

## INDEX

<b>S.NO</b>	<b>TOPIC</b>	<b>PAGE No.</b>
1	Implementation on stop and wait protocol and sliding window protocol.	1-2
2	Study the socket and client-server model.	3-4
3	Write a code simulating ARP/RARP protocols.	5-6
4	Write a code simulating PING and TRACEROUTE commands.	7-8
5	Create a socket for HTTP for web page upload and download.	8-10
6	Write a program to implement RPC(remote procedure call)	10-12
7	Implementation of subnetting.	13
8	Application using TCP sockets like A.Echo client and Echo server b. Chat c. File Transfer	13-17
9	Applications using TCP and UDP sockets like d, DNS e. SNMP f. file transfer	18- 21
10	Study of network simulator(NS). And simulation of congestion control algorithms using NS	22
11	Perform a case study about different routing algorithms	23-24
12	To learn handling and configuration of networking hardware.	25-26
13	Configuration of router ,hub,switch etc.	27-28
14	Running and using services/command like ping ,tracert etc.	29-30
15	Network packet analysis using tools like wireshark ,tcpdump,etc.	31-32
16	Network simulation using tools like cisco packet tracer,NS2 etc.	33-34
17	Socket programming using UDP and TCP.	35

# 1. Implementation of stop and wait protocol and sliding window protocol.

## Stop and wait protocol:

```
import java.util.Scanner;

public class StopAndWaitProtocol {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int frames;
        System.out.print("Enter number of frames to send: ");
        frames = sc.nextInt();
        for (int i = 1; i <= frames; i++) {
            System.out.println("\nSending Frame " + i);
            System.out.println("Waiting for ACK...");
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
            System.out.println("ACK received for Frame " + i);
        }
        System.out.println("\nAll frames sent successfully.");
    }
}
```

## Sliding window protocol:

```
import java.util.Scanner;

public class SlidingWindowProtocol {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int frames, windowSize;
        System.out.print("Enter number of frames: ");
        frames = sc.nextInt();
        System.out.print("Enter window size: ");
        windowSize = sc.nextInt();
        int sent = 0;
        while (sent < frames) {
            System.out.println("\nSending frames:");
            for (int i = sent + 1;
                i <= sent + windowSize && i <= frames;
                i++) {
                System.out.println("Frame " + i + " sent");
            }
            System.out.println("Waiting for ACK...");
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
            for (int i = sent + 1;
                i <= sent + windowSize && i <= frames;
                i++) {
                System.out.println("ACK received for Frame " + i);
            }
        }
    }
}
```

```
sent += windowSize;
    }
    System.out.println("\nAll frames transmitted successfully.");
}
}
```

### **Output:**

#### **for stop and wait protocol:**

```
Enter number of frames to send: 2

Sending Frame 1
Waiting for ACK...
ACK received for Frame 1

Sending Frame 2
Waiting for ACK...
ACK received for Frame 2

All frames sent successfully.

=== Code Execution Successful ===
```

#### **For sliding window protocol:**

```
Enter number of frames: 3
Enter window size: 2

Sending frames:
Frame 1 sent
Frame 2 sent
Waiting for ACK...
ACK received for Frame 1
ACK received for Frame 2

Sending frames:
Frame 3 sent
Waiting for ACK...
ACK received for Frame 3

All frames transmitted successfully.

=== Code Execution Successful ===
```

## 2. Study of socket programming and client -server model.

### Server code:

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

public class Server {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(5000);
            System.out.println("Server is waiting for client...");
            Socket s = ss.accept();
            System.out.println("Client connected");
            DataInputStream dis =
                new DataInputStream(s.getInputStream());
            String str = dis.readUTF();
            System.out.println("Message from client: " + str);
            DataOutputStream dos =
                new DataOutputStream(s.getOutputStream());
            dos.writeUTF("Message received by server");
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

### Client code:

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

public class Client {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 5000);
            DataOutputStream dos =
                new DataOutputStream(s.getOutputStream());
            dos.writeUTF("Hello Server");
            DataInputStream dis =
                new DataInputStream(s.getInputStream());
            String str = dis.readUTF();
            System.out.println("Message from server: " + str);
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

## **Output**

### **Server console:**

```
Server is waiting for client...  
Client connected  
Message from client: Hello Server|
```

### **Client console:**

```
Message from server: Message received by server|
```

### 3. Write a code simulating ARP/RARP protocols.

```
import java.util.Scanner;

public class ARPRARP {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] ip = {
            "192.168.1.1",
            "192.168.1.2",
            "192.168.1.3"
        };
        String[] mac = {
            "AA:BB:CC:11",
            "AA:BB:CC:22",
            "AA:BB:CC:33"
        };
        System.out.println("1. ARP Protocol");
        System.out.println("2. RARP Protocol");

        System.out.print("Enter your choice: ");
        int choice = sc.nextInt();
        sc.nextLine();
        switch (choice) {
            case 1:
                System.out.print("Enter IP Address: ");
                String inputIP = sc.nextLine();
                boolean ipFound = false;
                for (int i = 0; i < ip.length; i++) {
                    if (inputIP.equals(ip[i])) {
                        System.out.println("MAC Address: " + mac[i]);
                        ipFound = true;
                        break;
                    }
                }
                if (!ipFound) {
                    System.out.println("IP Address not found");
                }
                break;
            case 2:
                System.out.print("Enter MAC Address: ");
                String inputMAC = sc.nextLine();
                boolean macFound = false;
                for (int i = 0; i < mac.length; i++) {
                    if (inputMAC.equals(mac[i])) {
                        System.out.println("IP Address: " + ip[i]);
                        macFound = true;
                        break;
                    }
                }
                if (!macFound) {
                    System.out.println("MAC Address not found");
                }
                break;
            default:
                System.out.println("Invalid Choice");
        }
    }
}
```

```
}  
}  
}
```

## **Output:**

```
1. ARP Protocol  
2. RARP Protocol  
Enter your choice: 2  
Enter MAC Address: AA:BB:CC:11  
IP Address: 192.168.1.1  
  
=== Code Execution Successful ===
```

```
1. ARP Protocol  
2. RARP Protocol  
Enter your choice: 1  
Enter IP Address: 192.168.1.1  
MAC Address: AA:BB:CC:11  
  
=== Code Execution Successful ===
```

#### 4. Write a code simulating PING and TRACEROTE commands.

##### a. Simulating PING command:

```
import java.util.Scanner;
public class PingSimulation {
    public static void main(String[] args)
    {
        Scanner sc = new
Scanner(System.in);
        System.out.print("Enter IP
Address or Hostname: ");
        String host = sc.nextLine();
        System.out.println("Pinging " +
host + " with 32 bytes of data:");
        for (int i = 1; i <= 4; i++) {
            System.out.println("Reply from
" + host +
                ": bytes=32 time=" + (10 +
i) + "ms TTL=64");
            try {
                Thread.sleep(1000);
            } catch (Exception e) {
            }
        }
        System.out.println("\nPing
completed successfully.");
    }
}
```

##### b. Simulating TRACEROTE

```
import java.util.Scanner;
public class TracerouteSimulation {
    public static void main(String[] args)
    {
        Scanner sc = new
Scanner(System.in);
        System.out.print("Enter
Destination Host: ");
        String host = sc.nextLine();
        System.out.println("Tracing route
to " + host + "... \n");
        String[] routers = {
            "192.168.1.1",
            "172.16.0.1",
            "10.0.0.1",
            host
        };
        for (int i = 0; i < routers.length;
i++) {
            System.out.println((i + 1) + "
" +
                routers[i] + " " +
                (20 + i * 5) + "ms");
            try {

```

```
        Thread.sleep(1000);
    } catch (Exception e) {
    }
}
System.out.println("\nTrace
complete.");
}
}
```

### **Output:**

#### **PING Simulation:**

```
Enter IP Address or Hostname: google.com
Pinging google.com with 32 bytes of data:
Reply from google.com: bytes=32 time=11ms TTL=64
Reply from google.com: bytes=32 time=12ms TTL=64
Reply from google.com: bytes=32 time=13ms TTL=64
Reply from google.com: bytes=32 time=14ms TTL=64

Ping completed successfully.

=== Code Execution Successful ===
```

#### **TRACEROUTE Simulation:**

```
Enter Destination Host: google.com
Tracing route to google.com...

 1  192.168.1.1    20ms
 2  172.16.0.1    25ms
 3  10.0.0.1     30ms
 4  google.com    35ms

Trace complete.

=== Code Execution Successful ===
```

## 5. Create a socket for HTTP for web page upload and download.

### Server code:

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

public class HTTPServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(8080);
            System.out.println("Server started at port 8080");
            Socket socket = serverSocket.accept();
            System.out.println("Client connected");
            BufferedReader br = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String data = br.readLine();
            System.out.println("Received Web Page Content:");
            System.out.println(data);
            PrintWriter pw = new PrintWriter(
                socket.getOutputStream(), true);
            pw.println("Web page uploaded successfully");
            socket.close();
            serverSocket.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

### Client code:

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

public class HTTPClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 8080);
            PrintWriter pw = new PrintWriter(
                socket.getOutputStream(), true);
            pw.println("<html><body><h1>Hello World</h1></body></html>");
            BufferedReader br = new BufferedReader(
                new InputStreamReader(socket.getInputStream()));
            String response = br.readLine();
            System.out.println("Server Response: " + response);
            socket.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

## **Output:**

### **Server console:**

```
Server started at port 8080  
Client connected  
Received Web Page Content:  
<html><body><h1>Hello World</h1></body></html>
```

### **Client Console:**

```
Server Response: Web page uploaded successfully  
|  
=== Code Execution Successful ===
```

## 6. Write a program to implement RPC(remote procedure call).

### Server Side:

```
import java.io.*;
import java.net.*;
public class RPCServer {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(5000);
            System.out.println("RPC Server is running...");
            Socket s = ss.accept();
            DataInputStream dis =
                new DataInputStream(s.getInputStream());
            int a = dis.readInt();
            int b = dis.readInt();
            int result = a + b;
            DataOutputStream dos =
                new DataOutputStream(s.getOutputStream());
            dos.writeInt(result);
            System.out.println("Addition performed remotely");
            s.close();
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

### Client Side:

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

public class RPCClient {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            Socket s = new Socket("localhost", 5000);
            DataOutputStream dos =
                new DataOutputStream(s.getOutputStream());
            System.out.print("Enter first number: ");
            int a = sc.nextInt();
            System.out.print("Enter second number: ");
            int b = sc.nextInt();
            dos.writeInt(a);
            dos.writeInt(b);
            DataInputStream dis =
                new DataInputStream(s.getInputStream());
            int result = dis.readInt();
            System.out.println("Result from RPC Server: " + result);
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

```
}  
}
```

### **Output:**

#### **Server console:**

```
RPC Server is running...  
Addition performed remotely
```

#### **Client console:**

```
Enter first number: 10  
Enter second number: 20  
Result from RPC Server: 30|  
  
=== Code Execution Successful ===
```

## 7. Implementation of Subnetting.

```
import java.util.Scanner;

public class Subnetting {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter IP Address: ");
        String ip = sc.nextLine();
        System.out.print("Enter number of subnets: ");
        int subnets = sc.nextInt();
        int bits = 0;
        int temp = subnets;
        while (temp > 1) {
            bits++;
            temp = temp / 2;
        }
        int newMask = 24 + bits;
        int totalHosts = (int)Math.pow(2, (32 - newMask)) - 2;
        System.out.println("\nIP Address: " + ip);
        System.out.println("Number of Subnets: " + subnets);
        System.out.println("New Subnet Mask: /" + newMask);
        System.out.println("Hosts per Subnet: " + totalHosts);
    }
}
```

### Output:

```
Enter IP Address: 192.168.1.0
Enter number of subnets: 4

IP Address: 192.168.1.0
Number of Subnets: 4
New Subnet Mask: /26
Hosts per Subnet: 62

=== Code Execution Successful ===
```

## 8. Application using TCP sockets like

### a. Echo client and echo server b. Chat c. file transfer

#### A. Echo Client and Echo Server:

##### Echo Server Code:

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

public class EchoServer {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(5000);
            System.out.println("Echo Server Started");
            Socket s = ss.accept();
            BufferedReader br = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            PrintWriter pw = new PrintWriter(
                s.getOutputStream(), true);
            String msg = br.readLine();
            System.out.println("Client Message: " + msg);
            pw.println(msg);
            s.close();
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

##### }echo client code:

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

public class EchoClient {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            Socket s = new Socket("localhost", 5000);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            PrintWriter pw = new PrintWriter(
                s.getOutputStream(), true);
            System.out.print("Enter Message: ");
            String msg = sc.nextLine();
            pw.println(msg);
            String response = br.readLine();
            System.out.println("Server Reply: " + response);
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Chat server:

### Chat Server code:

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

public class ChatServer {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(6000);
            System.out.println("Chat Server Started");
            Socket s = ss.accept();
            BufferedReader br = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            PrintWriter pw = new PrintWriter(
                s.getOutputStream(), true);
            String msg = br.readLine();
            System.out.println("Client: " + msg);
            pw.println("Hello Client");
            s.close();
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

### Chat Client code:

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

public class ChatClient {
    public static void main(String[] args) {
        try {
            Scanner sc = new Scanner(System.in);
            Socket s = new Socket("localhost", 6000);
            BufferedReader br = new BufferedReader(
                new InputStreamReader(s.getInputStream()));
            PrintWriter pw = new PrintWriter(
                s.getOutputStream(), true);
            System.out.print("Enter Message: ");
            String msg = sc.nextLine();
            pw.println(msg);
            String reply = br.readLine();
            System.out.println("Server: " + reply);
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

## B. File Transfer:

**File transfer server code:**

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

public class FileServer {
    public static void main(String[] args) {
        try {
            ServerSocket ss = new ServerSocket(7000);
            System.out.println("File Server Started");
            Socket s = ss.accept();
            FileInputStream fis =
                new FileInputStream("test.txt");
            OutputStream os = s.getOutputStream();
            int ch;
            while ((ch = fis.read()) != -1) {
                os.write(ch);
            }
            System.out.println("File Sent");
            fis.close();
            s.close();
            ss.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

**File Transfer client code:**

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

public class FileClient {
    public static void main(String[] args) {
        try {
            Socket s = new Socket("localhost", 7000);
            InputStream is = s.getInputStream();
            FileOutputStream fos =
                new FileOutputStream("received.txt");
            int ch;
            while ((ch = is.read()) != -1) {
                fos.write(ch);
            }
            System.out.println("File Received");
            fos.close();
            s.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

## Output:

### Echo Server:

```
Enter Message: Hello  
Server Reply: Hello  
=== Code Execution Successful ===
```

### Chat application:

```
Enter Message: Hi Server  
Server: Hello Client  
=== Code Execution Successful ===
```

### File transfer:

```
File Received  
=== Code Execution Successful ===
```

## 9. Application using TCP and UDP socket like d. DNS e. SNMP f. File Transfer.

### D. DNS resolver using UDP:

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

public class DNSApplication {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("1. Server");
        System.out.println("2. Client");
        System.out.print("Enter choice: ");
        int choice = sc.nextInt();
        sc.nextLine();
        try {
            if (choice == 1) {
                DatagramSocket ds = new DatagramSocket(5000);
                byte[] receive = new byte[1024];
                DatagramPacket dp =
                    new DatagramPacket(receive, receive.length);
                System.out.println("DNS Server Started");
                ds.receive(dp);
                String domain =
                    new String(dp.getData(), 0, dp.getLength());
                String ip = "142.250.183.14";
                byte[] send = ip.getBytes();
                DatagramPacket reply =
                    new DatagramPacket(
                        send,
                        send.length,
                        dp.getAddress(),
                        dp.getPort());
                ds.send(reply);
                ds.close();
            }
            else if (choice == 2) {
                DatagramSocket ds = new DatagramSocket();
                System.out.print("Enter Domain Name: ");
                String domain = sc.nextLine();
                byte[] send = domain.getBytes();
                DatagramPacket dp =
                    new DatagramPacket(
                        send,
                        send.length,
                        InetAddress.getByName("localhost"),
                        5000);
                ds.send(dp);
                byte[] receive = new byte[1024];
                DatagramPacket reply =
                    new DatagramPacket(receive, receive.length);
                ds.receive(reply);
                String ip =
                    new String(reply.getData(), 0,
                    reply.getLength());
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```

        System.out.println("IP Address: " + ip);
        ds.close();
    }
} catch (Exception e) {
    System.out.println(e);
}
}
}
}

```

### **E. SNMP Client for Network Monitoring Using UDP:**

```

import java.net.*;
import java.util.Scanner;

public class SNMPApplication {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("1. Agent");
        System.out.println("2. Manager");
        System.out.print("Enter choice: ");
        int choice = sc.nextInt();
        sc.nextLine();
        try {
            if (choice == 1) {
                DatagramSocket ds = new DatagramSocket(6000);
                byte[] receive = new byte[1024];
                DatagramPacket dp =
                    new DatagramPacket(receive, receive.length);
                System.out.println("SNMP Agent Running");
                ds.receive(dp);
                String request =
                    new String(dp.getData(), 0, dp.getLength());
                System.out.println("Request: " + request);
                String response = "System Active";
                byte[] send = response.getBytes();
                DatagramPacket reply =
                    new DatagramPacket(
                        send,
                        send.length,
                        dp.getAddress(),
                        dp.getPort());
                ds.send(reply);
                ds.close();
            }
            else if (choice == 2) {
                DatagramSocket ds = new DatagramSocket();
                System.out.print("Enter SNMP Request: ");
                String req = sc.nextLine();
                byte[] send = req.getBytes();
                DatagramPacket dp =
                    new DatagramPacket(
                        send,
                        send.length,
                        InetAddress.getByName("localhost"),
                        6000);
                ds.send(dp);
                byte[] receive = new byte[1024];
            }
        }
    }
}

```

```

        DatagramPacket reply =
            new DatagramPacket(receive, receive.length);
        ds.receive(reply);
        String response =
            new String(reply.getData(), 0, reply.getLength());
        System.out.println("Agent Response: " + response);
        ds.close();
    }
} catch (Exception e) {
    System.out.println(e);
}
}
}
}

```

#### **F. file transfer using TCP:**

```

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

public class TCPFileTransfer {
    public static void main(String[] args) {
        Scanner sc = new
Scanner(System.in);
        System.out.println("1. Server");
        System.out.println("2. Client");
        System.out.print("Enter choice: ");
        int choice = sc.nextInt();
        try {
            if (choice == 1) {
                ServerSocket ss = new
ServerSocket(7000);
                System.out.println("File Server
Started");
                Socket s = ss.accept();
                FileInputStream fis =
                new
FileInputStream("test.txt");
                OutputStream os =
s.getOutputStream();
                int ch;
                while ((ch = fis.read()) != -1) {
                    os.write(ch);
                }
                System.out.println("File Sent");
                fis.close();
                s.close();
                ss.close();
            }
            else if (choice == 2) {
                Socket s = new
Socket("localhost", 7000);
                InputStream is =
s.getInputStream();

                FileOutputStream fos =
                new

```

```
FileOutputStream("received.txt");
    int ch;
    while ((ch = is.read()) != -1) {
        fos.write(ch);
    }
    System.out.println("File
Received");
    fos.close();
    s.close();
}
} catch (Exception e) {
    System.out.println(e);
}
}
```

### **Output:**

#### **DNS resolver using UDP:**

```
1. Server
2. Client
Enter choice: 2
Enter Domain Name: google.com
IP Address: 142.250.183.14
```

#### **SNMP Client for Network Monitoring Using UDP:**

```
1. Agent
2. Manager
Enter choice: 2
Enter SNMP Request: GET STATUS
Agent Response: System Active
```

#### **File transfer using TCP:**

```
1. Server
2. Client
Enter choice: 2
    File Received
```

## **1. Study of network simulator (NS). and simulation of congestion control algorithms using NS.**

Network Simulator (NS) is a widely used tool for simulating network protocols and scenarios. It's particularly useful for studying congestion control algorithms. Here's an overview of NS :

- i. NS (from network simulator) refers to a series of discrete-event network simulators, including ns-1, ns-2, and ns-3.
- ii. These simulators are primarily used in research and teaching.
- iii. ns-3 is free software available under the GNU GPLv2 license for research, development, and use.
- iv. The goal of the ns-3 project is to create an open simulation environment for networking research that aligns with modern networking needs.
- v. It encourages community contribution, peer review, and validation of the software.

### **\* Simulation of Network Congestion Control Algorithms using NS :**

To simulate congestion control algorithms in NS, follow these steps:

1. **Install NS:** Begin by installing NS on your system. NS-2 is a popular version of NS used for network simulation.
2. **Create Network Topology:** Define the network topology you want to simulate. This includes nodes, links, and traffic sources.
3. **Configure Nodes:** Configure the nodes in your network, including their IP addresses, routing protocols, and congestion control algorithms.
4. **Define Traffic:** Specify the traffic patterns in your network. This can include TCP or UDP traffic, application-level traffic, and congestion scenarios.
5. **Implement Congestion Control Algorithms:** Implement the congestion control algorithms you want to simulate. Common algorithms include TCP Reno, TCP Vegas, TCP New Reno, TCP Cubic, etc.
6. **Run the Simulation:** Execute the simulation using NS. Monitor performance metrics such as throughput, packet loss, delay, and congestion window size.
7. **Evaluate Results:** Analyze the simulation results to evaluate the performance of the congestion control algorithms under different network conditions and loads.

### **\* Example Scenario :**

Let's say you want to simulate TCP Reno and TCP Cubic congestion control algorithms in a simple network topology with two nodes connected by a link. Here's a simplified example of how you might do it:

- a) Define the network topology in NS, including two nodes and a link between them.
- b) Configure TCP Reno on one node and TCP Cubic on the other node.
- c) Define traffic sources (e.g., FTP traffic) between the nodes.
- d) Run the simulation and collect performance metrics.
- e) Analyze the throughput, packet loss, and congestion window evolution for both TCP Reno and TCP Cubic under varying network conditions (e.g., different bandwidths, delays, and packet loss rates)

## **2. Perform a case study about the different routing algorithms to select the network path with its optimum and economical during data transfer. i. Link State routing ii. Flooding iii. Distance vector.**

### **Case Study: Routing Algorithms for Optimum and Economical Network Path Selection**

In this case study, we will explore three different routing algorithms: Link State Routing, Flooding, and Distance Vector Routing. We will examine how each algorithm works, its advantages and disadvantages, and when it is most suitable to use.

#### **1. Link State Routing**

##### **How it Works:**

- Each router maintains a map of the entire network.
- Routers periodically exchange information about their links with other routers.
- Using this information, each router calculates the shortest path to every other router in the network using Dijkstra's algorithm.
- The shortest path is then used for routing packets.

##### **Advantages:**

- Provides an optimal path from source to destination.
- Adapts well to network changes as it recalculates routes when the network topology changes.

##### **Disadvantages:**

- Requires more memory and processing power on routers to maintain and calculate the network map.
- Sensitive to network failures, as a single link failure can trigger the recalculation of routes by all routers.

##### **Suitability:**

- Best suited for networks where reliability and optimal paths are crucial, such as in large-scale networks with high traffic.

#### **2. Flooding**

##### **How it Works:**

- When a router receives a packet, it forwards the packet to all of its neighbors, except the one from which it received the packet.
- Each router that receives the packet does the same until the packet reaches its destination or its TTL (Time To Live) expires.

##### **Advantages:**

- Simple and easy to implement.
- Does not require routers to maintain routing tables.

##### **Disadvantages:**

- Can lead to network congestion and wasted bandwidth as packets are forwarded to all neighbors.
- Does not guarantee that the packet will reach its destination.

##### **Suitability:**

- Suitable for small, simple networks or for broadcasting information to all nodes in a network.

### **3. Distance Vector Routing**

#### **How it Works:**

- Each router maintains a table that contains the distance (cost) to all other routers in the network.
- Routers periodically exchange information about their routing tables with their neighbors.
- Based on this information, each router updates its routing table to reflect the shortest path to each destination.

#### **Advantages:**

- Simple and easy to implement.
- Consumes less bandwidth compared to flooding as routers only exchange routing information with neighbors.

#### **Disadvantages:**

- Slow to converge, especially in large networks or networks with high traffic.
- May suffer from routing loops or count-to-infinity problems if not properly implemented.

#### **Suitability:**

- Suitable for small to medium-sized networks with moderate traffic where simplicity is more important than optimal routing.

#### **Conclusion**

- **Link State Routing** is suitable for large-scale networks where reliability and optimal paths are critical.
  - **Flooding** is suitable for small, simple networks or for broadcasting information.
  - **Distance Vector Routing** is suitable for small to medium-sized networks where simplicity is more important than optimal routing.
- The choice of routing algorithm depends on the specific requirements of the network, such as size, traffic load, and the importance of optimal routing and simplicity.

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

Learning about handling and configuring networking hardware like RJ-45 connectors, CAT-6 cables, and crimping tools is essential for network setup and maintenance.

**Here's a basic overview of these components and how to work with them:**

#### **1. RJ-45 Connector:**

- The RJ-45 connector is used for Ethernet networking and is commonly found on the ends of Ethernet cables.
- To terminate an Ethernet cable with an RJ-45 connector, you will need a crimping tool and a cable stripper.
- Strip the outer jacket of the cable using the cable stripper, exposing the inner twisted pairs of wires.
- Arrange the wires according to the T568A or T568B wiring standard.
- Insert the wires into the RJ-45 connector and use the crimping tool to crimp the connector onto the cable.

#### **2. CAT-6 Cable:**

- CAT-6 cable is a type of Ethernet cable with enhanced performance specifications.
- It consists of four twisted pairs of copper wires, enclosed in a plastic jacket.
- CAT-6 cables are used for Gigabit Ethernet and other high-speed networking applications.

#### **3. Crimping Tool:**

- A crimping tool is used to attach connectors to the ends of cables.
- It has a built-in mechanism that crimps (squeezes) the connector onto the cable, creating a secure connection.

#### **4. Steps for Crimping an RJ-45 Connector onto a CAT-6 Cable:**

1. Use a cable stripper to remove about 1-1.5 inches of the outer jacket from the end of the CAT-6 cable.
2. Separate the twisted pairs of wires and arrange them in the desired order (T568A or T568B).
3. Trim the ends of the wires to ensure they are all the same length.
4. Insert the wires into the RJ-45 connector, making sure they are fully inserted and in the correct order.
5. Place the connector and cable into the crimping tool and squeeze the handle firmly to crimp the connector onto the cable.
6. Repeat the process for the other end of the cable if necessary.

## **5. Testing:**

- After crimping the connectors, it's a good practice to test the cables using a cable tester to ensure they are properly wired and functional.

## **6. Additional Tips:**

- Always use quality connectors and cables to ensure reliable connections.
- Follow the T568A or T568B wiring standard consistently for all connections in your network.
- Properly label and organize your cables to facilitate troubleshooting and maintenance. Practicing with these components and tools will help you become more proficient in handling and configuring networking hardware.

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

Configuring routers, hubs, switches, and other networking devices is a fundamental skill for network administrators.

**Here's how you can configure these devices using real devices or simulators:**

### **1. Router Configuration:**

Using Real Devices (Example: Cisco Router):

1. Connect to the router using a console cable and terminal emulation software (e.g., PuTTY).
2. Enter privileged EXEC mode by typing enable and entering the enable password.
3. Enter global configuration mode by typing configure terminal.
4. Configure interfaces, IP addresses, routing protocols, and other settings.
5. Save the configuration using write memory or `copy running-config startup-config` to make the changes permanent.

**Using Simulators (Example: Cisco Packet Tracer):**

1. Open Cisco Packet Tracer and add a router to the workspace.
2. Double-click on the router to open the configuration window.
3. Configure interfaces, IP addresses, routing protocols, and other settings using the graphical interface.
4. Save the configuration using the save icon to make the changes permanent.

### **2. Hub Configuration:**

**Using Real Devices (Example: Ethernet Hub):**

- Hubs are simple devices and do not require configuration. They operate at the physical layer and simply forward packets to all connected devices.

**Using Simulators:**

- Since hubs do not have configurations, there's no need to configure them in simulators.

### **3. Switch Configuration:**

**Using Real Devices (Example: Cisco Switch):**

1. Connect to the switch using a console cable and terminal emulation software.
2. Enter privileged EXEC mode and then enter global configuration mode.
3. Configure VLANs, interfaces, IP addresses, and other settings.
4. Save the configuration using write memory or `copy running-config startup-config`.

**Using Simulators (Example: Cisco Packet Tracer):**

1. Open Cisco Packet Tracer and add a switch to the workspace.
2. Double-click on the switch to open the configuration window.

3. Configure VLANs, interfaces, IP addresses, and other settings using the graphical interface.

4. Save the configuration using the save icon.

#### **4. Other Devices:**

- For other devices like firewalls, load balancers, and access points, the configuration process will vary depending on the device and manufacturer.

- Generally, the process involves connecting to the device, entering configuration mode, and then configuring settings such as interfaces, IP addresses, security policies, etc.

It's important to note that configuring real devices should be done with caution, as incorrect configurations can disrupt network operations. Using simulators or virtual labs is a safe way to practice and learn about network device configurations.

## 5. Running and using services/commands like ping, traceroute, nslookup, arp, telnet, ftp, etc.

Running and using networking services and commands like ping, traceroute, nslookup, arp, telnet, and ftp is essential for network troubleshooting and administration.

**Here's a each command/service and how to use them:**

### 1. Ping:

- **Description:** Ping is used to test the reachability of a host on an Internet Protocol (IP) network.

- **Usage:** ping [host]

- **Example:** ping [www.example.com](http://www.example.com)

### 2. Traceroute (or Tracert on Windows):

- **Description:** Traceroute is used to trace the path packets take from one networked device to another.

- **Usage:** traceroute [host]

- **Example:** traceroute [www.example.com](http://www.example.com)

### 3. Nslookup (or Dig on Linux):

- **Description:** Nslookup is used to query Domain Name System (DNS) servers to obtain domain name or IP address mapping.

- **Usage:** nslookup [domain]

- **Example:** nslookup [www.example.com](http://www.example.com)

### 4. Arp:

- **Description:** Arp displays and modifies the IP-to-physical address translation tables used by the Address Resolution Protocol (ARP).

- **Usage:** arp -a (to display ARP cache)

- **Example:** arp -a

### 5. Telnet:

**Description:** Telnet is used to establish a connection to a remote system over the network.

- **Usage:** telnet [host] [port]

- **Example:** telnet www.example.com 80 (to connect to a web server)

## 6. FTP (File Transfer Protocol):

- **Description:** FTP is used to transfer files between a client and a server on a network.
- **Usage:** ftp [host]
- **Example:** ftp <ftp.example.com>

These commands are available on most operating systems and can be used from the command line or terminal. They are valuable for diagnosing network connectivity issues, checking DNS records, examining network paths, and transferring files over a network.

## 6. Network packet analysis using tools like Wireshark, tcpdump, etc.

Network packet analysis tools like Wireshark and tcpdump are essential for troubleshooting network issues, analyzing network traffic, and understanding network protocols.

**Here's how you can use these tools:**

### 1. Wireshark:

#### Downloading and Installing:

- Visit the Wireshark website (<https://www.wireshark.org/>) and download the appropriate version for your operating system.
- Install Wireshark by following the on-screen instructions.

#### Capturing Packets:

- Open Wireshark and select the network interface you want to capture packets on.
- Click the "Start" button to begin capturing packets.

#### Analyzing Packets:

- Wireshark will display a list of captured packets. You can analyze individual packets to view details such as source and destination addresses, protocols, and packet contents.

#### Filtering Packets:

- Use Wireshark's display filters to focus on specific types of packets or traffic patterns.

For example, you can filter packets by protocol (e.g., "tcp") or IP address (e.g., "ip.addr == 192.168.1.1").

#### Saving Captured Packets:

- You can save the captured packets to a file for further analysis or sharing.

### 2. tcpdump:

#### Using tcpdump on Linux:

- Open a terminal window.

Use the following command to start capturing packets on a specific network interface:

```
sudo tcpdump -i [interface].
```

- To save captured packets to a file, use: `sudo tcpdump -i [interface] -w [filename.pcap]`.

- Use Ctrl+C to stop capturing packets.

#### Analyzing Captured Packets:

- After capturing packets, you can analyze them using Wireshark or tcpdump itself.
- To analyze packets using tcpdump, use: `tcpdump -r [filename.pcap]`.

### **Filtering Packets:**

- You can use tcpdump's filtering capabilities to focus on specific packets. For example, to filter packets from a specific IP address, use: ``tcpdump -i [interface] src host [ip_address]``.

**Note:** Be cautious when using packet capture tools on live networks, as capturing and analyzing packets without proper authorization may violate privacy and security policies.

## **7. Network simulation using tools like Cisco Packet Tracer, NetSim, OMNeT++, NS2, NS3, etc.**

Network simulation tools are valuable for designing, analyzing, and simulating network scenarios without the need for physical hardware.

**Here's an overview of some popular network simulation tools:**

### **1. Cisco Packet Tracer:**

#### **Features:**

- Cisco Packet Tracer is a network simulation tool primarily used for Cisco networking equipment.
- It allows users to simulate network configurations, visualize packet flow, and practice networking concepts.

#### **Usage:**

- Design and configure virtual networks with Cisco devices (routers, switches, etc.).
- Simulate network topologies and test configurations.
- Practice troubleshooting and network management tasks.

### **2. NetSim:**

#### **Features:**

- NetSim is a network simulation and emulation software for network engineers.
- It provides a platform to simulate complex networks and test various network protocols and technologies.

#### **Usage:**

- Design and simulate large-scale networks with diverse devices and configurations.
- Test and validate network designs and protocols.
- Analyze network performance and behavior under different conditions.

### **3. OMNeT++:**

#### **Features:**

- OMNeT++ is a discrete event simulation environment for modeling complex systems, including networks.
- It supports various network protocols and technologies and provides a modular framework for simulation development.

#### **Usage:**

- Model and simulate communication networks with different protocols and architectures.
- Analyze network performance, scalability, and reliability.
- Develop and test new network protocols and algorithms.

#### **4. NS2 (Network Simulator 2):**

##### **Features:**

- NS2 is a widely used open-source network simulation tool.
- It provides a simulation environment for modeling and analyzing networking protocols and scenarios.

##### **Usage:**

- Simulate wired and wireless networks with different topologies and protocols.
- Evaluate network performance metrics such as throughput, delay, and packet loss.
- Develop and test new networking algorithms and protocols.

#### **5. NS3 (Network Simulator 3):**

##### **Features:**

- NS3 is the successor to NS2 and provides a more modern and flexible simulation environment.
- It supports a wide range of network technologies and protocols and offers a modular architecture for simulation development.

##### **Usage:**

- Model and simulate complex network scenarios with realistic behavior.
- Analyze the performance of network protocols and applications.
- Develop and test new networking concepts, protocols, and algorithms.

## **8. Socket Programming using UDP and TCP (e.g., simple DNS, date & time client/server, echo client/server, iterative & concurrent servers).**

### **\* Socket Programming :**

Socket programming involves communication between processes over a network using sockets. Here are examples of simple client-server applications using UDP and TCP protocols:

#### **a) UDP (User Datagram Protocol):**

- i. Connectionless protocol.
- ii. Suitable for scenarios where low overhead and speed are crucial.
- iii. No guarantee of message delivery or order.
- iv. Ideal for real-time applications like video streaming, online gaming, and DNS.
- v. Example: Simple DNS server/client.
- vi. UDP server listens on a specific port, receives DNS queries, and responds with corresponding IP addresses.
- vii. UDP client sends DNS queries to the server and processes the responses.

#### **b) TCP (Transmission Control Protocol):**

- i. Connection-oriented protocol.
- ii. Ensures reliable data transfer and proper sequencing.
- iii. Slower than UDP due to additional overhead.
- iv. Ideal for applications requiring data integrity, such as file transfer and email.
- v. Examples:
  - Date & Time server/client:
    - o TCP server provides the current date and time to connected clients.
    - o Clients connect, request the date/time, and receive the response.
  - Echo server/client:
    - o TCP server echoes back any data received from clients.
    - o Clients send data, and the server echoes it back.