = B^a \pmod{p} = A^b \pmod{p}$$