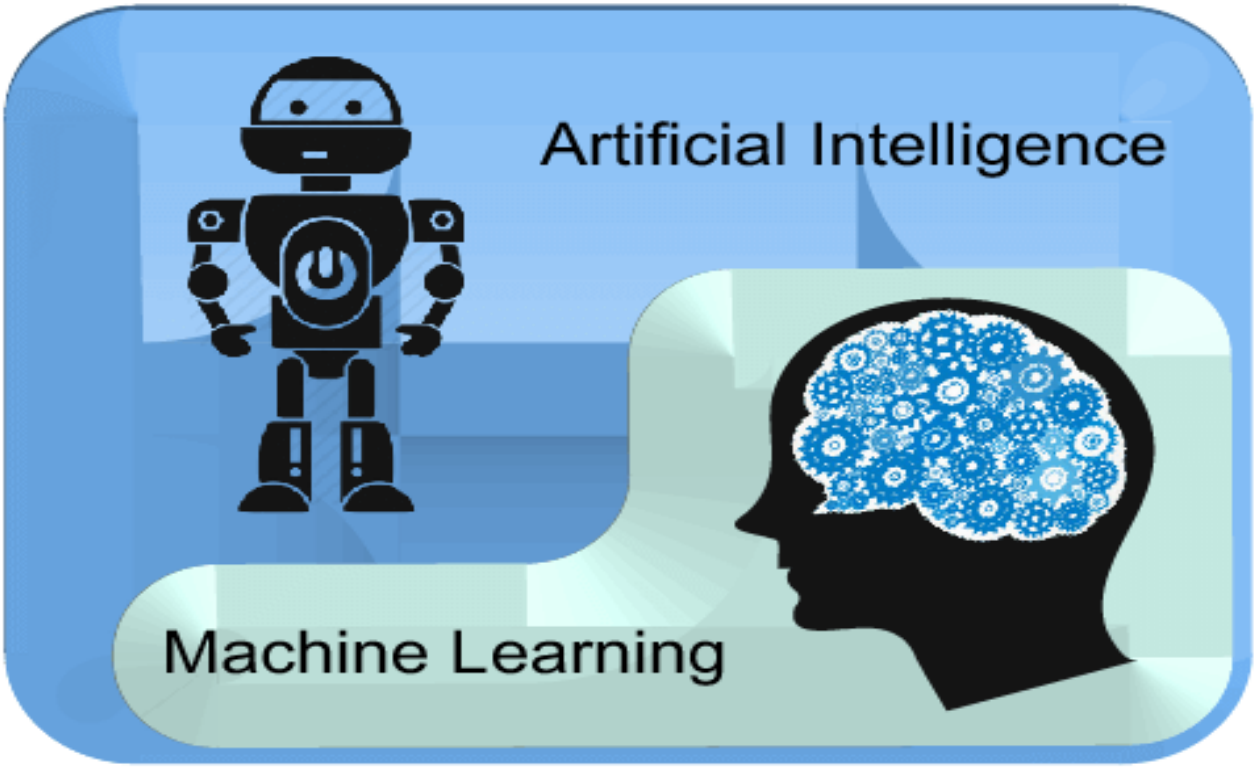
SCHOOL OF ENGINEERING

DAYANANDA SAGAR UNIVERSITY
SCHOOL OF ENGINEERING

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING
Artificial Intelligence & Machine Learning

Computer Networks Laboratory Manual
22AM3504



Dayananda Sagar University
Devarakaggalahalli, Harohalli Kanakapura Road,
Dt, Ramanagara, Karnataka 562112

SCHOOL OF ENGINEERING

DAYANANDA SAGAR UNIVERSITY
SCHOOL OF ENGINEERING

Devarakaggalahalli, Harohalli Kanakapura Road,
Dt, Ramanagara, Karnataka 562112

School of Engineering

Vision and Mission

Vision

To be a centre of excellence in education, research & training, innovation & entrepreneurship and to produce citizens with exceptional leadership qualities to serve national and global needs.

Mission

To achieve our objectives in an environment that enhances creativity, innovation and scholarly pursuits while adhering to our vision.

Values

The Pursuit of Excellence

A commitment to strive continuously to improve ourselves and our systems with the aim of becoming the best in our field.

Fairness

A commitment to objectivity and impartiality, to earn the trust and respect of society.

Leadership

A commitment to lead responsively and creatively in educational and research processes.

Integrity and Transparency

A commitment to be ethical, sincere and transparent in all activities and to treat all individuals with dignity and respect.

DAYANANDA SAGAR UNIVERSITY
SCHOOL OF ENGINEERING
Devarakaggalahalli, Harohalli Kanakapura Road,
Dt, Ramanagara, Karnataka 562112

Computer Science and Engineering
(Artificial Intelligence and Machine Learning)

Vision

- To produce graduates in Computer Science and Engineering (Artificial Intelligence & Machine Learning) through excellence in education and research with an emphasis on sustainable eco-system that contributes significantly to the society.

Mission

The Department Computer Science and Engineering (Artificial Intelligence & Machine Learning) is committed to:

- Impart quality education through the state-of-the-art curriculum, infrastructure facilities, cutting edge technologies, Sustainable learning practices and lifelong learning.
- Collaborate with industry-academia and inculcate interdisciplinary research to transform professionals into technically competent.
- Produce engineers and techno-entrepreneurs for global needs.

DAYANANDA SAGAR UNIVERSITY
SCHOOL OF ENGINEERING

Devarakaggalahalli, Harohalli Kanakapura Road,
Dt, Ramanagara, Karnataka 562112

Computer Science and Engineering

(Artificial Intelligence and Machine Learning)

Program Educational Objectives (PEO's)

- **PEO1:** Apply appropriate theory, practices, and tools of machine intelligence to the specification, design, implementation, maintenance, and evaluation in the workplace or in higher education.
- **PEO2:** Adapt, contribute and innovate new technologies in their computing profession by working in teams to design, implement, and maintain in the key domains of Artificial Intelligence & Machine Learning.
- **PEO3:** Function effectively in the work place as competent Artificial Intelligence & Machine Learning Professionals, Entrepreneurs or Researchers or maintain employment through lifelong learning such as professional conferences, certificate programs or other professional educational activities, ethics, and societal awareness.

Programme Outcome (PO's)

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSO's)

- PSO1. Cognitive Outcome: Solve complex engineering problems in computing by applying the principles in Artificial Intelligence, Machine Learning, Network Engineering, Software Engineering, Data Engineering and Intelligent Systems.
- PSO2. Skill & Design Outcome: Apply technical skills and research skills through professional societies, certification programs, projects, internships and laboratory exercises to design & develop algorithms, programs, and projects using modern software tools to provide the sustainable solutions to Computer Science, Artificial Intelligence and Machine Learning problems related to the society and environment.

COURSE OBJECTIVES:

1. **Outline** the basic principles of computer networking and how computer network hardware and software operate.
2. **Evaluate** the operation and performance of practical data link protocols using the principles of framing, error detection and correction.
3. **Apply** the principles of network layer design to the analysis and evaluation of routing algorithms, congestion control techniques, internetworking and addressing.
4. **Investigate** the basic transport layer facilities and essentials of transport. Protocol
5. **Illustrate** the working of various application layer protocols.

COURSE OUTCOMES:

CO's	Description	BTL
At the end of the course the student will be able to:		
1	**Explain** the basic concepts of data communications including the key aspects of networking and their interrelationship, packet switching, circuit switching and cell switching as internal and external operations, physical structures, types, models, and internetworking.	L2
2	**Apply** the concept of Hamming distance, the significance of the minimum Hamming distance and its relationship to errors as well as the detection and correction of errors in block codes.	L3
3	**Solve** the problems related to various Routing Algorithms and also perform the Interpretation of routers, Internet Protocol IPv4, and IPv6.	L3
4	**Recognize** transport layer services and infer UDP and TCP protocols and **Distinguish** between UDP and TCP Protocols.	L4
5	**Infer** the significance, and purpose of protocols (FTP, SMTP), standards, and use in data communications and networking and **analyze** the most common DNS resource records that occur in a zone file.	L4

Mapping Levels of COs to POs / PSOs														
				Program Outcomes (POs)									PSOs	
	1	2	3	4	5	6	7	8	9	10	11	12	1	2
COs	Engineering Knowledge	Problem Analysis	Design & Development	Conduct Investigations of Complex Problems	Modern Tool Usage	The Engineer and Society	Environment and Sustainability	Ethics	Individual & Team Work	Communication	Project management and Finance	Life-long Learning	Cognitive Outcome	Skill & Design Outcome
CO1	3	-	-	-	-	-	-	-	-	-	-	-	1	1
CO2	3	3	3		-	-	-	-	-	-	-	-	2	2
CO3	3	3	3	-	-	-	-	-	-	-	-	-	2	2
CO4	3	3	3	-	-	-	-	-	-	-	-	-	1	1
CO5	3	3	3	-	-	-	-	-	-	-	-	-	1	1

General Lab Guidelines:

1 Lab **Attendance & Punctuality**

- Students must report to the lab on time.
- Minimum of 80% attendance is mandatory to appear for the final lab exam.

2 Preparation **Before Lab**

- Read the experiment and its objective before coming to the lab.
- Bring your lab record book, observation book, and necessary stationery.

3 Lab **Conduct**

- Maintain silence and discipline inside the lab.
- Handle all equipment and network devices with care.
- Follow all safety rules while working with cables and networking hardware.

4 Execution **of Experiments**

- Ensure that the required software tools (e.g., packet tracer, JAVE, Visual Studio, C/C++ compiler, IDEs) are working.
- Execute the programs individually or in assigned pairs/groups.
- Save your work regularly and maintain backups.

5 Documentation

- Write clear aim, algorithm/logic, source code, sample input-output, and result

in the observation book.
- Get your work verified by the faculty during the lab session.
- Complete and submit the fair record on time for each experiment.

6 Lab **Exam & Viva**
- Practical exams will be based on the listed experiments.
- Be prepared for viva-voce on concepts related to each lab exercise.
- Maintain academic integrity—copying code from others is strictly prohibited.

7 Lab **Resources**
- Do not modify any hardware or software configuration unless instructed.
- Report any malfunction immediately to the lab instructor.

8 Cleanliness **& Exit**
- Shut down all systems properly after use.
- Leave your workspace neat and tidy.
- Log out from your accounts and switch off monitors.

DO'S:-
- Uniform and ID card are must.
- Strictly follow the procedures for conduction of experiments.
- Records have to be submitted every week for evaluation.
- Chairs and stools should be kept under the workbenches when not in use.
- After the lab session, switch off every supply, disconnect and disintegrate the
- experiments and return the components.
- Keep your belongings in designated area.
- Never use damaged instruments, wires or connectors. Hand these parts to the instructor/ teaching assistant.
- Sign the log book when you enter/leave the laboratory.
- Record program source code, screenshots of execution (if applicable), and results.
- Keep faculty signatures with date wherever required.

DONT'S:-
- Don't touch open wires unless you are sure that there is no voltage. Always disconnect the plug by pulling on the connector body not by the cable. Switch off the supply while you make changes to the experiment.
- Don't leave the experiment table unattended when the experimental setup supply is on.
- Students are not allowed to work in laboratory alone or without presence of the teaching staff/ instructor.
- No additional material should be carried by the students during regular labs.
- Avoid stepping on electrical wires or any other computer cables.

- Do not include outdated or irrelevant experiments that are not in the syllabus.
- Do not leave experiments incomplete; ensure all practicals are updated.
- Do not plagiarize source code—always provide the implemented version you tested.

List of Experiments

About Computer Networks, Java code – Visual Studio, Packet Tracer

Computer Networks

The **Computer Networks Lab** is designed to provide students with practical exposure to the key concepts, protocols, algorithms, and tools used in networking. The lab focuses on both **simulation-based** and **programming-based** experiments to reinforce theoretical knowledge through hands-on practice.

Key Focus Areas

- Understanding and analyzing **line coding techniques** (Unipolar, Bipolar, Manchester, Differential Manchester).
- Implementing **error detection** algorithms like **CRC (Cyclic Redundancy Check)**.
- Practicing **network routing algorithms** such as **Bellman-Ford, Distance Vector Routing, and Link State Routing**.
- Applying **IP addressing and subnetting** concepts to design efficient subnets.
- Simulating real-world networking scenarios using **Cisco Packet Tracer**.

JAVA Code- Visual Studio

- Java is used to implement basic networking algorithms, routing techniques, and error detection methods.
- **Visual Studio Code (VS Code)** is a popular, lightweight IDE used for writing, debugging, and running Java programs.
- **Java Sockets** help simulate basic client-server communication.
- Students learn to compile (javac) and run (java) programs, handle file input/output, and process network data.

Key Benefits:

- Enhances coding skills for network-based applications.
- Encourages clear understanding of protocol logic through code.
- Easy debugging and version control with extensions in VS Code.

Cisco Packet Tracer

- **Cisco Packet Tracer** is a powerful network simulation tool that allows students to build, configure, and troubleshoot virtual network topologies.
- It is used for practicals like **DHCP communication, DNS server simulation, web server requests**, and testing network configurations.
- Provides drag-and-drop interface for connecting routers, switches, PCs, and servers.
- Students gain confidence in setting up real-world scenarios such as

subnetting, IP addressing, routing tables, and **file transfer protocols**.

Key Benefits:

- Simulates real-time networking without the need for physical hardware.
- Visualizes packet flow, routing updates, and server responses.
- Supports step-by-step practice of complex concepts.

Experiment 1
Analyze the various line coding techniques used for data transmission of a digital signal over a transmission line

Objective:

1. To understand the importance of line coding in digital communication.
2. To implement and visualize how different line coding schemes represent binary data.
3. To compare **Unipolar NRZ**, **Polar NRZ**, **Manchester**, and **Differential Manchester** encoding techniques.
4. To analyze the encoded output for a given binary input sequence.

Explanation:

In digital data transmission, **line coding** converts digital data into a digital signal that can be transmitted over physical media. It determines how bits (0s and 1s) are represented using voltage levels or signal transitions.

The program demonstrates **four common line coding schemes:**

1. **Unipolar NRZ (Non-Return to Zero):**
    - Logic '1' is represented by a positive voltage (+1) and logic '0' by zero voltage (0).
    - Simple but has a DC component and poor synchronization.
2. **Polar NRZ:**
    - Logic '1' is represented by positive voltage (+1), and logic '0' by negative voltage (-1).
    - Better synchronization and no DC component compared to Unipolar.
3. **Manchester Encoding:**
    - Each bit period is split into two halves:
        - Logic '1': High to Low transition (+1 to -1).
        - Logic '0': Low to High transition (-1 to +1).
    - Provides clock synchronization and is widely used in Ethernet.
4. **Differential Manchester Encoding:**
    - A transition occurs at the middle of each bit period.
    - Logic '1': Transition at the beginning.

- Logic '0': No transition at the beginning.
- Self-clocking and resistant to polarity inversion.

Source Code:

```
import java.util.Scanner;

public class pro4 {
    //functionname
    static void unipolarNRZ(int[] data) {
        System.out.println("Unipolar NRZ Encoding:");
        for (int bit : data) {   //intitialisng the bit information to bit
            if (bit == 1) System.out.print("+1 ");
            else System.out.print("0");
        }
        System.out.println();
    }

    static void polarNRZ(int[] data) {
        System.out.println("Polar NRZ Encoding:");
        for (int bit : data) {
            if (bit == 1) System.out.print("+1");
            else System.out.print("-1");
        }
        System.out.println();
    }

    static void manchester(int[] data) {
        System.out.println("Manchester Encoding:");
        for (int bit : data) {
            if (bit == 1) System.out.print("+1, -1\t ");
            else System.out.print("-1, +1 ");
        }
        System.out.println();
    }

    static void differentialManchester(int[] data) {
        System.out.println("Differential Manchester Encoding:");
        int lastTransition = 1; // Start with high signal
        for (int bit : data) {
            if (bit == 1) {
                System.out.print("-1, +1 "); // Transition for 1
            } else {
                System.out.print(lastTransition == 1 ? "-1, +1 " : "+1, -1 "); //
```

```
Transition for 0
            }
            lastTransition = -lastTransition; // Flip last transition state
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of bits in the data: ");
        int n = sc.nextInt();
        int[] data = new int[n];
        System.out.println("Enter the binary data bits (0s and 1s): ");
        for (int i = 0; i < n; i++) {
            data[i] = sc.nextInt();
        }

        System.out.println("Choose a line coding technique:");
        System.out.println("1. Unipolar NRZ");
        System.out.println("2. Polar NRZ");
        System.out.println("3. Manchester");
        System.out.println("4. Differential Manchester");
        System.out.print("Enter your choice (1-4): ");
        int choice = sc.nextInt();

        switch (choice) {
            case 1:
                unipolarNRZ(data);
                break;
            case 2:
                polarNRZ(data);
                break;
            case 3:
                manchester(data);
                break;
            case 4:
                differentialManchester(data);
                break;
            default:
                System.out.println("Invalid choice. Please select between 1 and 4.");
        }

        sc.close();
    }
}
```

Results:

- The given binary sequence **"11001110"** was encoded using the selected line coding technique *(**Manchester** in this example).*
- The output clearly shows the transition pairs representing each bit.
- Students can compare outputs for all four techniques to observe the differences in signal representation.

Conclusion:

This experiment successfully demonstrates how binary data can be represented using various line coding techniques. Each method has its unique features, advantages, and application scenarios in digital communication systems.

Experiment 2

Design a program for error-detecting code using CRC-CCITT (16- bits)

Objective:

1. To understand the Cyclic Redundancy Check (CRC) method for error detection in digital communication.
2. To implement CRC encoding and decoding in Java.
3. To analyze how CRC helps detect errors during data transmission.
4. To verify both error-free and corrupted data scenarios using the generated CRC code.

Explanation:

- Cyclic Redundancy Check (CRC) is a popular error-detecting technique used in networks and storage devices.
- The sender side appends a CRC code (also called checksum) to the data based on polynomial division.
- The receiver re-divides the received data by the same polynomial:
  - If the remainder is zero, the data is error-free.
  - If the remainder is non-zero, the data has errors.

Program Flow:

1. The user inputs:
   - The data bits to send.
   - The divisor bits (the generator polynomial).
2. The divideDataWithDivisor() function performs bitwise polynomial division using XOR.
3. The remainder is calculated, and the CRC code is generated.
4. The user provides the received data:
   - If you input the correct codeword, it simulates error-free reception.
   - If you alter bits intentionally, it simulates error detection.
5. The receiveData() function checks the remainder:
   - If remainder = 0 → "Data received without any error."
   - If remainder ≠ 0 → "Corrupted data received..."

Source Code

```
import java.util.*;
public class CRC {
   // import required classes and packages
   // main() method start
   public static void main(String args[]) {
      try (// create scanner class object to take input from user
      Scanner scan = new Scanner(System.in)) {
         // declare n for the size of the data
         int size;
         // take the size of the data from the user
```

```
        System.out.println("Enter the size of the data array: ");
        size = scan.nextInt();
        // declaration of the data array
        int data[] = new int[size];
        // take bits of the data from the user
        System.out.println("Enter data bits in the array one by one: ");
        for(int i = 0 ; i < size ; i++) {
            System.out.println("Enter bit " + (size-i) + ":");
            data[i] = scan.nextInt();
        }
        // take the size of the divisor from the user
        System.out.println("Enter the size of the divisor array:");
        size = scan.nextInt();
        // declaration of the divisor array
        int divisor[] = new int[size];
        System.out.println("Enter divisor bits in the array one by one: ");
        for(int i = 0 ; i < size ; i++) {
            System.out.println("Enter bit " + (size-i) + ":");
            divisor[i] = scan.nextInt();
        }
  // Divide the input data by the input divisor and store the result in the rem array
        int rem[] = divideDataWithDivisor(data, divisor);
        // iterate rem using for loop to print each bit
        for(int i = 0; i < rem.length-1; i++) {
            System.out.print(rem[i]);
        }
        System.out.println("\nGenerated CRC code is: ");

        for(int i = 0; i < data.length; i++) {
            System.out.print(data[i]);
        }
        for(int i = 0; i < rem.length-1; i++) {
            System.out.print(rem[i]);
        }
        System.out.println();
        // we create a new array that contains the original data with its CRC code
        // the size of the sentData array with be equal to the sum of the data and the
rem arrays length
        int sentData[] = new int[data.length + rem.length - 1];
        System.out.println("Enter bits in the array which you want to send: ");
        for(int i = 0; i < sentData.length; i++) {
            System.out.println("Enter bit " +(sentData.length - 1)+ ":");
            sentData[i] = scan.nextInt();
        }
        receiveData(sentData, divisor);
    }
  }
  // create divideDataWithDivisor() method to get CRC
  static int[] divideDataWithDivisor(int oldData[], int divisor[]) {
    // declare rem[] array
```

```
    int rem[] = new int[divisor.length];
    int i;
    int data[] = new int[oldData.length + divisor.length];
// use system's arraycopy() method for copying data into rem and data arrays
    System.arraycopy(oldData, 0, data, 0, oldData.length);
    System.arraycopy(data, 0, rem, 0, divisor.length);
    // iterate the oldData and exor the bits of the remainder and the divisor
    for(i = 0; i < oldData.length; i++) {
        System.out.println((i+1) + ".) First data bit is : "+ rem[0]);
        System.out.print("Remainder : ");
        if(rem[0] == 1) {
            // We have to exor the remainder bits with divisor bits
            for(int j = 1; j < divisor.length; j++) {
                rem[j-1] = exorOperation(rem[j], divisor[j]);
                System.out.print(rem[j-1]);
            }
        }
        else {
            // We have to exor the remainder bits with 0
            for(int j = 1; j < divisor.length; j++) {
                rem[j-1] = exorOperation(rem[j], 0);
                System.out.print(rem[j-1]);
            }
        }
        // The last bit of the remainder will be taken from the data
        // This is the 'carry' taken from the dividend after every step
        // of division
        rem[divisor.length-1] = data[i+divisor.length];
        System.out.println(rem[divisor.length-1]);
    }
    return rem;
}
// create exorOperation() method to perform exor data
static int exorOperation(int x, int y) {
    // This simple function returns the exor of two bits
    if(x == y) {
        return 0;
    }
    return 1;
}
// method to print received data
static void receiveData(int data[], int divisor[]) {

    int rem[] = divideDataWithDivisor(data, divisor);
    // Division is done
    for(int i = 0; i < rem.length; i++) {
        if(rem[i] != 0) {
            // if the remainder is not equal to zero, data is currupted
            System.out.println("Currupted data received...");
            return;
```

```
            }
        }
        System.out.println("Data received without any error.");
    }
}
```

Results:

- CRC successfully generates a check code for the given data and key.
- The receiver side correctly detects whether the received codeword has any errors.
- This demonstrates the robustness of CRC for error detection in data transmission.

Conclusion:
The experiment verifies that CRC-CCITT (16-bit) is a reliable method for detecting errors during digital data transmission. Students gain hands-on understanding of polynomial division and how CRC enhances data integrity in real-world networks.

Experiment 3

Design a program to find the shortest path between vertices using Belman- ford algorithm

Objective:

- To understand the **Bellman-Ford Algorithm** for single-source shortest path computation.
- To implement the algorithm in **Java** to find shortest paths in a weighted graph, even with negative edge weights.
- To detect the presence of any **negative-weight cycles** in the given graph.
- To verify the correctness of the shortest path output for various input graphs.

Explanation:

The Bellman-Ford Algorithm is a single-source shortest path algorithm that computes the shortest distances from a source vertex to all other vertices in a weighted graph.
Unlike Dijkstra's algorithm, Bellman-Ford can handle graphs with negative edge weights and can detect negative-weight cycles.
Working Steps:

1. Input: The user enters the number of vertices and edges, then specifies each edge with its source, destination, and weight.
2. Initialization: The distance array dist[] is initialized — the source vertex is set to distance 0, all others to ∞ (INFINITY).
3. Relaxation: For V-1 times, all edges are relaxed — this means checking if a shorter path can be found for each edge, and updating distances if so.
4. Negative-Weight Cycle Detection: One more pass is done over the edges. If any edge can still be relaxed, a negative-weight cycle exists.
5. Output: Displays shortest distances from the source vertex to every other vertex or reports the presence of a negative cycle.

Applications:

- Used in network routing protocols like RIP (Routing Information Protocol).
- Applied in transportation, logistics, and path planning in robotics and GPS systems.

Source Code

```
import java.util.Scanner;

public class BellmanFord {
   public static void bellmanFord(int[][] graph, int V, int E, int src) {
      int[] dist = new int[V];
```

```
        // Step 1: Initialize distances
        for (int i = 0; i < V; i++) {
            dist[i] = Integer.MAX_VALUE;
        }
        dist[src] = 0;

        // Step 2: Relax all edges |V| - 1 times
        for (int i = 1; i < V; i++) {
            for (int j = 0; j < E; j++) {
                int u = graph[j][0];
                int v = graph[j][1];
                int weight = graph[j][2];
                if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                    dist[v] = dist[u] + weight;
                }
            }
        }

        // Step 3: Check for negative-weight cycles
        for (int j = 0; j < E; j++) {
            int u = graph[j][0];
            int v = graph[j][1];
            int weight = graph[j][2];
            if (dist[u] != Integer.MAX_VALUE && dist[u] + weight < dist[v]) {
                System.out.println("Graph contains negative weight cycle");
                return;
            }
        }

        printSolution(dist, V);
    }

    public static void printSolution(int[] dist, int V) {
        System.out.println("Vertex\t\tDistance from Source");
        for (int i = 0; i < V; i++) {
            System.out.println(i + "\t\t" + dist[i]);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of vertices: ");
        int V = sc.nextInt();
        System.out.print("Enter the number of edges: ");
        int E = sc.nextInt();
```

```
        int[][] graph = new int[E][3];
        System.out.println("Enter the edges with source, destination, and
weight:");
        for (int i = 0; i < E; i++) {
            System.out.print("Edge " + (i + 1) + ": ");
            graph[i][0] = sc.nextInt(); // Source
            graph[i][1] = sc.nextInt(); // Destination
            graph[i][2] = sc.nextInt(); // Weight
        }

        System.out.print("Enter the source vertex: ");
        int src = sc.nextInt();

        bellmanFord(graph, V, E, src);

        sc.close();
    }
}
```

Sample Output/Results:

Sample Input:

Enter the number of vertices: 4

Enter the number of edges: 4

Enter the edges with source, destination, and weight:

Edge 1: 0 1 -1

Edge 2: 0 2 4

Edge 3: 1 2 3

Edge 4: 1 3 2

Enter the source vertex: 0

Expected Output:

Vertex Distance from Source

0 0

1 -1

2 2

3 1

If there were any negative-weight cycles, the program would output:

Graph contains negative weight cycle

Result:

- The program successfully computes the shortest paths using Bellman-Ford.
- It detects and reports negative-weight cycles if present.
- The experiment demonstrates how to handle graphs with negative edges in practical networking scenarios.

Experiment 4

Given a graph derive the routing table using distance vector routing and link state routing algorithm

Objective:

- To understand how routing tables are built using **Distance Vector Routing** and **Link State Routing** algorithms.
- To implement both algorithms in Java for a given network graph.
- To compare how DVR and LSR work to find shortest paths to all destinations.
- To observe how routing information is updated and maintained at each router.

Explanation:

Distance Vector Routing (DVR):

- Each router maintains a **distance vector table** with the cost to reach every other node.
- Routers exchange their distance vectors with immediate neighbors.
- Each router updates its table based on the **Bellman-Ford equation**: if a shorter path is found via a neighbor, the routing table is updated.
- Routing information is periodically exchanged until no changes occur.

Key Points:

- Simple to implement.
- Slow convergence for large networks.
- Used in protocols like **RIP (Routing Information Protocol)**.

Link State Routing (LSR):

- Each router builds a **complete map (topology)** of the network using link state advertisements.
- Each router runs **Dijkstra's algorithm** to compute the shortest path to every other router.
- Unlike DVR, every router has the same network view, resulting in consistent routing tables.

Key Points:

- Faster convergence than DVR.
- Requires more processing power and memory.
- Used in protocols like **OSPF (Open Shortest Path First)** and **IS-IS**.

Source Code

4.1 Simplified DVR

```
import java.util.Arrays;
public class SimplifiedDVR {
    static int INF = 9999; // Representing infinity
        // Distance Vector Routing Algorithm
    public static void distanceVector(int[][] graph) {
        int n = graph.length;
        int[][] dist = new int[n][n];
                // Initialize distance vector with the graph itself
        for (int i = 0; i < n; i++)
            dist[i] = Arrays.copyOf(graph[i], n);
                // Update distance vector using neighbors
        for (int k = 0; k < n; k++) { // k is intermediate node
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
        // Print result
        for (int i = 0; i < n; i++)
            System.out.println("Router " + i + " distance vector: " +
Arrays.toString(dist[i]));
    }
    public static void main(String[] args) {
        int[][] graph = {
            {0, 2, INF, 1},
            {2, 0, 3, 2},
            {INF, 3, 0, 1},
            {1, 2, 1, 0}
        };
        distanceVector(graph);
    }
}
```

Output:

```
Router 0 shortest paths: [0, 2, 2, 1]
Router 1 shortest paths: [2, 0, 3, 2]
Router 2 shortest paths: [2, 3, 0, 1]
Router 3 shortest paths: [1, 2, 1, 0]
```

4.2 Simplified LSR

```
import java.util.Arrays;
public class SimplifiedLSR {
    static int INF = 9999;
    // Dijkstra's algorithm for Link State Routing
    public static void dijkstra(int[][] graph, int src) {
        int n = graph.length;
        int[] dist = new int[n];
        boolean[] visited = new boolean[n];
        Arrays.fill(dist, INF);
        dist[src] = 0;
        for (int i = 0; i < n - 1; i++) {
            int u = minDistance(dist, visited);
            visited[u] = true;
            for (int v = 0; v < n; v++) {
                if (!visited[v] && graph[u][v] != INF && dist[u] + graph[u][v] < dist[v]) {
                    dist[v] = dist[u] + graph[u][v];
                }
            }
        }
        System.out.println("Router " + src + " shortest paths: " +
Arrays.toString(dist));
    }
    public static int minDistance(int[] dist, boolean[] visited) {
        int min = INF, minIndex = -1;
        for (int i = 0; i < dist.length; i++)
            if (!visited[i] && dist[i] <= min) {
                min = dist[i];
                minIndex = i;
            }
        return minIndex;
    }
    public static void main(String[] args) {
        int[][] graph = {
            {0, 2, INF, 1},
            {2, 0, 3, 2},
            {INF, 3, 0, 1},
            {1, 2, 1, 0}
        };
        for (int i = 0; i < graph.length; i++)
            dijkstra(graph, i);
    }
}
```

Output

```
Router 0 shortest paths: [0, 2, 2, 1]
Router 1 shortest paths: [2, 0, 3, 2]
```

Router 2 shortest paths: [2, 3, 0, 1]
Router 3 shortest paths: [1, 2, 1, 0]

Result:

- The DVR and LSR algorithms correctly computed the shortest paths between all nodes.
- Both methods produced consistent routing tables for this simple graph.
- Students gain hands-on experience in how routing information is exchanged or computed for efficient packet delivery in real-world networks.

Experiment 5

Try out some simple subnetting problems.

Objective:

1. To understand the concept and importance of subnetting in IP networking.
2. To calculate hosts per subnet, subnet mask, first host, and last host for a given IP address and CIDR notation.
3. To implement a simple Java program that automates subnetting calculations.
4. To interpret and verify subnet details for network design.

Explanation:

- Subnetting is the process of dividing a large IP network into smaller, more manageable subnetworks (subnets) to optimize address usage, improve security, and enhance network performance.
- The IP address consists of two parts: the Network ID and the Host ID. Subnetting borrows bits from the Host ID to create multiple smaller networks.
- CIDR (Classless Inter-Domain Routing) notation (e.g., /24) indicates how many bits are used for the network part.
- The number of hosts per subnet is calculated as 2^h - 2 where h is the number of host bits (subtracting 2 for network and broadcast addresses).
- The program takes an IP address and CIDR as input, then calculates:
  - Subnet mask
  - Number of hosts
  - First valid host address
  - Last valid host address

Source Code

```
import java.util.Scanner;

public class SubnettingExample {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input: IP Address and CIDR notation
        System.out.print("Enter IP address (e.g., 192.168.1.0): ");
        String ipAddress = scanner.nextLine();

        System.out.print("Enter CIDR notation (e.g., /24): ");
        int cidr = scanner.nextInt();
```

```
        int totalBits = 32; // IPv4 has 32 bits
        int networkBits = cidr;
        int hostBits = totalBits - networkBits;

        // Hosts per subnet: 2^hostBits - 2 (for network and broadcast)
        int hostsPerSubnet = (int) Math.pow(2, hostBits) - 2;
        System.out.println("Hosts per subnet: " + hostsPerSubnet);

        String subnetMask = calculateSubnetMask(cidr);
        System.out.println("Subnet Mask: " + subnetMask);

        String firstHost = calculateFirstHost(ipAddress, cidr);
        System.out.println("First Host: " + firstHost);

        String lastHost = calculateLastHost(ipAddress, cidr);
        System.out.println("Last Host: " + lastHost);

        scanner.close();
    }

    private static String calculateSubnetMask(int cidr) {
        int mask = 0xFFFFFFFF << (32 - cidr);
        return String.format("%d.%d.%d.%d",
                (mask >> 24) & 0xFF,
                (mask >> 16) & 0xFF,
                (mask >> 8) & 0xFF,
                mask & 0xFF);
    }

    private static String calculateFirstHost(String ipAddress, int cidr) {
        String[] octets = ipAddress.split("\\.");
        int[] ip = new int[4];
        for (int i = 0; i < 4; i++) {
            ip[i] = Integer.parseInt(octets[i]);
        }
        int maskBits = 32 - cidr;
        ip[3] = (ip[3] & (0xFF << maskBits)) + 1;
        return String.format("%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
    }

    private static String calculateLastHost(String ipAddress, int cidr) {
        String[] octets = ipAddress.split("\\.");
        int[] ip = new int[4];
        for (int i = 0; i < 4; i++) {
            ip[i] = Integer.parseInt(octets[i]);
```

```
        }
        int maskBits = 32 - cidr;
        int hostBits = (1 << maskBits) - 2;
        ip[3] = (ip[3] & (0xFF << maskBits)) + hostBits;
        return String.format("%d.%d.%d.%d", ip[0], ip[1], ip[2], ip[3]);
    }
}
```

Sample Output/Results:

Input:

Enter IP address (e.g., 192.168.1.0): 192.168.1.0

Enter CIDR notation (e.g., /24): 26

Output:

Hosts per subnet: 62

Subnet Mask: 255.255.255.192

First Host: 192.168.1.1

Last Host: 192.168.1.62

Explanation of Output:

- A /26 subnet mask means 26 bits are used for the network; 6 bits are left for hosts.
- Number of hosts = 2^6 - 2 = 62 usable hosts per subnet.
- Subnet Mask: 255.255.255.192 represents the /26 prefix.
- First usable host: 192.168.1.1 (immediately after the network address).
- Last usable host: 192.168.1.62 (just before the broadcast address).

Result:

- The program successfully calculates subnet details for the given network.
- It helps visualize how IP addressing, subnet mask, and host ranges are determined in real-world networking.

Experiment 6

Using TCP/IP sockets, write a client–server program to make the client send the file name and the server send back the contents of the requested file if present. Implement the above program using message queues or FIFOs as IPC channels; simulate the network using Cisco Packet Tracer.

Objective:

1. To understand client–server communication using the TCP/IP protocol suite.
2. To simulate the process of file request and file delivery over a network.
3. To configure and demonstrate Dynamic Host Configuration Protocol (DHCP) for automatic IP address assignment.
4. To practically observe how IPC channels (like message queues or FIFOs) can conceptually support inter-process communication between client and server.
5. To verify connectivity and successful data exchange using Cisco Packet Tracer.

Explanation:

In real-world networks, client-server communication involves:

- A client requesting a resource (e.g., file name).
- A server processing the request and sending back the file content if available.
- Communication typically happens over a TCP connection, ensuring reliable delivery.

In this lab, you simulate this concept using Cisco Packet Tracer:

- A DHCP server dynamically allocates IP addresses to clients (PCs).
- PCs send simulated requests to the server.
- Message transfer represents how sockets and IPC mechanisms (like message queues/FIFOs) would work at the system level.

Steps Implemented in Cisco Packet Tracer:

1. Create Network Topology:
    - 1 x 2960-24TT Switch
    - 3 x PCs (PC0, PC1, PC2)
    - 1 x Server PT
2. Connect Devices:
    - Use Copper Straight-Through cables to connect all PCs and the server to the switch.
3. Configure the Server:
    - Set the server IP to 192.168.1.35.
    - Configure DHCP service:
        - Default Gateway: 192.168.1.35
        - Start IP Address: 192.168.1.40

- Maximum Users: 3

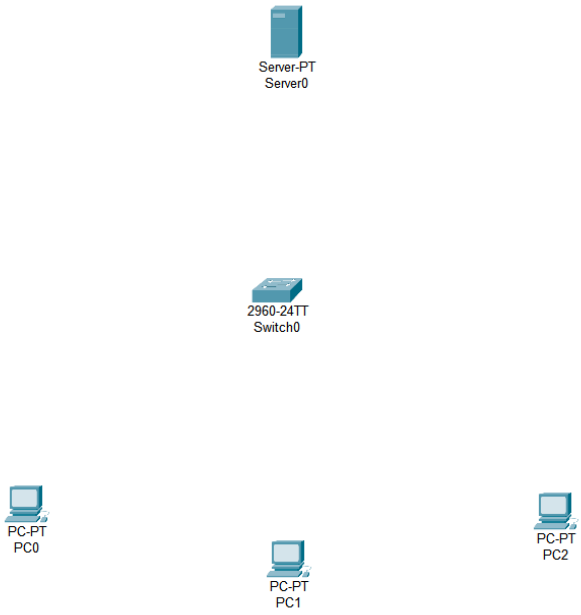
4. Configure PCs for DHCP:
   - Each PC uses DHCP to get an IP address automatically (e.g., 192.168.1.40 to 192.168.1.42).
5. Verify Communication:
   - Use Add Simple PDU tool to send messages (representing file requests) from PCs to the server.
   - Observe the successful delivery in simulation mode.
6. IPC Concept:
   - Though Cisco Packet Tracer doesn't execute actual socket or IPC code, the logical data exchange shows how message queues or FIFOs would pass data between processes on the server side.

Step 1: Select the required Network Devices

- Navigate the Network Devices section.
- Click on Switches.
- Drag and drop the 2960-24TT Switch into workspace.

Step 2: Select the required End Devices

- Navigate the End devices section.
- Drag and drop three PC PTs into the workspace.
- Finally, drag and drop a Server PT into the workspace.



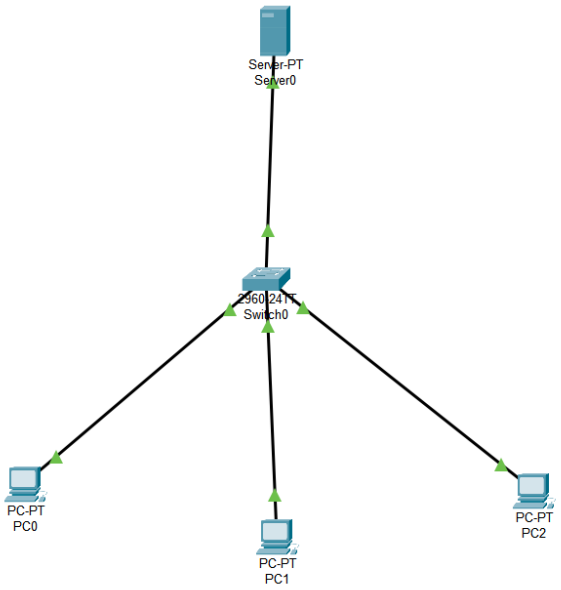
Step 3: Connect the PCs to the Switch

- Select the Copper Straight-Through cable from the connections menu.
- Connect the FastEthernet0 port of PC0 to the FastEthernet0/1 port of Switch0.
- Connect the FastEthernet0 port of PC1 to the FastEthernet0/2 port of Switch0.

- Connect the FastEthernet0 port of PC2 to the FastEthernet0/3 port of Switch0.

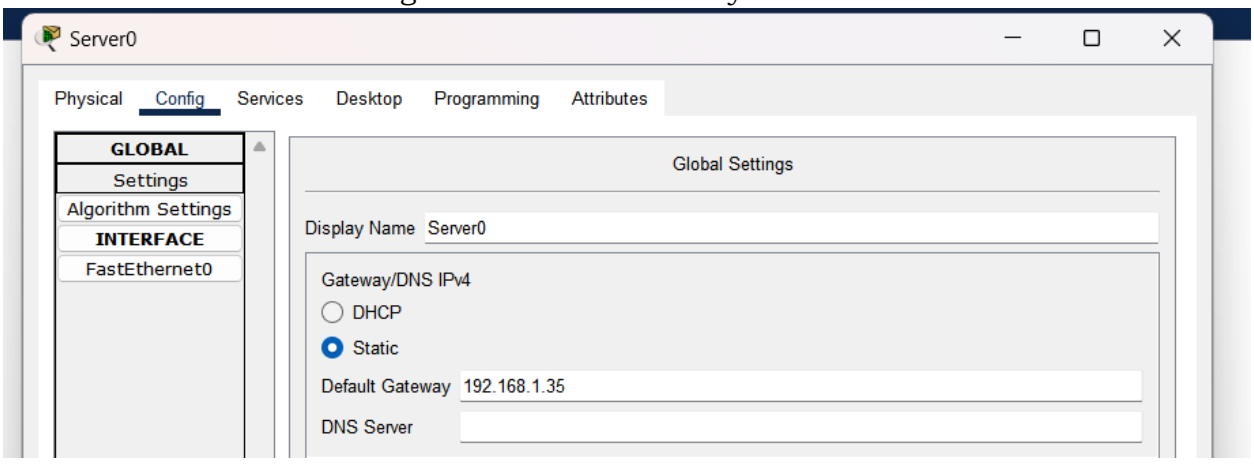
Step 4: Connect the Switch to the Server

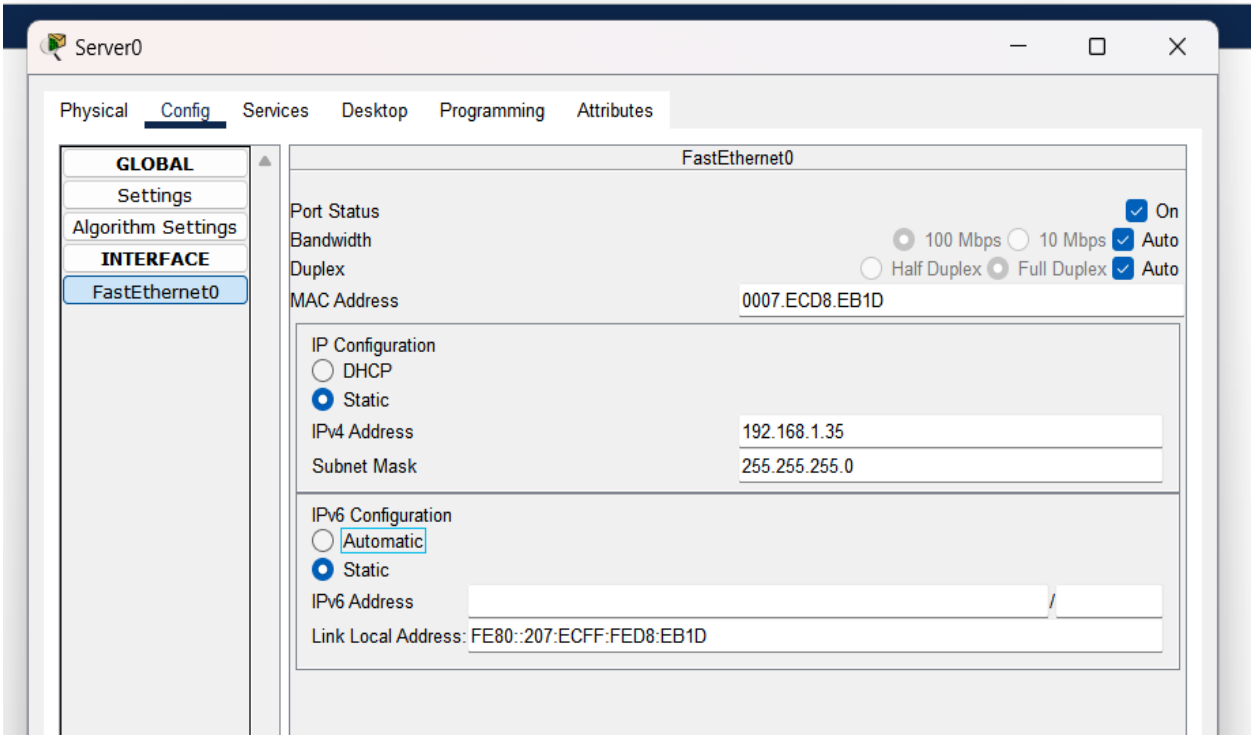
- Again, using the Copper Straight-Through cable, connect the FastEthernet0 port of Server0 to the FastEthernet0/4 port of Switch0.



Step 5: Configure the server

- Click on the Server PT and select the Config tab.
- In the Config tab, in Global Settings, enter the default gateway as 192.168.1.35.
- Next select FastEthernet0 under INTERFACE.
- Under IP Configuration, type 192.168.1.35 in the IPv4 Address field. The Subnet Mask will be generated automatically.

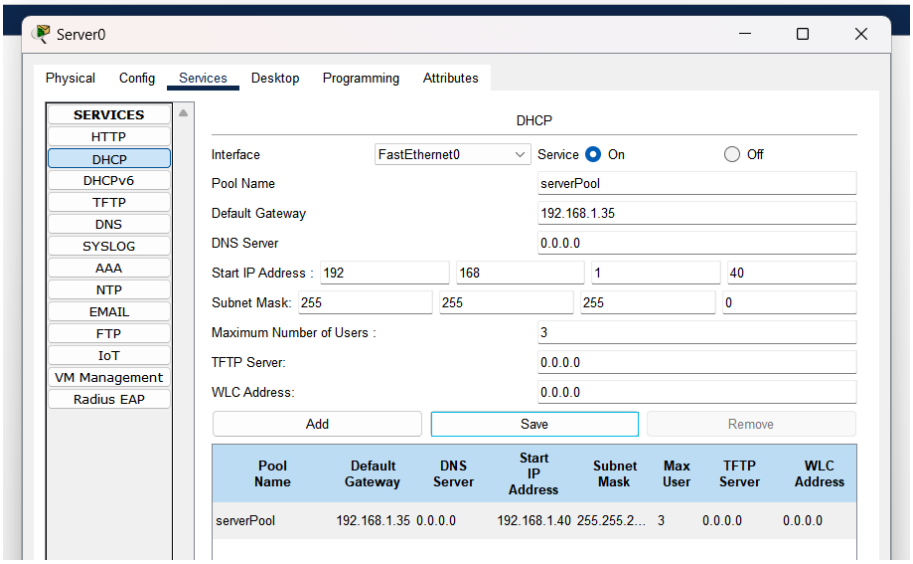




- Next, select the Services tab and click on DHCP under SERVICES.
- Turn on the DHCP service and enter the following details:

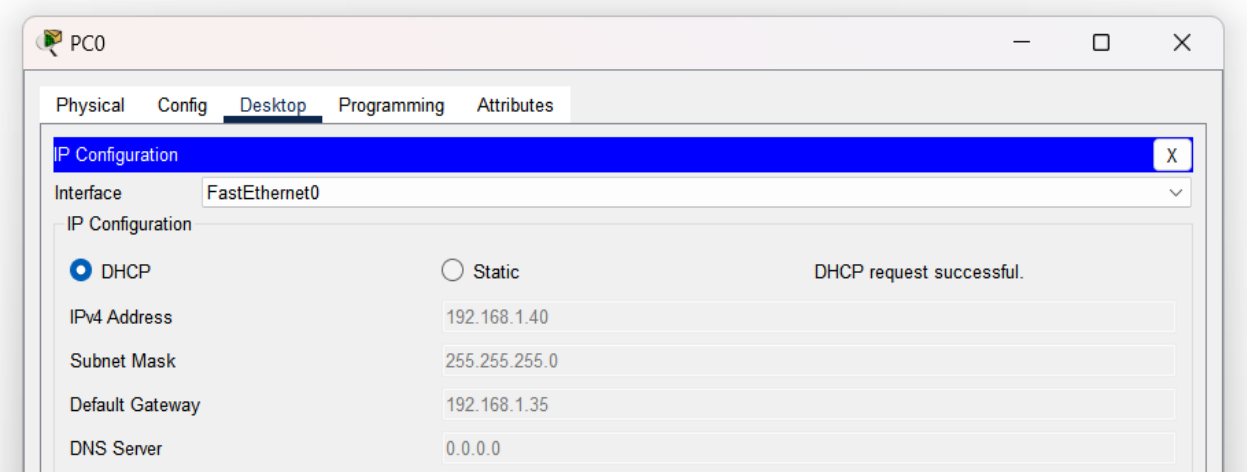
Default Gateway: 192.168.1.35
Start IP Address: 192.168.1.40
Maximum Number of Users: 3 (3 PCs)

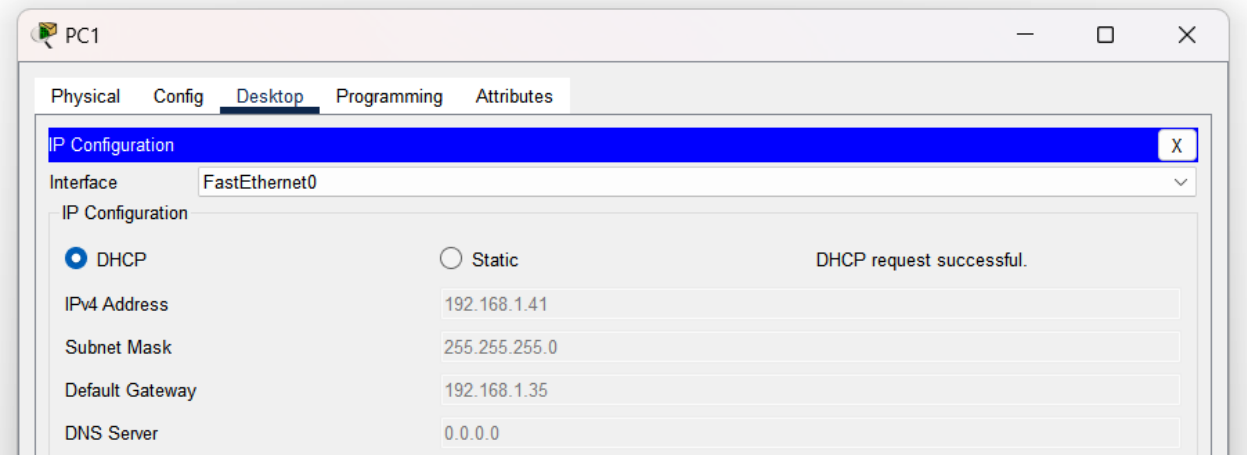
- Click Save to apply the configuration.

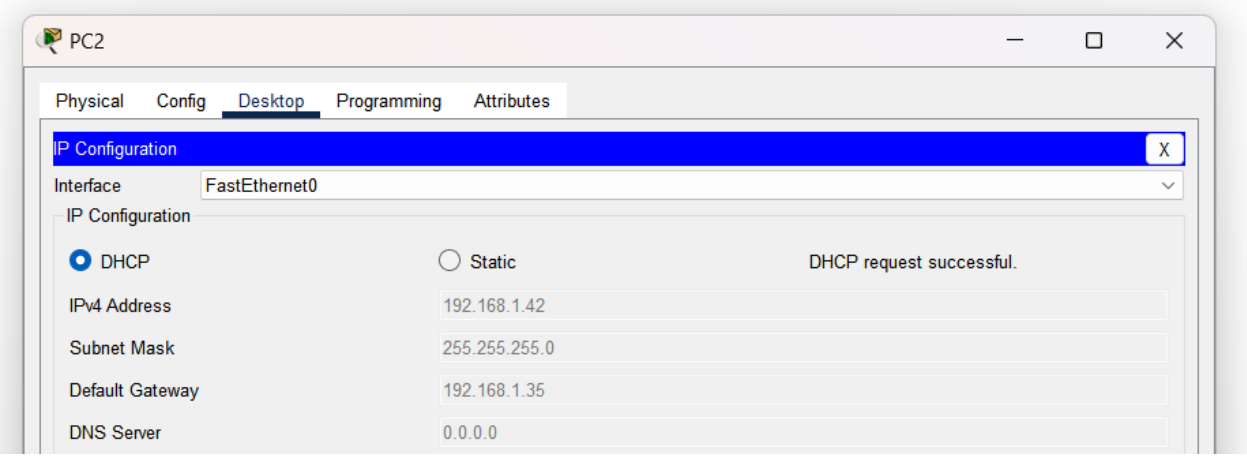


Step 6: Configure the PCs

- Click on PC0, under the Desktop tab, select IP Configuration.
- Under IP Configuration, click on DHCP to request an IP Adress automatically.
- Verify that the DHCP request is successful.
- Repeat the same process for PC1 and PC2.
- The IP Addresses will be from 192.168.1.40 to 192.168.1.42

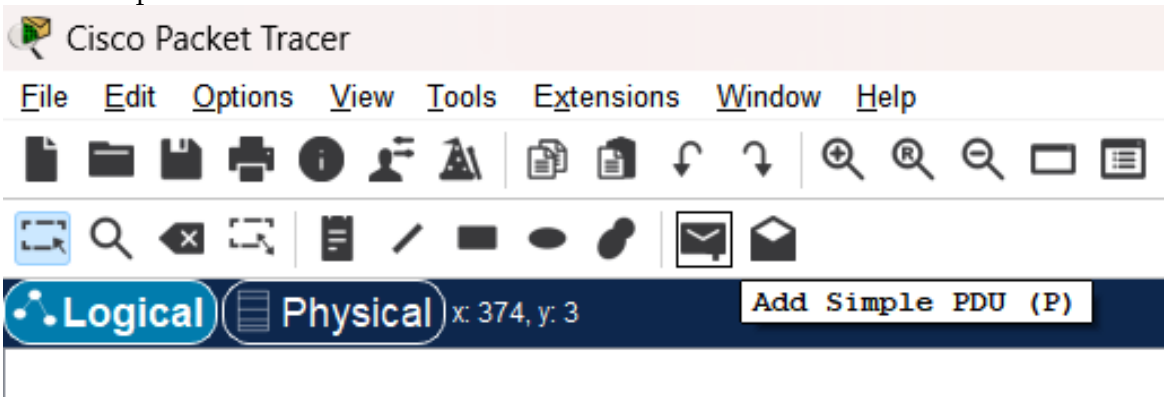






Step 7: Verify the Connectivity

- Select the Add Simple PDU tool.
- Send a PDU from PC0 to the Server.
- Run the simulation to ensure successful message delivery.
- Repeat the same for the other PCs.



Result:

- The experiment demonstrates client-server communication using TCP/IP concepts and dynamic addressing with DHCP.
- The network was successfully simulated in Cisco Packet Tracer, verifying IP address allocation, request transmission, and response.
- The scenario illustrates how message queues/FIFOs can be used as IPC mechanisms between client and server processes on a real OS.

Conclusion:

Students gain practical exposure to configuring a basic client-server network, setting up DHCP, and understanding the flow of data requests/responses in a real-world context using simulation tools.

Experiment 7

Implement a webserver program to fetch a URL request and display the home page of the same in the browser ; using cisco packet tracer

Objective:

- To understand how a web server hosts web pages in a network.
- To simulate how a client PC fetches a web page by sending an HTTP request.
- To configure HTTP services on a server device in Cisco Packet Tracer.
- To visualize the data flow when a user types a URL in a browser.
- To verify successful communication between clients and server using Cisco Packet Tracer.

Explanation:

In real-world networking, when a user enters a URL in a browser:

1. The browser sends an HTTP GET request to the web server's IP address.
2. The web server processes the request and sends back the HTML page (or other resources).
3. The browser renders the received page for the user.

In this simulation, you:

- Use Cisco Packet Tracer to set up a small LAN with a server configured as a web server.
- Configure HTTP service on the server.
- Use PCs as clients to send HTTP requests using the Web Browser utility.
- Verify that the requested web page (e.g., the home page) is successfully fetched and displayed.

Steps:

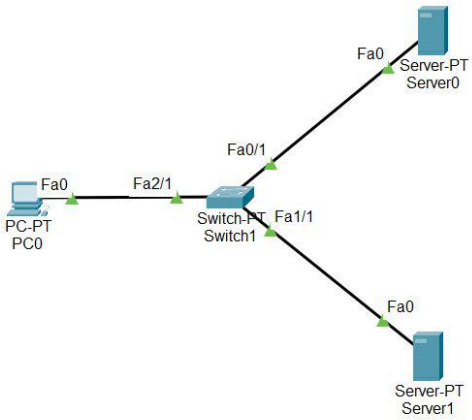
Step 1: First, open the cisco packet tracer desktop and select the devices given below:

S.NO	Device	Model Name	Qty.
1.	PC	PC	1
2.	switch	PT-switch	1
3.	server	Server-PT	2

IP Addressing Table

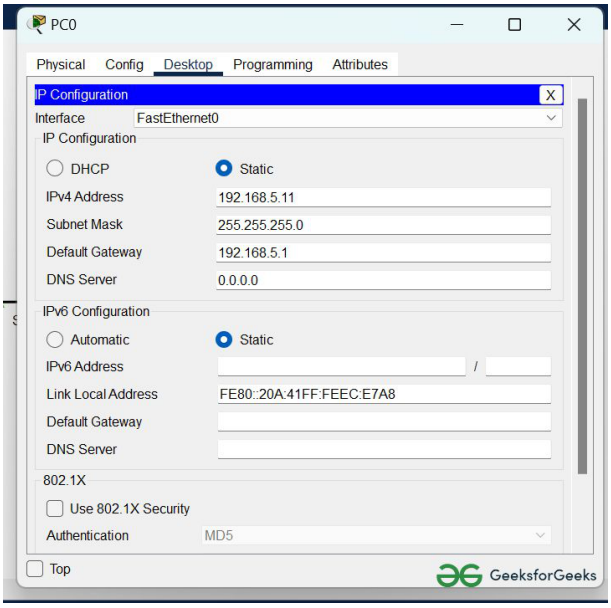
S.NO	Device	IPv4 Address	Subnet Mask	Default Gateway	DNS
1.	PC0	192.168.5.11	255.255.255.0	192.168.5.1	nil
2.	server0	192.168.5.12	255.255.255.0	192.168.5.1	nil
3.	DNS	192.168.5.13	255.255.255.0	192.168.5.1	192.168.5.13

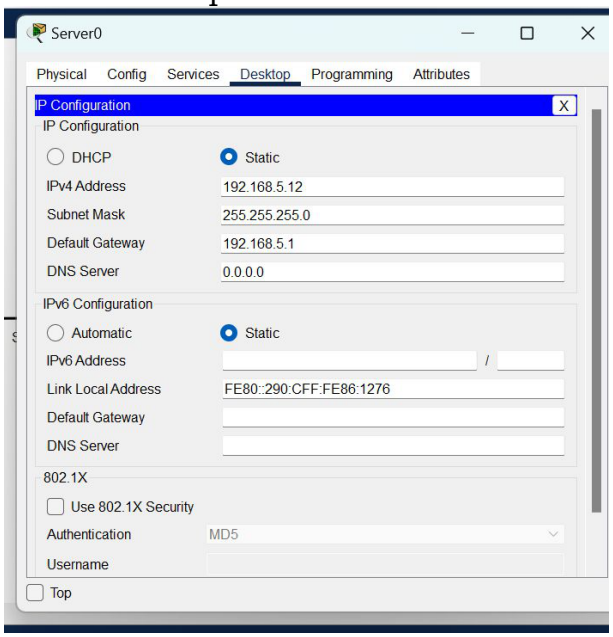
- Then, create a network topology as shown below the image.
- Use an Automatic connecting cable to connect the devices with others.

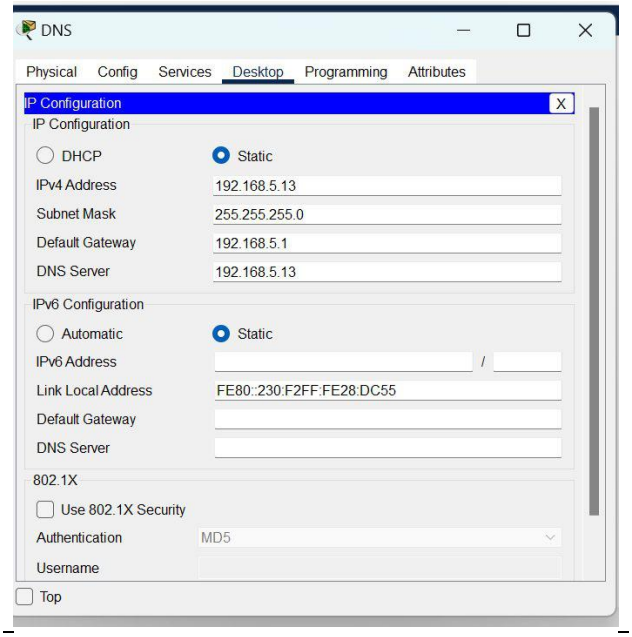


Step 2: Configure the PCs (hosts) Server0 and DNS with IPv4 address and Subnet Mask according to the IP addressing table given above.

- To assign an IP address, click on the device.
- Then, go to desktop and IP configuration and there you will find IPv4 configuration.
- Fill IPv4 address and subnet mask and other inputs.

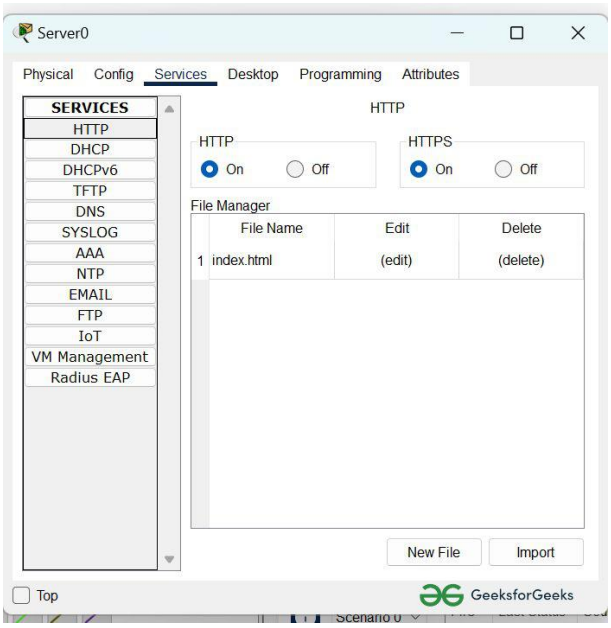


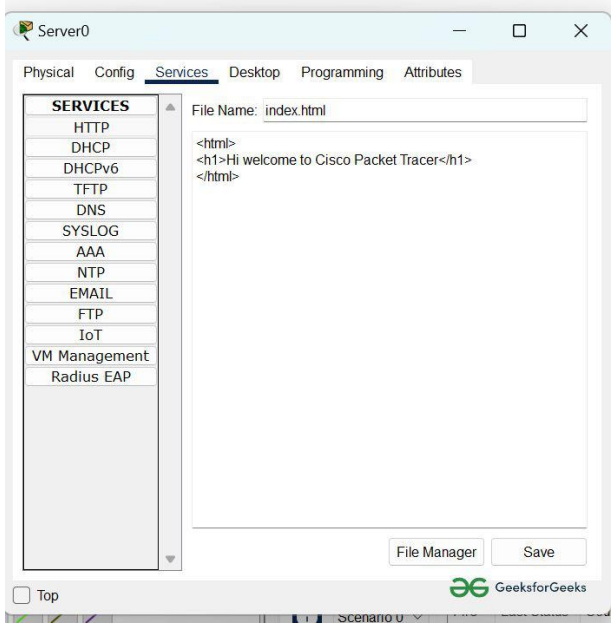




Step 3: Configure the HTTP sever0

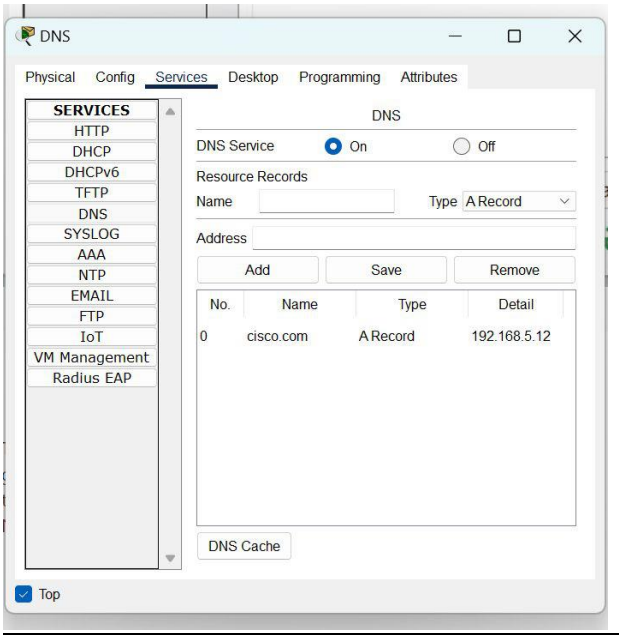
- To configure the HTTP server.
- Go to services then click on HTTP
- Then delete all of the files except the index.html and edit it.





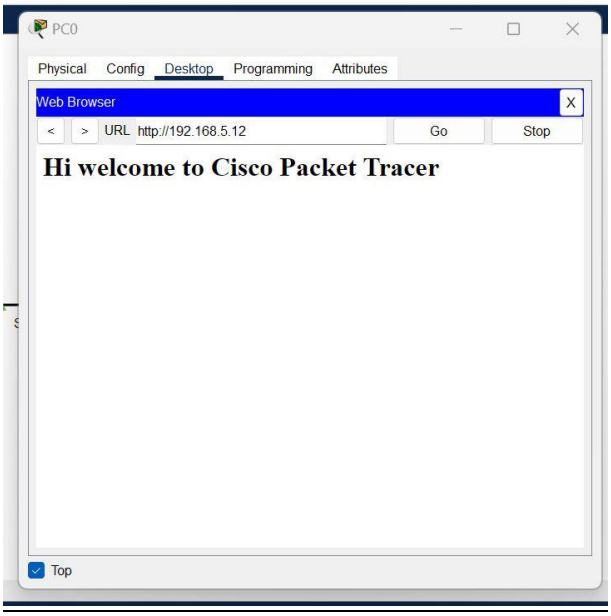
Step 4: Configure the DNS server

- To configure the DNS server.
- Go to services then click on DNS.
- Then turn on the DNS services.
- Name the server cisco.com and type address 192.168.5.12
- And add the record.



Step 5: Verify the server by using the web browser in the Host.

- Enter the IP address of server0 and click on GO.
- It will show the results.



Result:

- Successfully simulated a web server using Cisco Packet Tracer.
- Clients could send HTTP requests and receive web content.
- Students understood how URL requests are resolved to IP addresses and how HTTP delivers web pages.

Conclusion:

This experiment demonstrates the complete flow of HTTP communication in a simple LAN and shows how web servers serve web content to multiple clients.

Experiment 8

Implement a simple DNS server to resolve the IP address for the given domain name; using cisco packet tracer

Objective:

1. To understand the **Domain Name System (DNS)** and how it maps domain names to IP addresses.
2. To configure and simulate a simple **DNS server** in **Cisco Packet Tracer**.
3. To set up client PCs to query the DNS server and resolve domain names.
4. To test and verify domain resolution and connectivity.

Explanation:

In real-world networking:

- **DNS** acts like a phonebook for the Internet — it translates **human-readable domain names** (e.g., www.example.com) to **machine-readable IP addresses** (e.g., 192.168.1.10).
- Without DNS, users would have to remember IP addresses for every website.
- The DNS process involves:
    1. A **client PC** sends a DNS query for a domain.
    2. The **DNS server** looks up the domain and responds with the mapped IP.
    3. The client then uses that IP to communicate with the target server (e.g., web server).

Steps:

Step 1: Create the Network Topology

- Drag and drop:
    - **1 Switch**
    - **1 Server**
    - **At least 2 PCs**
- Connect all devices using **Copper Straight-Through cables**.

Step 2: Configure the DNS Server

1. Click on the **Server**.
2. Go to **Config Tab → FastEthernet0**:
    - Set IP Address: 192.168.1.100
    - Subnet Mask: 255.255.255.0
3. Go to **Services Tab → DNS**:
    - Turn **DNS Service ON**
    - Under **DNS Records**, add:
        - **Name:** www.labdns.com

        - **Address:** 192.168.1.200 *(e.g., the IP of your web server)*
    - Click **Add** to save the mapping.

Step 3: Configure a Web Server (Optional)

- If testing web browsing:
    - Click the same or another server.
    - Under **Services → HTTP**, ensure the **HTTP Service** is ON.
    - This server should use the IP you mapped in the DNS record (e.g., 192.168.1.200).

Step 4: Configure PCs

1. Click on **PC0**, then **Desktop → IP Configuration**.
    - IP Address: 192.168.1.10 (example)
    - Subnet Mask: 255.255.255.0
    - Default Gateway: 192.168.1.1 (optional)
    - **DNS Server:** 192.168.1.100 *(IP of your DNS server)*
2. Repeat for other PCs.

Step 5: Test DNS Resolution

- Click on **PC0 → Desktop → Web Browser**.
    - Enter: http://www.labdns.com
    - The PC will query the DNS server to resolve the domain name to 192.168.1.200.
    - If your web server is set up, the home page should display.
- You can also test using **Command Prompt → nslookup**:
    - Type nslookup www.labdns.com
    - It should return: 192.168.1.200.

Result:

- Successfully configured a **DNS server** in Cisco Packet Tracer.
- Verified that **clients** can resolve domain names to IP addresses.
- Confirmed the resolved IP is used for further communication (e.g., web browsing).

Conclusion:

This experiment shows how a **DNS server** functions within a network to translate domain names into IP addresses, making network access more user-friendly.

(refer: https://bmsce.ac.in/Content/IS/CN_-1_Sample_Lab_Programs.pdf)

Computer Networks Lab Viva Questions & Answers

1. Line Coding Techniques:

Q1. What is line coding?
A1. Line coding is the process of converting digital data into a digital signal for transmission over a physical medium.

Q2. Name three types of line coding techniques.
A2. Unipolar NRZ, Polar NRZ, Manchester, and Differential Manchester.

Q3. Which line coding technique is self-clocking?
A3. Manchester and Differential Manchester are self-clocking because they have transitions for synchronization.

Q4. What is the main disadvantage of Unipolar NRZ?
A4. It has a DC component and poor synchronization.

2. CRC Error Detection:

Q5. What is CRC?
A5. CRC (Cyclic Redundancy Check) is an error-detecting code that detects accidental changes to raw data in digital networks.

Q6. Why do we use polynomial division in CRC?
A6. To generate a checksum (redundancy bits) that helps detect errors at the receiver side.

Q7. What does a zero remainder mean in CRC checking?
A7. It indicates error-free data transmission.

3. Routing Algorithms:

Q8. What is the purpose of the Bellman-Ford algorithm?
A8. To find the shortest path from a single source to all other vertices in a weighted graph, even with negative weights.

Q9. What is the difference between Distance Vector Routing and Link State Routing?
A9. DVR uses periodic updates with neighbors and Bellman-Ford, while LSR builds a full network map and uses Dijkstra's algorithm.

Q10. Name a protocol that uses Distance Vector Routing.
A10. RIP (Routing Information Protocol).

Q11. Name a protocol that uses Link State Routing.
A11. OSPF (Open Shortest Path First).

4. Subnetting:

Q12. What is subnetting?
A12. Subnetting divides a large network into smaller sub-networks to optimize IP address usage and improve network management.

Q13. What is the formula to calculate hosts per subnet?

A13. 2n−22^{n} - 22n−2 where n is the number of host bits; 2 addresses are reserved for network ID and broadcast.

Q14. What is CIDR?
A14. CIDR (Classless Inter-Domain Routing) allows flexible subnet masks, not limited to class A, B, or C.

5. DHCP & Client-Server:
Q15. What is DHCP?
A15. Dynamic Host Configuration Protocol automatically assigns IP addresses to hosts on a network.

Q16. What is the difference between static and dynamic IP addressing?
A16. Static IPs are manually assigned; dynamic IPs are assigned automatically by DHCP.

Q17. What is the role of a socket in a client-server model?
A17. Sockets provide an endpoint for sending and receiving data between client and server.

6. Web Server & URL Request:
Q18. What protocol does a web browser use to fetch a web page?
A18. HTTP or HTTPS.

Q19. What happens when you enter a URL in a web browser?
A19. The browser resolves the domain name using DNS, then sends an HTTP request to the server's IP address and fetches the page.

7. DNS:
Q20. What is DNS?
A20. Domain Name System translates human-readable domain names into IP addresses.

Q21. Why do we need DNS?
A21. So users don't have to remember numeric IP addresses for every website.

Q22. What command can you use to check DNS resolution?
A22. nslookup or ping with a domain name.

8. General
Q23. What is a protocol?
A23. A set of rules that govern data communication between devices.

Q24. What is the OSI model?
A24. A seven-layer conceptual model that standardizes network communication.

Q25. Which OSI layer does routing belong to?
A25. Network layer (Layer 3).