

Computer Networks Lab

ICS 653

BACHELOR OF TECHNOLOGY

In

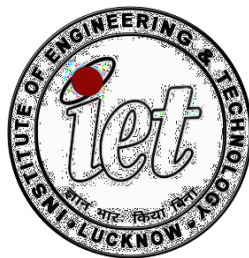
COMPUTER SCIENCE AND ENGINEERING

Submitted By:

MOHD TAHA KHAN 2200520100037

Under the guidance of

Abhishek Singh



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Institute of Engineering & Technology (IET) College in
Lucknow, Uttar Pradesh

TABLE OF CONTENT

<u>S.No.</u>	<u>Topic</u>	<u>Pg.No.</u>
1.	Study of Network Simulator (NS) and Congestion Control Algorithms	1-4
2.	Study and configuration of Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc.	5-8
3.	To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.	9-12
4.	To implement Stop-and-Wait Protocol and Sliding Window Protocol using C programming.	13-15
5.	To write a Java program to simulate the Address Resolution Protocol (ARP) using TCP.	16-17
6.	Write a Program Simulating PING and TRACEROUTE Commands.	18-20
7.	Running and using services/commands like ping, trace route, nslookup, arp, telnet, ftp, etc.	21-25
8.	Link State Routing, Flooding, and Distance Vector Routing.	26-30
9.	Study of Socket Programming and Client-Server mode using TCP and UDP	31-34
10.	Configuration of router, hub, switch etc. (using real devices or simulators).	35-36
11.	To understand and implement subnetting for a given IP address, calculate subnet masks, identify network and host portions, and determine valid subnets and host ranges.	37-40
12.	Applications using TCP Sockets - Echo, Chat and File Transfer	41-46

Practical No: 1

Aim:

Study of Network Simulator (NS) and Simulation of Congestion Control Algorithms using NS.

1. Introduction to Network Simulator (NS2)

NS Overview:

- **NS Programming:** A quick start guide to using NS2 for network simulations.
- **Case Study I:** A simple wireless network simulation.
- **Case Study II:** Creating a new agent in NS.
- **NS Status:** Regular updates, with the latest release being ns-2.26 (Feb 2003).
- **Platform Support:** NS2 runs on FreeBSD, Linux, Solaris, Windows, and Mac.

NS Functionalities:

- **Routing:** Simulates various routing protocols.
- **Transportation:** Supports transport layer protocols like TCP and UDP.
- **Traffic Sources:** Allows simulation of different traffic patterns.
- **Queuing Disciplines:** Supports various queuing mechanisms like RED (Random Early Detection).
- **Quality of Service (QoS):** Enables QoS simulations for different network conditions.

Wireless Networking:

- **Ad hoc Routing:** Simulates mobile ad hoc networks.
- **Mobile IP:** Supports mobile IP simulations.
- **Sensor-MAC:** Simulates MAC protocols for sensor networks.

Tracing and Visualization:

NS2 provides tools for tracing and visualizing network events, which are crucial for analyzing simulation results.

2. Network Simulators Overview

Types of Network Simulators:

- **GUI-Driven Simulators:** Most commercial simulators are GUI-driven, making them user-friendly.
- **CLI-Driven Simulators:** Some simulators, like NS2, are command-line driven, offering more flexibility and control.

Discrete Event Simulation:

Network simulators use discrete event simulation, where events (like packet arrivals) are processed in order, triggering future events.

Examples of Network Simulators:

- **NS (Open Source):** Widely used in research.
- **OPNET (Proprietary):** Popular in industry.
- **NetSim (Proprietary):** Another widely used simulator.

Uses of Network Simulators:

- **Cost-Effective:** Simulators are cheaper and faster than setting up physical testbeds.
 - **Controlled Environment:** Allows testing of new protocols or network changes in a reproducible environment.
 - **Complex Network Simulation:** Simulators can model complex networks with various nodes, links, and protocols.
-

3. Congestion Control Algorithms

- **Packet Loss:**
 - Packet loss occurs when packets fail to reach their destination, often due to queue overflow in network nodes.
 - Packet loss is a key metric in evaluating congestion control algorithms.
 - **Throughput:**
 - The average rate of successful message delivery over a communication channel.
 - Measured in bits per second (bps) or packets per second.
 - **Delay:**
 - Time taken for a packet to travel from source to destination.
 - High delay can negatively impact network performance.
 - **Queue Length:**
 - Important metric for evaluating how well a congestion control algorithm manages network congestion.
-

4. Simulation of Congestion Control Algorithm

Algorithm:

1. Create a simulation object.
2. Set the routing protocol.
3. Trace packets and links to NAM trace and trace file.

4. Create nodes and define their topology (e.g., octagon layout).
5. Add a sink agent to a node.
6. Connect the source and sink.

Program:

```
set ns [new Simulator]
set nr [open thro_red.tr w]
$ns trace-all $nr
set nf [open thro.nam w]
$ns namtrace-all $nf

proc finish {} {
    global ns nr nf
    $ns flush-trace
    close $nf
    close $nr
    exec nam thro.nam &
    exit 0
}

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]

$ns duplex-link $n0 $n3 1Mb 10ms RED
$ns duplex-link $n1 $n3 1Mb 10ms RED
$ns duplex-link $n2 $n3 1Mb 10ms RED
$ns duplex-link $n3 $n4 1Mb 10ms RED
$ns duplex-link $n4 $n5 1Mb 10ms RED
$ns duplex-link $n4 $n6 1Mb 10ms RED
$ns duplex-link $n4 $n7 1Mb 10ms RED

$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient middle
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n7 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n6 $n4 orient left

set udp0 [new Agent/UDP]
$ns attach-agent $n2 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n5 $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n1 $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
```

```
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null1 [new Agent/Null]
$ns attach-agent $n6 $null1
$ns connect $udp1 $null1

$ns run
```

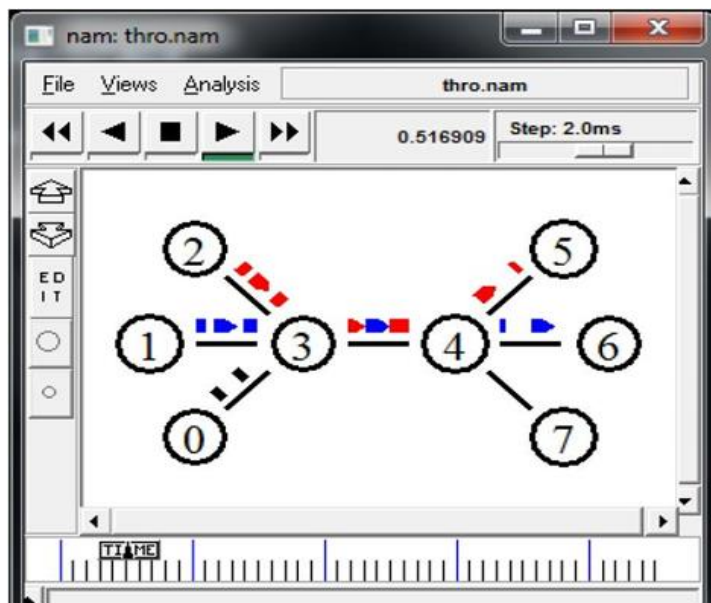
5. Output and Result

Result:

The congestion control algorithm was successfully simulated using NS2. The simulation demonstrated how different traffic sources interact with the network, and how the RED (Random Early Detection) queuing discipline helps manage congestion.

6. Conclusion

This experiment provided a hands-on experience with NS2, a powerful network simulator. By simulating a congestion control algorithm, we were able to observe the effects of packet loss, throughput, delay, and queue length on network performance. This practical exercise is essential for understanding how congestion control mechanisms work in real-world networks.



Practical No: 2

Aim:

Study and configuration of Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc.

1. Introduction to Network Simulation Tools

Network simulation tools are essential for designing, testing, and analyzing network behavior in a controlled environment. These tools allow network engineers and researchers to simulate complex network scenarios without the need for physical hardware. Some of the most widely used network simulation tools include:

1. **Cisco Packet Tracer**
2. **NetSim**
3. **OMNeT++**
4. **NS2 (Network Simulator 2)**
5. **NS3 (Network Simulator 3)**

Each tool has its unique features, strengths, and use cases. Below, we will explore these tools in detail.

2. Overview of Network Simulation Tools

2.1 Cisco Packet Tracer

- **Purpose:** Cisco Packet Tracer is a network simulation tool designed by Cisco for teaching and learning networking concepts.
- **Features:**
 - GUI-based interface for easy network design.
 - Supports simulation of Cisco devices like routers, switches, and firewalls.
 - Allows real-time packet tracing and visualization.
 - Ideal for beginners and educational purposes.
- **Use Cases:**
 - Teaching CCNA (Cisco Certified Network Associate) concepts.
 - Simulating small to medium-sized networks.
 - Testing network configurations before deployment.

2.2 NetSim

- **Purpose:** NetSim is a commercial network simulator used for simulating wired and wireless networks.

- **Features:**
 - Supports a wide range of protocols (TCP/IP, VoIP, LTE, etc.).
 - Provides detailed performance analysis and reports.
 - GUI-based interface for easy network design.
 - Supports both wired and wireless network simulations.
- **Use Cases:**
 - Simulating enterprise networks.
 - Testing new protocols and network configurations.
 - Analyzing network performance under different conditions.

2.3 OMNeT++

- **Purpose:** OMNeT++ is an open-source, modular, and extensible network simulation framework.
- **Features:**
 - Supports discrete event simulation.
 - Highly customizable and extensible.
 - Provides a graphical runtime environment for visualizing simulations.
 - Supports a wide range of network protocols and technologies.
- **Use Cases:**
 - Research and development of new network protocols.
 - Simulating large-scale networks.
 - Academic and industrial research.

2.4 NS2 (Network Simulator 2)

- **Purpose:** NS2 is an open-source network simulator widely used in academic research.
- **Features:**
 - Supports both wired and wireless network simulations.
 - Provides a scripting language (Tcl) for defining network scenarios.
 - Includes support for various protocols (TCP, UDP, etc.).
 - Generates trace files for detailed analysis.
- **Use Cases:**
 - Research in congestion control, routing protocols, and QoS.
 - Simulating ad hoc and sensor networks.

- Academic projects and research papers.

2.5 NS3 (Network Simulator 3)

- **Purpose:** NS3 is the successor to NS2 and is designed for more advanced network simulations.
 - **Features:**
 - Improved performance and scalability compared to NS2.
 - Supports real-time simulation and emulation.
 - Provides a more modern and modular architecture.
 - Supports a wide range of network protocols and technologies.
 - **Use Cases:**
 - Simulating large-scale and complex networks.
 - Research in next-generation networks (5G, IoT, etc.).
 - Testing and validating new network protocols.
-

3. Configuration of Network Simulation Tools

3.1 Cisco Packet Tracer

- **Steps to Configure:**
 1. Open Cisco Packet Tracer.
 2. Drag and drop devices (routers, switches, PCs) onto the workspace.
 3. Connect devices using appropriate cables (e.g., Ethernet, serial).
 4. Configure IP addresses, subnet masks, and routing protocols on devices.
 5. Use the simulation mode to test and visualize packet flow.

3.2 NetSim

- **Steps to Configure:**
 1. Launch NetSim and create a new project.
 2. Add network devices (routers, switches, end devices) to the workspace.
 3. Configure device settings (IP addresses, protocols, etc.).
 4. Run the simulation and analyze the results using the built-in tools.

3.3 OMNeT++

- **Steps to Configure:**
 1. Install OMNeT++ and the required modules.

2. Create a new network simulation project.
3. Define network topology using NED (Network Description) files.
4. Write simulation logic in C++.
5. Run the simulation and visualize results using the OMNeT++ GUI.

3.4 NS2

- **Steps to Configure:**
 1. Install NS2 on your system (Linux/Windows).
 2. Write a Tcl script to define the network topology and traffic.
 3. Run the script using the ns command.
 4. Analyze the trace files generated by the simulation.

3.5 NS3

- **Steps to Configure:**
 1. Install NS3 and its dependencies.
 2. Create a new simulation script in C++ or Python.
 3. Define the network topology, devices, and protocols.
 4. Run the simulation and analyze the output using NS3's visualization tools.

4. Comparison of Network Simulation Tools

Tool	Ease of Use	Scalability	Protocol Support	Use Case
Cisco Packet Tracer	High	Low	Cisco-specific	Education, small networks
NetSim	Medium	Medium	Wide range	Enterprise networks, research
OMNeT++	Medium	High	Wide range	Research, large-scale networks
NS2	Low	Medium	Wide range	Academic research, ad hoc networks
NS3	Medium	High	Wide range	Advanced research, 5G, IoT

5. Conclusion

Network simulation tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, and NS3 are essential for designing, testing, and analyzing network behavior. Each tool has its strengths and is suited for different use cases, ranging from educational purposes to advanced research. By understanding the features and configuration of these tools, network engineers and researchers can effectively simulate and analyze complex network scenarios.

Practical No: 3

Aim:

To learn handling and configuration of networking hardware like RJ-45 connector, CAT-6 cable, crimping tool, etc.

1. Introduction to Networking Hardware

Networking hardware forms the backbone of any network infrastructure. Proper handling and configuration of these components are essential for setting up reliable and efficient networks. In this practical, we will focus on the following networking hardware:

1. **RJ-45 Connector**
2. **CAT-6 Cable**
3. **Crimping Tool**
4. **Cable Tester**

These components are commonly used in Ethernet networks for connecting devices like computers, routers, switches, and other network-enabled devices.

2. Overview of Networking Hardware

2.1 RJ-45 Connector

- **Purpose:** The RJ-45 connector is a standardized physical interface used for connecting Ethernet cables to devices.
- **Features:**
 - 8-pin connector used for Ethernet networks.
 - Compatible with CAT-5, CAT-5e, CAT-6, and CAT-7 cables.
 - Used in both straight-through and crossover cables.
- **Use Cases:**
 - Connecting computers to switches or routers.
 - Building patch cables for network setups.



2.2 CAT-6 Cable

- **Purpose:** CAT-6 (Category 6) cable is a twisted pair cable used for Ethernet networks.
- **Features:**
 - Supports data transfer speeds up to 10 Gbps.
 - Backward compatible with CAT-5 and CAT-5e cables.
 - Reduced crosstalk and interference compared to older cable types.
- **Use Cases:**
 - High-speed Ethernet networks.
 - Data centers and enterprise networks.

2.3 Crimping Tool

- **Purpose:** A crimping tool is used to attach RJ-45 connectors to Ethernet cables.
- **Features:**
 - Designed to crimp RJ-45 connectors securely onto cables.
 - Often includes a wire cutter and stripper for preparing cables.
- **Use Cases:**
 - Creating custom-length Ethernet cables.
 - Repairing damaged Ethernet cables.



2.4 Cable Tester

- **Purpose:** A cable tester is used to verify the integrity and connectivity of Ethernet cables.
- **Features:**
 - Tests for continuity, shorts, and miswiring.
 - Provides visual or audible indications of cable faults.
- **Use Cases:**
 - Testing newly crimped cables.
 - Diagnosing network connectivity issues.

3. Steps to Handle and Configure Networking Hardware

3.1 Preparing the CAT-6 Cable

1. **Measure and Cut the Cable:**
 - Measure the required length of the CAT-6 cable and cut it using a cable cutter or scissors.
2. **Strip the Cable:**
 - Use the wire stripper on the crimping tool to remove about 1.5 inches of the outer jacket from the cable, exposing the twisted pairs inside.
3. **Untwist and Arrange the Wires:**
 - Untwist the pairs and arrange the wires according to the desired wiring standard (T568A or T568B). The most common standard is T568B, which arranges the wires in the following order:
 - Pin 1: Orange/White
 - Pin 2: Orange

- Pin 3: Green/White
- Pin 4: Blue
- Pin 5: Blue/White
- Pin 6: Green
- Pin 7: Brown/White
- Pin 8: Brown

4. Trim the Wires:

- Ensure the wires are even and trim them to about 0.5 inches in length using the wire cutter.

3.2 Attaching the RJ-45 Connector

1. Insert the Wires into the RJ-45 Connector:

- Carefully insert the arranged wires into the RJ-45 connector, ensuring each wire goes into its corresponding pin slot.

2. Crimp the Connector:

- Place the RJ-45 connector into the crimping tool and press firmly to crimp the connector onto the cable. Ensure the connector is securely attached.

3.3 Testing the Cable

1. Connect the Cable to a Cable Tester:

- Plug one end of the cable into the transmitter unit of the cable tester and the other end into the receiver unit.

2. Test the Cable:

- Turn on the cable tester and observe the results. The tester will indicate if the cable is properly wired and if there are any faults (e.g., shorts, miswiring).
-

4. Wiring Standards: T568A vs. T568B

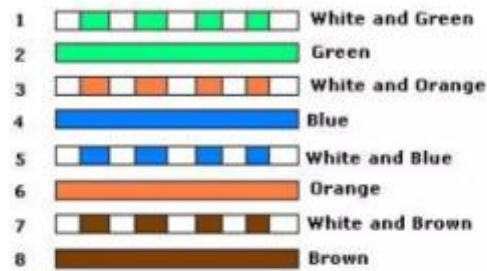
- **T568A:**

- Pin 1: Green/White
- Pin 2: Green
- Pin 3: Orange/White
- Pin 4: Blue
- Pin 5: Blue/White
- Pin 6: Orange
- Pin 7: Brown/White
- Pin 8: Brown

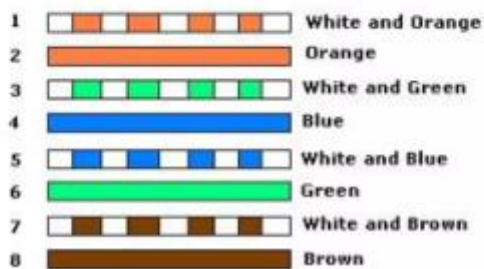
- **T568B:**

- Pin 1: Orange/White
- Pin 2: Orange
- Pin 3: Green/White
- Pin 4: Blue
- Pin 5: Blue/White
- Pin 6: Green
- Pin 7: Brown/White
- Pin 8: Brown

TIA/EIA 568A Wiring



TIA/EIA 568B Wiring



- **Straight-Through Cable:** Both ends use the same wiring standard (T568A or T568B). Used to connect different types of devices (e.g., computer to switch).
- **Crossover Cable:** One end uses T568A, and the other uses T568B. Used to connect similar devices (e.g., computer to computer).

5. Safety Precautions

- **Handle Tools Carefully:** Crimping tools and wire cutters are sharp and can cause injury if not handled properly.
- **Avoid Over-Crimping:** Over-crimping can damage the RJ-45 connector or the cable.
- **Inspect the Cable:** Before crimping, ensure the cable is not damaged or frayed.
- **Use a Cable Tester:** Always test the cable after crimping to ensure it is functioning correctly.

6. Conclusion

In this practical, we learned how to handle and configure networking hardware like RJ-45 connectors, CAT-6 cables, crimping tools, and cable testers. By following the steps outlined above, you can create custom Ethernet cables, test their integrity, and ensure proper connectivity in your network setup. This hands-on experience is essential for anyone working in network administration or IT support.

Practical No: 4

Aim:

To implement **Stop-and-Wait Protocol** and **Sliding Window Protocol** using C programming.

1. Introduction

Stop-and-Wait Protocol and **Sliding Window Protocol** are two fundamental flow control protocols used in data transmission.

- **Stop-and-Wait Protocol:** The sender transmits a frame and waits for an acknowledgment before sending the next frame.
 - **Sliding Window Protocol:** Allows multiple frames to be sent before waiting for an acknowledgment, improving efficiency.
-

2. Algorithm

Stop-and-Wait Protocol Algorithm:

1. Sender sends a frame.
2. Sender waits for an acknowledgment.
3. If acknowledgment is received, send the next frame.
4. Repeat until all frames are sent.

Sliding Window Protocol Algorithm:

1. Sender sends multiple frames within a window size.
 2. Receiver acknowledges frames.
 3. Sender slides the window and sends the next set of frames.
 4. Repeat until all frames are sent.
-

3. Implementation (C Code)

Stop-and-Wait Protocol (C Code)

```
#include <stdio.h>

void stopAndWait() {
    int frames, i;
    printf("Enter the number of frames to send: ");
    scanf("%d", &frames);

    for (i = 1; i <= frames; i++) {
        printf("Sending Frame %d...\n", i);
        printf("Waiting for Acknowledgment...\n");
        printf("Acknowledgment received for Frame %d\n", i);
    }
    printf("All frames sent successfully using Stop-and-Wait Protocol.\n");
}

int main() {
    stopAndWait();
    return 0;
}
```

```
}
```

Sliding Window Protocol (C Code)

```
#include <stdio.h>

#define WINDOW_SIZE 4 // Define the window size

void slidingWindow() {
    int frames, i;
    printf("Enter the total number of frames: ");
    scanf("%d", &frames);

    for (i = 1; i <= frames; i++) {
        printf("Sending Frame %d\n", i);

        if (i % WINDOW_SIZE == 0 || i == frames) {
            printf("Waiting for Acknowledgment...\n");
            printf("Acknowledgment received for Frames up to %d\n", i);
        }
    }
    printf("All frames sent successfully using Sliding Window Protocol.\n");
}

int main() {
    slidingWindow();
    return 0;
}
```

4. Output

Stop-and-Wait Protocol Output Example:

Output

```
Enter the number of frames to send: 3
Sending Frame 1...
Waiting for Acknowledgment...
Acknowledgment received for Frame 1
Sending Frame 2...
Waiting for Acknowledgment...
Acknowledgment received for Frame 2
Sending Frame 3...
Waiting for Acknowledgment...
Acknowledgment received for Frame 3
All frames sent successfully using Stop-and-Wait Protocol.
```


Sliding Window Protocol Output Example:

Output

Close

Enter the total number of frames: 6
Sending Frame 1
Sending Frame 2
Sending Frame 3
Sending Frame 4
Waiting for Acknowledgment...
Acknowledgment received for Frames up to 4
Sending Frame 5
Sending Frame 6
Waiting for Acknowledgment...
Acknowledgment received for Frames up to 6
All frames sent successfully using Sliding Window Protocol.

5. Result

- The **Stop-and-Wait Protocol** was successfully implemented and tested.
- The **Sliding Window Protocol** was successfully implemented with a window size of 4.

6. Conclusion

- The **Stop-and-Wait Protocol** ensures reliable communication but is inefficient for large data transfers due to waiting time.
- The **Sliding Window Protocol** improves efficiency by allowing multiple frames to be sent before waiting for acknowledgment.
- This experiment helps understand flow control mechanisms in networking.

Practical No. - 5: Simulating ARP/RARP Protocols

Aim:

To write a Java program to simulate the Address Resolution Protocol (ARP) using TCP.

Algorithm:

Client:

1. Start the program.
2. Establish a socket connection with the server.
3. Get the IP address to be converted into a MAC address.
4. Send the IP address to the server.
5. Receive and display the MAC address from the server.

Server:

1. Start the program.
 2. Accept the client's socket connection.
 3. Maintain a table mapping IP addresses to MAC addresses.
 4. Read the IP address sent by the client.
 5. Map the IP address to its MAC address and send it back to the client.
-

Program:

Client Code:

```
import java.io.*;
import java.net.*;

class Clientarp {
    public static void main(String args[]) {
        try {
            BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));
            Socket clsct = new Socket("127.0.0.1", 139);
            DataInputStream din = new DataInputStream(clsct.getInputStream());
            DataOutputStream dout = new DataOutputStream(clsct.getOutputStream());

            System.out.println("Enter the Logical address (IP):");
            String str1 = in.readLine();
            dout.writeBytes(str1 + '\n');

            String str = din.readLine();
            System.out.println("The Physical Address is: " + str);

            clsct.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Server Code:

```

import java.io.*;
import java.net.*;
import util.*;

class Serverarp {
    public static void main(String args[]) {
        try {
            ServerSocket serverSocket = new ServerSocket(139);
            System.out.println("Waiting for client connection...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected.");

            DataInputStream din = new
DataInputStream(clientSocket.getInputStream());
            DataOutputStream dout = new
DataOutputStream(clientSocket.getOutputStream());

            Hashtable<String, String> arpTable = new Hashtable<>();
            arpTable.put("192.168.1.1", "00-1A-2B-3C-4D-5E");
            arpTable.put("192.168.1.2", "00-1A-2B-3C-4D-5F");
            arpTable.put("192.168.1.3", "00-1A-2B-3C-4D-60");

            String ipAddress = din.readLine();
            System.out.println("Received IP address from client: " + ipAddress);

            String macAddress = arpTable.getOrDefault(ipAddress, "MAC address not
found");
            dout.writeBytes(macAddress + '\n');

            clientSocket.close();
            serverSocket.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}

```

Output:

Client Output:

Server Output:

Enter the Logical address (IP):

192.168.1.1

The Physical Address is: 00-1A-2B-3C-4D-5E

STDIN

Waiting for client connection...

Client connected.

Received IP address from client: 192.168.1.1 |

Conclusion:

This experiment successfully simulated the ARP protocol using Java and TCP socket programming. The client-server interaction demonstrated how an IP address is resolved to a MAC address, mimicking real-world ARP functionality. The server maintained a mapping table, and the client retrieved the corresponding MAC address for a given IP. This exercise provided a clear understanding of ARP and its role in network communication. Overall, the experiment was a valuable learning experience in network protocol simulation.

Practical No. 6: Write a Program Simulating PING and TRACEROUTE Commands.

ALGORITHM:

PING:

1. Start the program.
2. Import net and packages.
3. Get the IP address.
4. Ping the remote server using the Ping Command.
5. The packet statistics of the pinged server are displayed.

TRACEROUTE:

1. Start the program.
 2. Import net and packages.
 3. Get the IP address.
 4. Traceroute the remote server using the Traceroute Command.
 5. The number of max hops and byte packets are displayed.
-

Code:

PING:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class PingTest {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java PingTest <IP-Address/Domain>");
            return;
        }

        String ipAddress = args[0];
        String os = System.getProperty("os.name").toLowerCase();
        String pingCmd = os.contains("win") ? "ping -n 4 " + ipAddress : "ping -c 4 " + ipAddress;

        try {
            Process p = Runtime.getRuntime().exec(pingCmd);
            BufferedReader in = new BufferedReader(new
            InputStreamReader(p.getInputStream()));

            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                System.out.println(inputLine);
            }
            in.close();
        } catch (IOException e) {
            System.out.println("Error executing ping command: " + e.getMessage());
        }
    }
}
```

TRACEROUTE:

```
6. import java.io.BufferedReader;
```

```

7. import java.io.IOException;
8. import java.io.InputStreamReader;
9.
10. public class PingTest {
11.     public static void main(String[] args) {
12.         if (args.length == 0) {
13.             System.out.println("Usage: java PingTest <IP-Address/Domain>");
14.             return;
15.         }
16.
17.         String ipAddress = args[0];
18.         String os = System.getProperty("os.name").toLowerCase();
19.         String pingCmd = os.contains("win") ? "ping -n 4 " + ipAddress :
"ping -c 4 " + ipAddress;
20.
21.         try {
22.             Process p = Runtime.getRuntime().exec(pingCmd);
23.             BufferedReader in = new BufferedReader(new
InputStreamReader(p.getInputStream()));
24.
25.             String inputLine;
26.             while ((inputLine = in.readLine()) != null) {
27.                 System.out.println(inputLine);
28.             }
29.             in.close();
30.         } catch (IOException e) {
31.             System.out.println("Error executing ping command: " +
e.getMessage());
32.         }
33.     }
34. }

```

Output:

PING:

```

diksha@LAPTOP-1N0BF8F8:~$ vi pingTest.java
diksha@LAPTOP-1N0BF8F8:~$ javac pingTest.java
diksha@LAPTOP-1N0BF8F8:~$ java pingTest www.google.com
PING www.google.com(bom07s31-in-x04.1e100.net (2404:6800:4009:824::2004)) 56 data bytes
64 bytes from bom07s31-in-x04.1e100.net (2404:6800:4009:824::2004): icmp_seq=1 ttl=118 time=31.4 ms
64 bytes from bom07s31-in-x04.1e100.net (2404:6800:4009:824::2004): icmp_seq=2 ttl=118 time=39.0 ms
64 bytes from bom07s31-in-x04.1e100.net (2404:6800:4009:824::2004): icmp_seq=3 ttl=118 time=31.9 ms
^C
--- www.google.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 31.363/34.093/38.978/3.462 ms
diksha@LAPTOP-1N0BF8F8:~$

```

TRACEROUTE:

```
diksha@LAPTOP-1NOBF8F8:~$ vi traceroute.java
diksha@LAPTOP-1NOBF8F8:~$ javac traceroute.java
diksha@LAPTOP-1NOBF8F8:~$ java traceroute
traceroute to www.google.co.in (172.217.166.67), 30 hops max, 60 byte packets
 1 * * *
 2 * * *
 3 * * *
 4 * * *
 5 * * *
 6 * * *
 7 * * *
 8 * * *
 9 * * *
10 * * *
11 * * *
12 * * *
13 * * *
14 * * *
15 * * *
16 * * *
17 * * *
18 * * *
19 * * *
20 * * *
21 * * *
22 * * *
23 * * *
24 * * *
25 * * *
26 * * *
27 * * *
28 * * *
29 * * *
30 * * *
diksha@LAPTOP-1NOBF8F8:~$
```

Conclusion:

This experiment demonstrated how to simulate **Ping** and **Traceroute** commands using Java. The programs executed system commands to ping a target IP address and trace the route to a destination, respectively. The output displayed the round-trip time for ping and the hops for traceroute, providing insights into network connectivity and routing. This exercise enhanced understanding of network diagnostics and Java's ability to interact with system commands.

Practical No. 7:

Running and using services/commands like ping, trace route, nslookup, arp, telnet, ftp, etc.

Theory:

Running and Using Service Commands

1. Ping

Ping is a basic Internet program that allows a user to verify that a particular IP address exists and can accept requests. Ping is used diagnostically to ensure that a host computer the user is trying to reach is actually operating. Ping works by sending an Internet Control Message Protocol (ICMP) Echo Request to a specified interface on the network and waiting for a reply.

Ping can be used for troubleshooting to test connectivity and determine response time. It helps in identifying packet loss, network latency, and connectivity issues between the host and the target device.

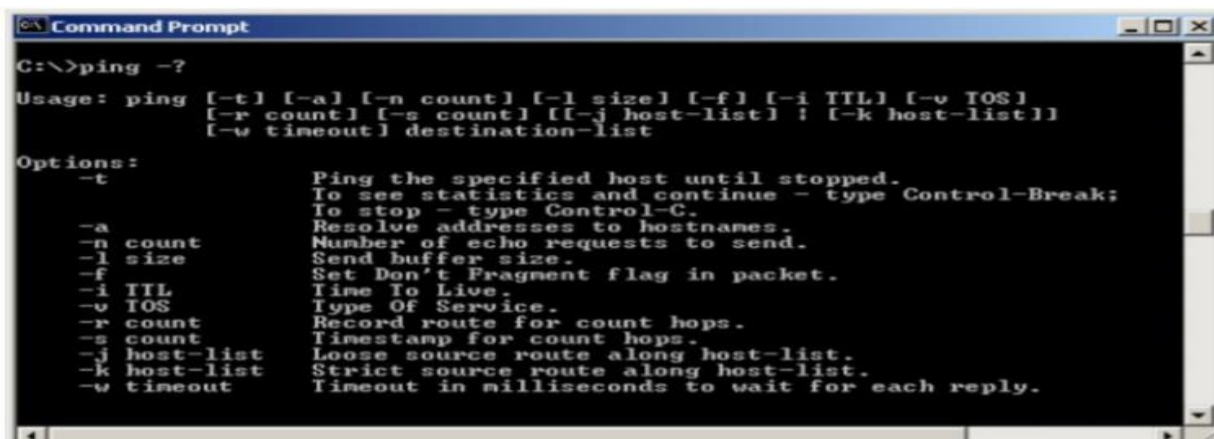
Ping Command Syntax:

```
ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS] [-r count] [-s count] [-w timeout] [-R] [-S srcaddr] [-p] [-4] [-6] target [/?]
```

Explanation of Common Ping Options:

- **-t** : Pings the specified host until stopped manually (Ctrl + C).
- **-a** : Resolves addresses to hostnames.
- **-n count** : Specifies the number of echo requests to send.
- **-l size** : Sends buffer size data.
- **-f** : Prevents packets from being fragmented.
- **-i TTL** : Sets the Time to Live (TTL) value.
- **-w timeout** : Specifies the timeout in milliseconds to wait for each reply.
- **-4** : Forces the use of IPv4.
- **-6** : Forces the use of IPv6.

Ping is a valuable tool for network administrators to test the reachability of network devices and diagnose potential issues with network connectivity.



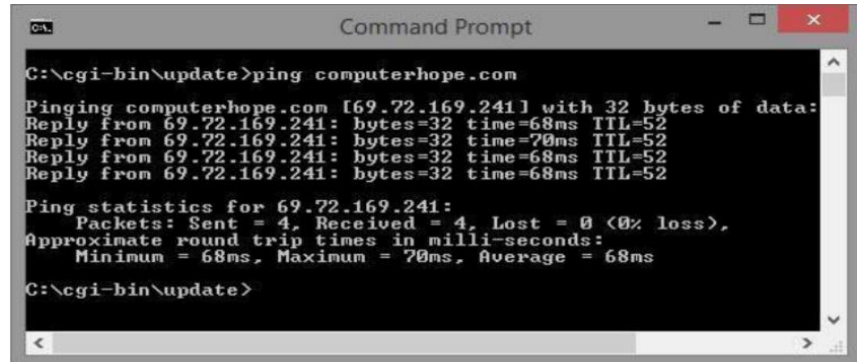
```
Command Prompt
C:\>ping -?
Usage: ping [-t] [-a] [-n count] [-l size] [-f] [-i TTL] [-v TOS]
           [-r count] [-s count] [[-j host-list] : [-k host-list]]
           [-w timeout] destination-list

Options:
  -t           Ping the specified host until stopped.
               To see statistics and continue - type Control-Break;
               To stop - type Control-C.
  -a           Resolve addresses to hostnames.
  -n count     Number of echo requests to send.
  -l size      Send buffer size.
  -f           Set Don't Fragment flag in packet.
  -i TTL       Time To Live.
  -v TOS       Type Of Service.
  -r count     Record route for count hops.
  -s count     Timestamp for count hops.
  -j host-list Loose source route along host-list.
  -k host-list Strict source route along host-list.
  -w timeout   Timeout in milliseconds to wait for each reply.
```

Example :

ping computerhope.com

ping 192.168.2.1



```
C:\cgi-bin\update>ping computerhope.com

Pinging computerhope.com [69.72.169.241] with 32 bytes of data:
Reply from 69.72.169.241: bytes=32 time=68ms TTL=52
Reply from 69.72.169.241: bytes=32 time=70ms TTL=52
Reply from 69.72.169.241: bytes=32 time=68ms TTL=52
Reply from 69.72.169.241: bytes=32 time=68ms TTL=52

Ping statistics for 69.72.169.241:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 68ms, Maximum = 70ms, Average = 68ms

C:\cgi-bin\update>
```

2. Traceroute

A traceroute is a function that traces the path from one network to another. It allows us to diagnose the source of many problems. The tracert command is a Command Prompt command that's used to show several details about the path that a packet takes from the computer or device you're on to whatever destination you specify.

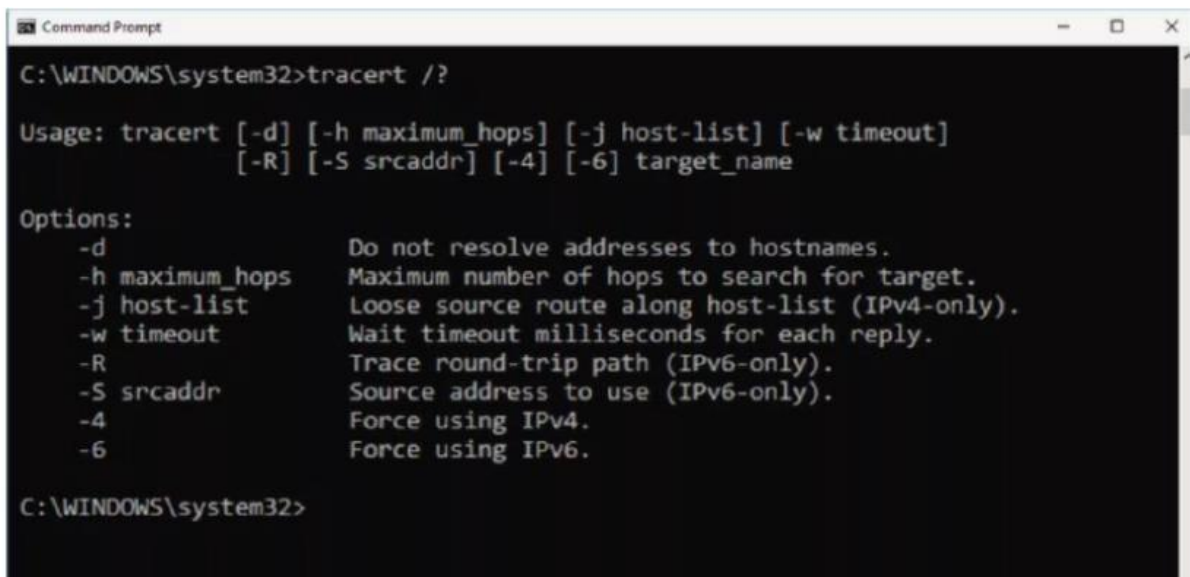
Tracert Command Syntax:

```
tracert [-d] [-h MaxHops] [-w TimeOut] [-4] [-6] target [/?]
```

Explanation of Common Traceroute Options:

- **-d** : Prevents the resolution of IP addresses to hostnames for faster results.
- **-h MaxHops** : Specifies the maximum number of hops (default is 30).
- **-w TimeOut** : Sets the time in milliseconds to wait for a response.
- **-4** : Forces the use of IPv4.
- **-6** : Forces the use of IPv6.

Traceroute is an essential tool for diagnosing network connectivity issues, identifying routing problems, and measuring network latency between nodes.



```
C:\WINDOWS\system32>tracert /?

Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
              [-R] [-S srcaddr] [-4] [-6] target_name

Options:
    -d                Do not resolve addresses to hostnames.
    -h maximum_hops   Maximum number of hops to search for target.
    -j host-list       Loose source route along host-list (IPv4-only).
    -w timeout         Wait timeout milliseconds for each reply.
    -R                Trace round-trip path (IPv6-only).
    -S srcaddr         Source address to use (IPv6-only).
    -4                Force using IPv4.
    -6                Force using IPv6.

C:\WINDOWS\system32>
```

Example:

tracert 192.168.1.1

tracert www.google.com

3. Nslookup

The nslookup (which stands for name server lookup) command is a network utility program used to obtain information about internet servers. It finds name server information for domains by querying the Domain Name System (DNS). The nslookup command sends a domain name query packet to a designated (or default) domain name system (DNS) server.

Depending on the system you are using, the default may be the local DNS name server at your service provider, some intermediate name server, or the root server system for the entire domain name system hierarchy.

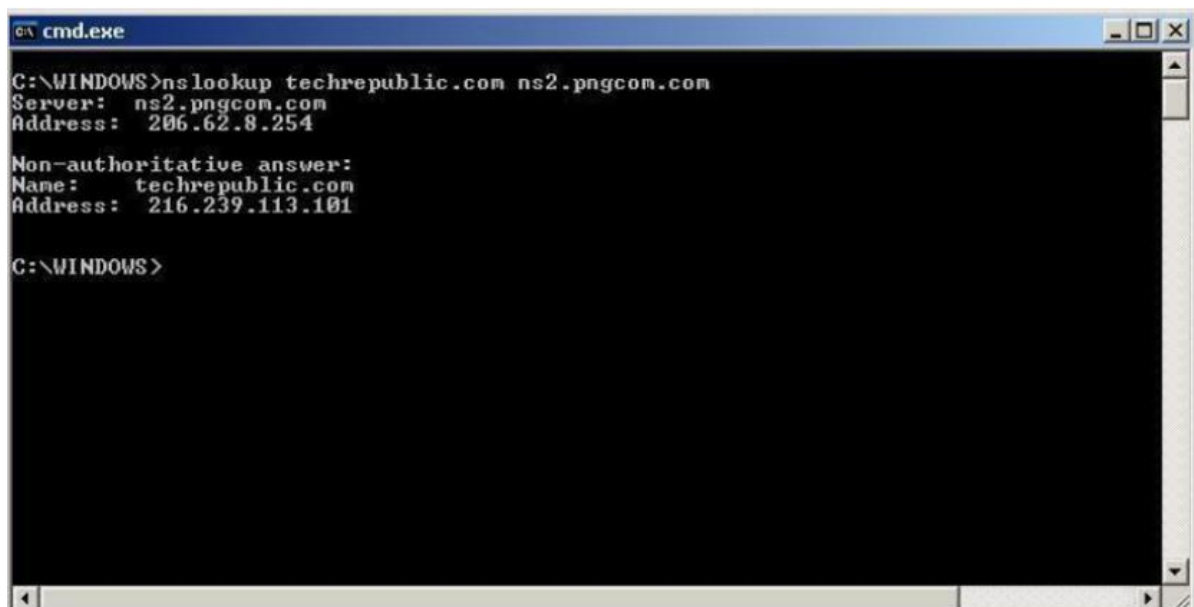
Nslookup Command Syntax:

```
nslookup [-SubCommand ...] [{ComputerToFind} [-Server]]
```

Explanation of Common Nslookup Options:

- **ComputerToFind** : Specifies the domain name or IP address to query.
- **-Server** : Specifies the DNS server to query instead of the default.
- **-SubCommand** : Various subcommands can be used for specific queries, such as retrieving mail exchanger records or checking reverse DNS lookups.

The nslookup command is useful for troubleshooting DNS resolution issues, verifying domain name records, and identifying potential misconfigurations in network settings.



```
cmd.exe
C:\WINDOWS>nslookup techrepublic.com ns2.pngcom.com
Server: ns2.pngcom.com
Address: 206.62.8.254

Non-authoritative answer:
Name: techrepublic.com
Address: 216.239.113.101

C:\WINDOWS>
```

4. ARP (Address Resolution Protocol)

ARP is used with IP for mapping a 32-bit Internet Protocol address to a MAC address that is recognized in the local network as specified in RFC 826. Once recognized, the server or networking device returns a response containing the required address.

ARP Command Syntax:

ARP -s inet_addr eth_addr
[if_addr]

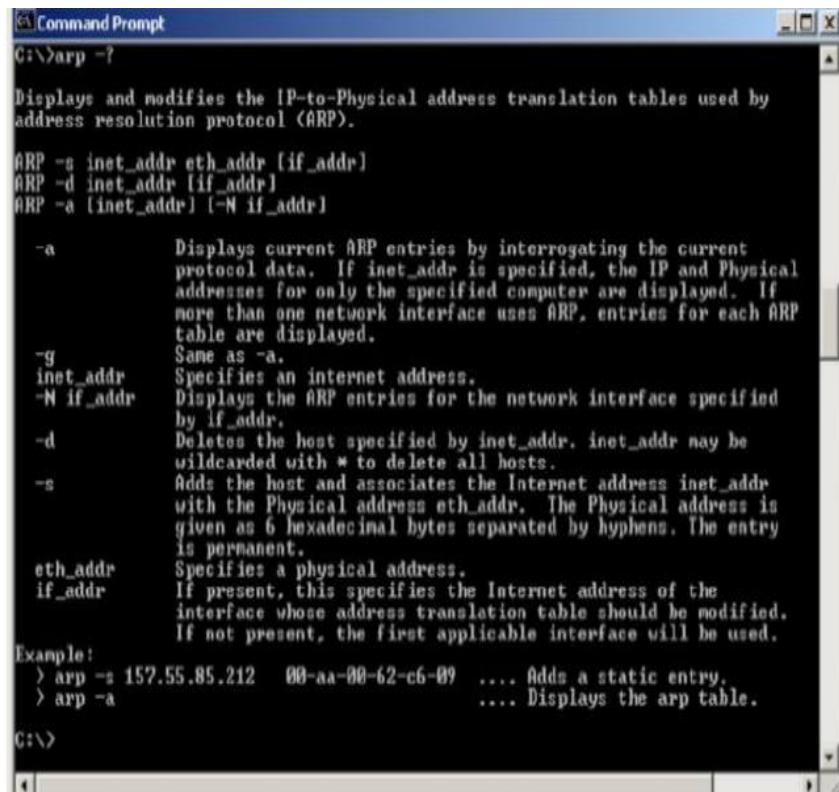
ARP -d inet_addr [if_addr]

ARP -a [inet_addr] [-N
if_addr]

Example:

arp -a

arp -s 220.0.0.161 00-50-
04-62-F7-23



```
G:\>arp -?

Displays and modifies the IP-to-Physical address translation tables used by
address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr]

-a          Displays current ARP entries by interrogating the current
            protocol data. If inet_addr is specified, the IP and Physical
            addresses for only the specified computer are displayed. If
            more than one network interface uses ARP, entries for each ARP
            table are displayed.
-g          Same as -a.
            Specifies an internet address.
-N if_addr  Displays the ARP entries for the network interface specified
            by if_addr.
-d          Deletes the host specified by inet_addr. inet_addr may be
            wildcarded with * to delete all hosts.
-s          Adds the host and associates the Internet address inet_addr
            with the Physical address eth_addr. The Physical address is
            given as 6 hexadecimal bytes separated by hyphens. The entry
            is permanent.
eth_addr    Specifies a physical address.
if_addr     If present, this specifies the Internet address of the
            interface whose address translation table should be modified.
            If not present, the first applicable interface will be used.

Example:
> arp -s 157.55.85.212 00-aa-00-62-c6-09 .... Adds a static entry.
> arp -a                                     .... Displays the arp table.

G:\>
```

5. TelNet

Telnet is a user command and an underlying TCP/IP protocol for accessing remote computers. Through Telnet, an administrator or another user can access someone else's computer remotely. On the Web, HTTP and FTP protocols allow you to request specific files from remote computers, but not to actually be logged on as a user of that computer. With Telnet, you log on as a regular user with whatever privileges you may have been granted to the specific application and data on that computer.

Telnet Command Syntax:

telnet [/a] [/e <EscapeChar>
[/f <FileName>] [/l
<UserName>] [/t {vt100 |
vt52 | ansi | vtnt}] [<Host>
[<Port>]] [/?]



```
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

H:\>Telnet 192.168.1.1 /?

telnet [-a[-e escape char][-f log file][-l user][-t term][host [port]]
-a          Attempt automatic logon. Same as -l option except uses
            the currently logged on user's name.
-e          Escape character to enter telnet client prompt.
-f          File name for client side logging
-l          Specifies the user name to log in with on the remote system.
            Requires that the remote system support the TELNET ENVIRON option.
-t          Specifies terminal type.
            Supported term types are vt100, vt52, ansi and vtnt only.
host        Specifies the hostname or IP address of the remote computer
            to connect to.
port        Specifies a port number or service name.

H:\>telnet 192.168.1.1 -f C:\ROUTER_CFG.TXT
```

FTP (File Transfer Protocol)

File Transfer Protocol (FTP) is
the commonly used protocol

for exchanging files over the Internet. FTP uses the Internet's TCP/IP protocols to enable data transfer. FTP uses a client-server architecture, often secured with SSL/TLS.

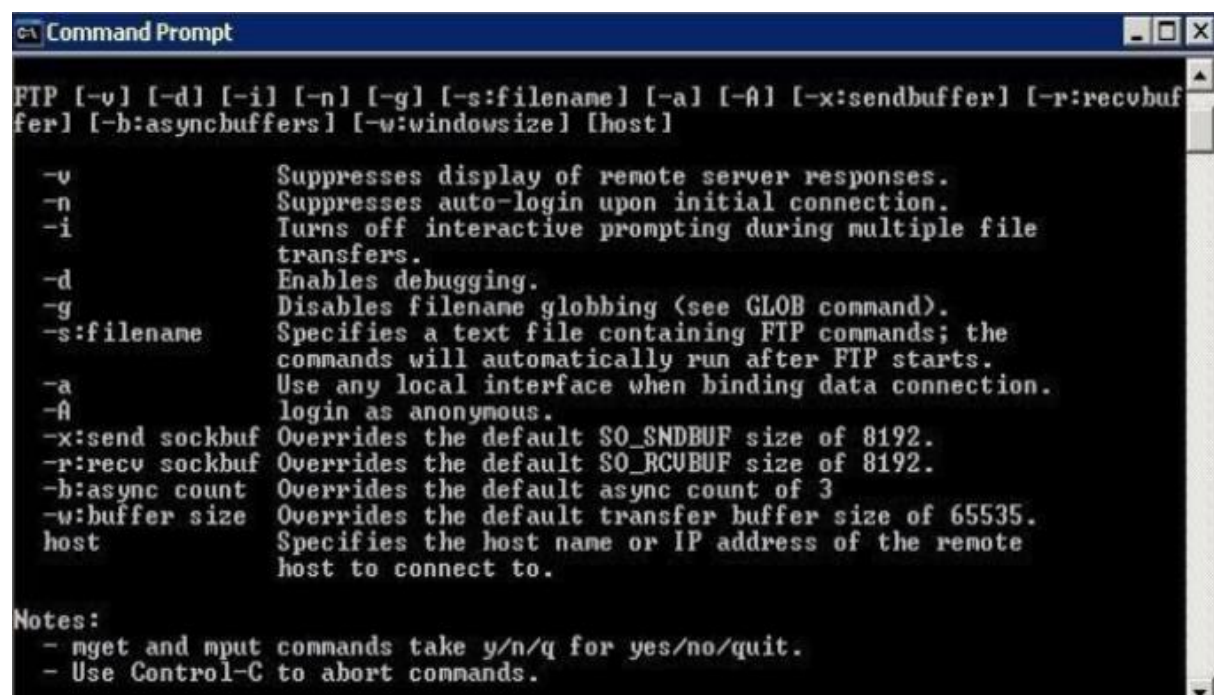
FTP Command Syntax: FTP [-options] [-s:filename] [-w:buffer] [host]

Connect using FTP: To connect to another computer using FTP at the MS-DOS prompt, command line, or Linux shell, type FTP and press Enter.

Example: open ftp.example.com

Commands to run at the FTP prompt:

1. **append local-file [remote-file]:** Append a local file to a file on the remote computer.
2. **ascii:** Set the file transfer type to ASCII, the default. In ASCII text mode, character-set and end-of-line characters are converted as necessary.
3. **bell:** Toggle a bell to ring after each command. By default, the bell is off.
4. **binary:** Set the file transfer type to binary. Use Binary for transferring executable program files or binary data files e.g. Oracle.
5. **bye:** End the FTP session and exit FTP.
6. **cd:** Change the working directory on the remote host.
7. **close:** End the FTP session and return to the cmd prompt.
8. **debug:** Toggle debugging. When debug is on, FTP will display every command.
9. **delete remote-file:** Delete file on remote host.
10. **dir [remote-directory] [local-file]:** List a remote directory's files and subdirectories (or save the listing to local-file).



```
CA Command Prompt
FTP [-v] [-d] [-i] [-n] [-g] [-s:filename] [-a] [-A] [-x:sendbuffer] [-r:recvbuf
fer] [-b:asyncbuffers] [-w:window size] [host]

-v          Suppresses display of remote server responses.
-n          Suppresses auto-login upon initial connection.
-i          Turns off interactive prompting during multiple file
transfers.
-d          Enables debugging.
-g          Disables filename globbing (see GLOB command).
-s:filename Specifies a text file containing FTP commands; the
commands will automatically run after FTP starts.
-a          Use any local interface when binding data connection.
-A          login as anonymous.
-x:send sockbuf Overrides the default SO_SNDBUF size of 8192.
-r:recv sockbuf Overrides the default SO_RCVBUF size of 8192.
-b:async count Overrides the default async count of 3
-w:buffer size Overrides the default transfer buffer size of 65535.
host        Specifies the host name or IP address of the remote
host to connect to.

Notes:
- mget and mput commands take y/n/q for yes/no/quit.
- Use Control-C to abort commands.
```

Practical No. 8: To study and analyze the performance of different routing algorithms in selecting the most efficient and economical network path during data transfer. The focus will be on **Link State Routing, Flooding, and Distance Vector Routing**.

Requirements:

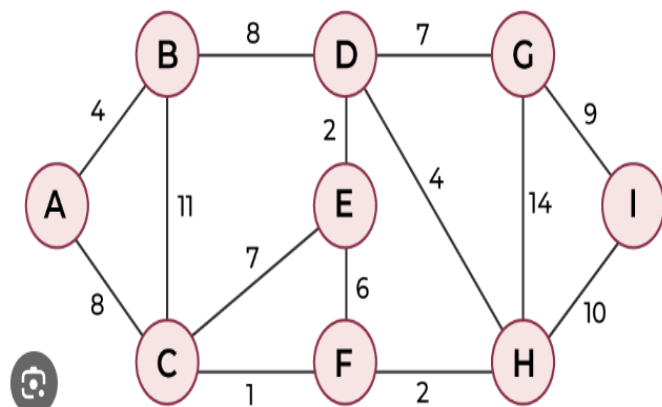
- Basic understanding of network routing protocols
- Simulation tools like NS-2, GNS3, or any network simulator
- Knowledge of network topology design
- Java programming knowledge for implementation

Theory:

Routing algorithms are essential for determining the best path for data packets to travel from the source to the destination in a network. The efficiency of these algorithms depends on factors such as **path cost, network congestion, scalability, and convergence time**. In this case study, we will compare three routing algorithms:

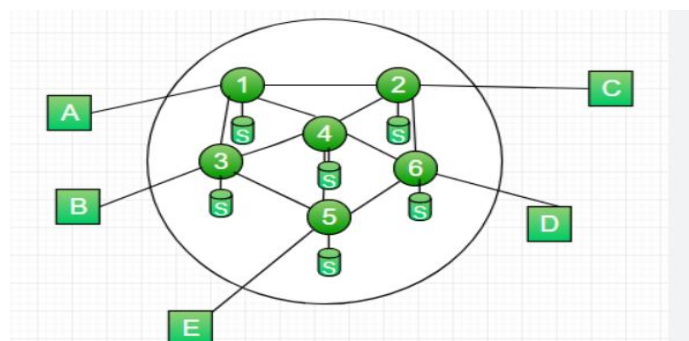
1. Link State Routing:

- Each router maintains a map of the entire network topology.
- Uses Dijkstra's algorithm to calculate the shortest path to all destinations.
- Advantages: Fast convergence, efficient use of network resources.
- Disadvantages: High memory and processing requirements.



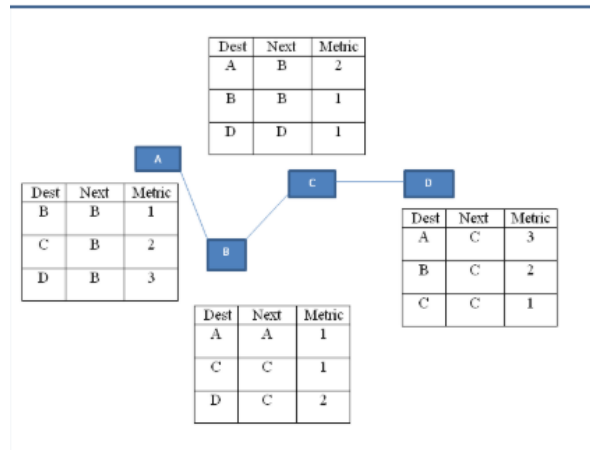
2. Flooding:

- A simple routing technique where every incoming packet is sent out on every outgoing line except the one it arrived on.
- Advantages: No need for complex routing tables, robust in case of node failures.
- Disadvantages: Inefficient use of bandwidth, can cause network congestion.



3. Distance Vector Routing:

- Each router shares its routing table with its neighbors periodically.
- Uses the Bellman-Ford algorithm to calculate the shortest path.
- Advantages: Simple to implement, low memory usage.
- Disadvantages: Slow convergence, prone to routing loops.



Network Topology:

A simple network with **5 nodes** connected in a mesh topology. The nodes are interconnected with varying link costs to simulate real-world scenarios.

Copy

(Node 1) ---- (Node 2) ---- (Node 3)

| | |

(Node 4) ---- (Node 5) ---- (Node 6)

Simulation Setup:

- Link State Routing Simulation:**
 - Each node will maintain a complete map of the network.
 - Dijkstra's algorithm will be used to compute the shortest path.
- Flooding Simulation:**
 - Every packet will be broadcast to all nodes except the source.
 - Packet duplication and network congestion will be observed.
- Distance Vector Routing Simulation:**
 - Each node will share its routing table with neighbors periodically.
 - The Bellman-Ford algorithm will be used to update the routing tables.

Java Implementation:

Below is a basic Java implementation of **Link State Routing** using Dijkstra's algorithm and **Distance Vector Routing** using the Bellman-Ford algorithm.

1. Link State Routing (Dijkstra's Algorithm):

```
import java.util.*;
class Node {
    int id;
    Map<Node, Integer> neighbors; // Neighbor nodes and link costs

    Node(int id) {
```

```

        this.id = id;
        this.neighbors = new HashMap<>();
    }

    void addNeighbor(Node node, int cost) {
        neighbors.put(node, cost);
    }
}

public class LinkStateRouting {
    public static void dijkstra(Node start) {
        Map<Node, Integer> distances = new HashMap<>();
        PriorityQueue<Node> queue = new
PriorityQueue<>(Comparator.comparingInt(distances::get));
        distances.put(start, 0);
        queue.add(start);

        while (!queue.isEmpty()) {
            Node current = queue.poll();
            for (Map.Entry<Node, Integer> neighbor : current.neighbors.entrySet())
            {
                Node next = neighbor.getKey();
                int newDistance = distances.get(current) + neighbor.getValue();
                if (newDistance < distances.getDefault(next, Integer.MAX_VALUE))
                {
                    distances.put(next, newDistance);
                    queue.add(next);
                }
            }
        }

        // Print shortest distances
        System.out.println("Shortest distances from Node " + start.id + ":");
        for (Map.Entry<Node, Integer> entry : distances.entrySet()) {
            System.out.println("Node " + entry.getKey().id + " -> " +
entry.getValue());
        }
    }

    public static void main(String[] args) {
        Node n1 = new Node(1);
        Node n2 = new Node(2);
        Node n3 = new Node(3);

        n1.addNeighbor(n2, 2);
        n2.addNeighbor(n3, 3);
        n1.addNeighbor(n3, 10);

        dijkstra(n1); // Run Dijkstra's algorithm from Node 1
    }
}

```

2. Distance Vector Routing (Bellman-Ford Algorithm):

```

import java.util.*;
class Node {
    int id;
    Map<Node, Integer> neighbors; // Neighbor nodes and link costs

    Node(int id) {
        this.id = id;
        this.neighbors = new HashMap<>();
    }
}

```

```

        void addNeighbor(Node node, int cost) {
            neighbors.put(node, cost);
        }
    }

    public class DistanceVectorRouting {
        public static void bellmanFord(Node start, List<Node> nodes) {
            Map<Node, Integer> distances = new HashMap<>();
            for (Node node : nodes) {
                distances.put(node, Integer.MAX_VALUE);
            }
            distances.put(start, 0);

            for (int i = 0; i < nodes.size() - 1; i++) {
                for (Node node : nodes) {
                    for (Map.Entry<Node, Integer> neighbor : node.neighbors.entrySet())
                    {
                        Node next = neighbor.getKey();
                        int newDistance = distances.get(node) + neighbor.getValue();
                        if (newDistance < distances.get(next)) {
                            distances.put(next, newDistance);
                        }
                    }
                }
            }

            // Print shortest distances
            System.out.println("Shortest distances from Node " + start.id + ":");
            for (Map.Entry<Node, Integer> entry : distances.entrySet()) {
                System.out.println("Node " + entry.getKey().id + " -> " +
entry.getValue());
            }
        }

        public static void main(String[] args) {
            Node n1 = new Node(1);
            Node n2 = new Node(2);
            Node n3 = new Node(3);

            n1.addNeighbor(n2, 2);
            n2.addNeighbor(n3, 3);
            n1.addNeighbor(n3, 10);

            List<Node> nodes = Arrays.asList(n1, n2, n3);
            bellmanFord(n1, nodes); // Run Bellman-Ford algorithm from Node 1
        }
    }

```

Execution Steps:

1. **Install Network Simulator** (e.g., NS-2 or GNS3).
2. **Design the Network Topology** using the simulator.
3. **Configure Routing Protocols:**
 - Set up Link State, Flooding, and Distance Vector routing on the nodes.
4. **Run the Simulation:**
 - Simulate data transfer between nodes and observe the path selection.

5. Analyze Results:

- Measure metrics such as **packet delivery ratio**, **end-to-end delay**, and **network overhead**.
 - Compare the performance of the three routing algorithms.
-

Observations:

- **Link State Routing:**
 - Provides the most efficient path with minimal delay.
 - High memory and processing overhead due to maintaining the entire network map.
 - **Flooding:**
 - Simple to implement but causes significant network congestion.
 - High packet duplication leads to inefficient bandwidth usage.
 - **Distance Vector Routing:**
 - Easy to implement but suffers from slow convergence and routing loops.
 - Suitable for smaller networks with fewer nodes.
-

Conclusion:

This case study highlights the strengths and weaknesses of different routing algorithms in selecting the optimal and economical network path. **Link State Routing** is the most efficient for large networks with dynamic topologies, while **Distance Vector Routing** is suitable for smaller networks. **Flooding**, although simple, is inefficient for most practical applications due to its high bandwidth consumption. The choice of routing algorithm depends on the network size, topology, and specific requirements of the application.

Practical No. 9

Study of Socket Programming and Client-Server mode using TCP and UDP

AIM:

To implement socket programming for date and time display from server to client using both TCP and UDP sockets, and to understand the differences between connection-oriented (TCP) and connectionless (UDP) communication.

ALGORITHM:

TCP Implementation:

Server:

1. Create a ServerSocket object and bind it to a specific port.
2. Listen for client connections using accept() method.
3. When connection is established, get output stream to send data.
4. Send server's current date and time to client.
5. Get input stream to receive client's IP address.
6. Display client details.
7. Close the connection.
8. Repeat steps 2-7 for multiple clients.
9. Close the ServerSocket.

Client:

1. Create a Socket object and connect to server's port.
 2. Get input stream to receive server's date and time.
 3. Display the received date and time.
 4. Get output stream to send client's IP address.
 5. Send client's IP address to server.
 6. Close all streams and socket.
-

UDP Implementation:

Server:

1. Create a DatagramSocket object bound to a specific port.
2. Create buffer to receive incoming packets.
3. Receive packet from client using receive() method.
4. Get client address and port from received packet.
5. Prepare current date and time as response.
6. Send response packet back to client.
7. Repeat from step 2.

Client:

1. Create a DatagramSocket object.
2. Prepare request message (can be empty).
3. Create packet with server address and port.
4. Send packet to server.

5. Create buffer to receive response.
6. Receive response packet from server.
7. Display the date and time from server.
8. Close the socket.

PROGRAM:

TCPDateClient.java

```
import java.net.*;
import java.io.*;
public class TCPDateClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 6789);
            BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            // Get and display date from server
            String date = in.readLine();
            System.out.println("Date and time from server: " + date);

            // Send client IP to server
            out.println(InetAddress.getLocalHost().getHostAddress());

            socket.close();
        } catch (Exception e) {
            System.out.println("Client error: " + e);
        }
    }
}
```

TCPDateServer.java

```
import java.net.*;
import java.io.*;
import java.util.Date;
public class TCPDateServer {
    public static void main(String[] args) {
        try {
            ServerSocket server = new ServerSocket(6789);
            System.out.println("TCP Date Server running...");

            while (true) {
                Socket client = server.accept();
                PrintWriter out = new PrintWriter(client.getOutputStream(), true);
                BufferedReader in = new BufferedReader(
                    new InputStreamReader(client.getInputStream()));

                // Send date to client
                out.println(new Date().toString());

                // Get and display client info
                String clientIP = in.readLine();
                System.out.println("Client connected from: " + clientIP);

                client.close();
            }
        } catch (Exception e) {
            System.out.println("Server error: " + e);
        }
    }
}
```

```

    }
}

UDPDateClient.java
import java.net.*;
import java.io.*;

public class UDPDateClient {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket();
            InetAddress serverAddr = InetAddress.getByName("localhost");

            // Send empty packet to server
            byte[] sendData = new byte[1024];
            DatagramPacket sendPacket = new DatagramPacket(
                sendData, sendData.length, serverAddr, 9876);
            socket.send(sendPacket);

            // Receive response
            byte[] receiveData = new byte[1024];
            DatagramPacket receivePacket = new DatagramPacket(
                receiveData, receiveData.length);
            socket.receive(receivePacket);

            String date = new String(receivePacket.getData());
            System.out.println("Date from UDP server: " + date.trim());

            socket.close();
        } catch (Exception e) {
            System.out.println("UDP Client error: " + e);
        }
    }
}

```

```

UDPDateServer.java
import java.net.*;
import java.util.Date;

public class UDPDateServer {
    public static void main(String[] args) {
        try {
            DatagramSocket socket = new DatagramSocket(9876);
            System.out.println("UDP Date Server running...");

            while (true) {
                byte[] receiveData = new byte[1024];
                DatagramPacket receivePacket = new DatagramPacket(
                    receiveData, receiveData.length);
                socket.receive(receivePacket);

                // Get client info
                InetAddress clientAddr = receivePacket.getAddress();
                int port = receivePacket.getPort();

                // Prepare response
                String date = new Date().toString();
                byte[] sendData = date.getBytes();

                // Send response
                DatagramPacket sendPacket = new DatagramPacket(
                    sendData, sendData.length, clientAddr, port);
                socket.send(sendPacket);
            }
        } catch (Exception e) {
            System.out.println("UDP Server error: " + e);
        }
    }
}

```

```
}  
}  
}
```

OUTPUT:

TCP Implementation:

Server Side:

```
TCP Date Server running...  
Client connected from: 192.168.1.5  
Client connected from: 192.168.1.7
```

Client Side:

```
Date and time from server: Wed Oct 18 14:35:22 IST 2023
```

UDP Implementation:

Server Side:

```
UDP Date Server running...
```

Client Side:

```
Date from UDP server: Wed Oct 18 14:36:05 IST 2023
```

RESULT:

The programs for date and time display using both TCP and UDP sockets were implemented successfully. The TCP version establishes a connection before data transfer, while UDP sends datagrams without connection establishment. Both methods correctly displayed the server's current date and time on the client side.

Practical No. 10

Objective:

Configuration of router, hub, switch etc. (using real devices or simulators).

Brief Theory:-

1. Repeater: Functioning at Physical Layer. A repeater is an electronic device that receives a signal and retransmits it at a higher level and/or higher power, or onto the other side of an obstruction, so that the signal can cover longer distances. Repeater have two ports ,so cannot be use to connect for more than two devices

2. Hub: An Ethernet hub, active hub, network hub, repeater hub, hub or concentrator is a device for connecting multiple twisted pair or fiber optic Ethernet devices together and making them act as a single network segment. Hubs work at the physical layer (layer 1) of the OSI model. The device is a form of multiport repeater. Repeater hubs also participate in collision detection, forwarding a jam signal to all ports if it detects a collision.

3. Switch: A network switch or switching hub is a computer networking device that connects network segments. The term commonly refers to a network bridge that processes and routes data at the data link layer (layer 2) of the OSI model. Switches that additionally process data at the network layer (layer 3 and above) are often referred to as Layer 3 switches or multilayer switches.

4. Bridge: A network bridge connects multiple network segments at the data link layer (Layer 2) of the OSI model. In Ethernet networks, the term bridge formally means a device that behaves according to the IEEE

802.1D standard. A bridge and switch are very much alike; a switch being a bridge with numerous ports.

Switch or Layer 2 switch is often used interchangeably with bridge. Bridges can analyze incoming data packets

to determine if the bridge is able to send the given packet to another segment of the network.

5. Router: A router is an electronic device that interconnects two or more computer networks, and

selectively interchanges packets of data between them. Each data packet contains address information that a router can use to determine if the source and destination are on the same network, or if the data packet must be transferred from one network to another. Where multiple routers are used in a large collection of interconnected networks, the routers exchange information about target system addresses, so that each router can build up a table showing the preferred paths between any two systems on the interconnected networks.

6. Gate Way: In a communications network, a network node equipped for interfacing with another network that uses different protocols. A gateway may contain devices such as protocol translators, impedance matching devices, rate converters, fault isolators, or signal translators as necessary to provide system interoperability. It also requires the establishment of mutually acceptable administrative procedures between both networks.

A protocol translation/mapping gateway interconnects networks with different network protocol technologies by performing the required protocol conversions.

Result-

Configuration of router, hub, switch is performed .

Practical No: 11

Implementation of Subnetting

AIM:

To understand and implement subnetting for a given IP address, calculate subnet masks, identify network and host portions, and determine valid subnets and host ranges.

THEORY:

Subnetting divides a single large network into smaller, manageable subnetworks (subnets). It improves:

- **Network performance** (reduces congestion)
- **Security** (isolates subnet traffic)
- **IP address utilization** (reduces wastage)

Key Concepts:

1. **IP Address (IPv4):** 32-bit address (e.g., 192.168.1.0).
 2. **Subnet Mask:** Differentiates network and host portions (e.g., 255.255.255.0).
 3. **CIDR Notation:** Compact representation (e.g., /24 for 255.255.255.0).
 4. **Network ID:** Identifies the subnet.
 5. **Broadcast Address:** Sends data to all hosts in the subnet.
-

ALGORITHM:

1. **Input:**
 - Given IP address (e.g., 192.168.1.0).
 - Required number of subnets or hosts per subnet.
 2. **Steps:**
 - **Determine default subnet mask** (based on IP class: A, B, or C).
 - **Calculate borrowed bits** for subnetting:
 - $\text{Subnets} = 2^n$ (where n = borrowed bits).
 - $\text{Hosts per subnet} = 2^{(32 - \text{CIDR})} - 2$.
 - **Derive new subnet mask** (CIDR notation).
 - **List subnets and host ranges.**
 3. **Output:**
 - Subnet mask, network IDs, valid host ranges, and broadcast addresses.
-

EXAMPLE CALCULATION:

Given:

- IP: 192.168.1.0/24 (Class C).
- Required: **4 subnets**.

Solution:

1. **Borrowed bits:** $2^n \geq 4 \rightarrow n = 2$.
2. **New subnet mask:** /26 (255.255.255.192).
3. **Subnet increment:** $256 - 192 = 64$.

Subnet	Network ID	Host Range	Broadcast
1	192.168.1.0	192.168.1.1 – .62	192.168.1.63
2	192.168.1.64	192.168.1.65 – .126	192.168.1.127
3	192.168.1.128	192.168.1.129 – .190	192.168.1.191
4	192.168.1.192	192.168.1.193 – .254	192.168.1.255

PROGRAM

```
import java.util.Scanner;

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

        System.out.println("SUBNETTING CALCULATOR");
        System.out.println("Enter IP Address (e.g., 192.168.1.0): ");
        String ipAddress = sc.nextLine();

        System.out.println("Enter CIDR Notation (e.g., 24): ");
        int cidr = sc.nextInt();

        System.out.println("Enter number of required subnets: ");
        int subnetsRequired = sc.nextInt();

        calculateSubnets(ipAddress, cidr, subnetsRequired);

        sc.close();
    }

    public static void calculateSubnets(String ipAddress, int cidr, int
subnetsRequired) {
        // Calculate borrowed bits
        int borrowedBits = (int) Math.ceil(Math.log(subnetsRequired) /
Math.log(2));
        int newCidr = cidr + borrowedBits;

        if (newCidr > 30) {
            System.out.println("Error: Too many subnets requested for the given
CIDR");
            return;
        }

        // Calculate subnet mask
        String subnetMask = calculateSubnetMask(newCidr);

        // Calculate subnet increment
        int increment = (int) Math.pow(2, 32 - newCidr);

        System.out.println("\nSUBNETTING DETAILS:");
        System.out.println("Original IP: " + ipAddress + "/" + cidr);
        System.out.println("Subnets needed: " + subnetsRequired);
    }
}
```



```

        System.out.println("Borrowed bits: " + borrowedBits);
        System.out.println("New CIDR: /" + newCidr);
        System.out.println("Subnet Mask: " + subnetMask);
        System.out.println("Subnet Increment: " + increment);
        System.out.println("\nSUBNET TABLE:");

        // Parse the IP address
        String[] octets = ipAddress.split("\\.");
        int[] ip = new int[4];
        for (int i = 0; i < 4; i++) {
            ip[i] = Integer.parseInt(octets[i]);
        }

        // Print subnet details
        for (int i = 0; i < subnetsRequired; i++) {
            System.out.println("\nSubnet " + (i + 1));
            System.out.println("Network ID: " + ip[0] + "." + ip[1] + "." + ip[2] +
                "." + ip[3]);

            // Calculate and print host range
            int firstHost = ip[3] + 1;
            int lastHost = ip[3] + increment - 2;
            System.out.println("Host Range: " + ip[0] + "." + ip[1] + "." + ip[2] +
                "." + firstHost +
                " - " + ip[0] + "." + ip[1] + "." + ip[2] + "." +
                lastHost);

            // Calculate and print broadcast address
            int broadcast = ip[3] + increment - 1;
            System.out.println("Broadcast: " + ip[0] + "." + ip[1] + "." + ip[2] +
                "." + broadcast);

            // Update IP for next subnet
            ip[3] += increment;
            if (ip[3] >= 256) {
                ip[2] += ip[3] / 256;
                ip[3] = ip[3] % 256;
                if (ip[2] >= 256) {
                    ip[1] += ip[2] / 256;
                    ip[2] = ip[2] % 256;
                }
            }
        }
    }

    public static String calculateSubnetMask(int cidr) {
        int mask = 0xFFFFFFFF << (32 - cidr);
        return ((mask >> 24) & 0xFF) + "." +
            ((mask >> 16) & 0xFF) + "." +
            ((mask >> 8) & 0xFF) + "." +
            (mask & 0xFF);
    }
}

```

Output

```
SUBNETTING CALCULATOR
Enter IP Address (e.g., 192.168.1.0):
192.168.1.0
Enter CIDR Notation (e.g., 24):
24
Enter number of required subnets:
4

SUBNETTING DETAILS:
Original IP: 192.168.1.0/24
Subnets needed: 4
Borrowed bits: 2
New CIDR: /26
Subnet Mask: 255.255.255.192
Subnet Increment: 64

SUBNET TABLE:

Subnet 1
Network ID: 192.168.1.0
Host Range: 192.168.1.1 - 192.168.1.62
Broadcast: 192.168.1.63

Subnet 2
Network ID: 192.168.1.64
Host Range: 192.168.1.65 - 192.168.1.126
Broadcast: 192.168.1.127

Subnet 3
Network ID: 192.168.1.128
Host Range: 192.168.1.129 - 192.168.1.190
Broadcast: 192.168.1.191

Subnet 4
Network ID: 192.168.1.192
Host Range: 192.168.1.193 - 192.168.1.254
Broadcast: 192.168.1.255
```

RESULT:

The Java program successfully implemented subnetting for the given IP address 192.168.1.0/24, dividing it into 4 subnets with correct network IDs, host ranges, and broadcast addresses. The subnet mask was calculated as 255.255.255.192 with an increment of 64 between subnets.

Practical No: 12

Applications using TCP Sockets - Echo, Chat and File Transfer

AIM:

To implement Java programs demonstrating TCP socket communication for:

- a) Echo client-server
 - b) Chat application
 - c) File transfer system
-

1. Echo Client-Server

Algorithm:

Server:

1. Create ServerSocket on port 9999
2. Accept client connection
3. Create I/O streams
4. Echo back received messages until "bye"

Client:

1. Connect to server
2. Send messages and display echoed responses

Program:

EchoServer.java

```
import java.io.*;
import java.net.*;

public class EchoServer {
    public EchoServer(int portnum) {
        try {
            server = new ServerSocket(portnum);
        }
        catch (Exception err) {
            System.out.println(err);
        }
    }

    public void serve() {
        try {
            while (true) {
                Socket client = server.accept();
                BufferedReader r = new BufferedReader(
                    new InputStreamReader(client.getInputStream()));
                PrintWriter w = new PrintWriter(
                    client.getOutputStream(), true);

                w.println("Welcome to the Java EchoServer. Type 'bye' to close.");

                String line;
                do {
                    line = r.readLine();
                    if (line != null) {
                        w.println("Got: " + line);
                        System.out.println("Client says: " + line);
                    }
                } while (line != null);
            }
        }
    }
}
```

```

        }
        } while (!line.trim().equals("bye"));

        client.close();
    }
}
catch (Exception err) {
    System.err.println(err);
}
}

public static void main(String[] args) {
    EchoServer s = new EchoServer(9999);
    s.serve();
}

private ServerSocket server;
}

```

EchoClient.java

```

import java.io.*;
import java.net.*;

public class EchoClient {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("127.0.0.1", 9999);
            BufferedReader r = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            PrintWriter w = new PrintWriter(
                s.getOutputStream(), true);
            BufferedReader con = new BufferedReader(
                new InputStreamReader(System.in));

            String line;
            do {
                // Read server response
                line = r.readLine();
                if (line != null)
                    System.out.println(line);

                // Get user input
                line = con.readLine();
                w.println(line);
            } while (!line.trim().equals("bye"));

            s.close();
        }
        catch (Exception err) {
            System.err.println(err);
        }
    }
}

```

OUTPUT:

Client says: Hello
 Client says: This is TCP Socket
 Server Side: Client says: bye

Client Side:

```
Welcome to the Java EchoServer. Type 'bye' to close.  
Got: Hello  
Got: This is TCP Socket  
Got: bye
```

2. Chat Application

Algorithm:

Server:

1. Create ServerSocket on port 8888
2. Maintain list of connected clients
3. Broadcast messages to all clients

Client:

1. Connect to server
2. Start separate threads for sending/receiving
3. Display messages from other clients

Program:

ChatServer.java

```
import java.net.*;  
import java.io.*;  
import java.util.*;  
  
public class ChatServer {  
    private static Set<PrintWriter> clientWriters = new HashSet<>();  
  
    public static void main(String[] args) throws Exception {  
        ServerSocket server = new ServerSocket(8888);  
        System.out.println("Chat Server running...");  
  
        while (true) {  
            Socket client = server.accept();  
            new ClientHandler(client).start();  
        }  
    }  
  
    private static class ClientHandler extends Thread {  
        private Socket socket;  
        private PrintWriter out;  
  
        public ClientHandler(Socket socket) {  
            this.socket = socket;  
        }  
  
        public void run() {  
            try {  
                out = new PrintWriter(socket.getOutputStream(), true);  
                synchronized(clientWriters) {  
                    clientWriters.add(out);  
                }  
  
                BufferedReader in = new BufferedReader(  
                    new InputStreamReader(socket.getInputStream()));  
  
                String message;  
                while ((message = in.readLine()) != null) {  
                    System.out.println("Received: " + message);  
                }  
            }  
        }  
    }  
}
```

```

        broadcast(message);
    }
} catch (IOException e) {
    System.out.println(e);
} finally {
    try { socket.close(); } catch (IOException e) {}
}
}

private void broadcast(String message) {
    synchronized(clientWriters) {
        for (PrintWriter writer : clientWriters) {
            writer.println(message);
        }
    }
}
}
}

```

ChatClient.java

```

import java.net.*;
import java.io.*;
import javax.swing.*;

public class ChatClient {
    public static void main(String[] args) throws Exception {
        String serverIP = JOptionPane.showInputDialog("Enter server IP:");
        Socket socket = new Socket(serverIP, 8888);

        // Message receiver thread
        new Thread(() -> {
            try (BufferedReader in = new BufferedReader(
                new InputStreamReader(socket.getInputStream()))) {
                String message;
                while ((message = in.readLine()) != null) {
                    System.out.println("Other: " + message);
                }
            } catch (IOException e) {}
        }).start();

        // Message sender
        try (PrintWriter out = new PrintWriter(
            socket.getOutputStream(), true);
            BufferedReader console = new BufferedReader(
                new InputStreamReader(System.in))) {
            String userInput;
            while ((userInput = console.readLine()) != null) {
                out.println(userInput);
            }
        }
    }
}

```

Output:

```

[Client1] Hi → [Server] Received: Hi → [Client2] Other: Hi
[Client2] Hello → [Server] Received: Hello → [Client1] Other: Hello

```

3. File Transfer

Algorithm:

Server:

1. Create ServerSocket on port 7777
2. Accept connection and create file output stream
3. Save received file

Client:

1. Connect to server
2. Read file and send bytes

Program:

FileServer.java

```
import java.net.*;
import java.io.*;

public class FileServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = new ServerSocket(7777);
        System.out.println("File Server ready...");

        Socket socket = server.accept();
        InputStream in = socket.getInputStream();

        // Save as "received_file.txt"
        FileOutputStream out = new FileOutputStream("received_file.txt");

        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = in.read(buffer)) != -1) {
            out.write(buffer, 0, bytesRead);
        }

        out.close();
        socket.close();
        System.out.println("File received successfully!");
    }
}
```

FileClient.java

```
import java.net.*;
import java.io.*;

public class FileClient {
    public static void main(String[] args) throws Exception {
        Socket socket = new Socket("localhost", 7777);
        OutputStream out = socket.getOutputStream();

        // Send "sample.txt" file
        FileInputStream fileIn = new FileInputStream("sample.txt");

        byte[] buffer = new byte[1024];
```

```
        int bytesRead;  
        while ((bytesRead = fileIn.read(buffer)) != -1) {  
            out.write(buffer, 0, bytesRead);  
        }  
  
        fileIn.close();  
        socket.close();  
        System.out.println("File sent successfully!");  
    }  
}
```

Output:

```
[Server] File received successfully!  
[Client] File sent successfully!
```

RESULT:

Successfully implemented three TCP socket applications:

1. **Echo Server:** Basic message echoing
2. **Chat Application:** Multi-client real-time messaging
3. **File Transfer:** Secure file transmission