

DATA COMMUNICATION & COMPUTER NETWORKS LAB MANUAL

LIST OF EXPERIMENTS

Module 1: Packet Tracer software

1. Study of different types of network cables and practically implement the cross-wired cable and straight through cable using clamping tool.
2. Connect the computers in Local Area Network.
3. Study of basic network commands and network configuration commands.
4. Configure a network topology using packet tracer software.

Module 2: Network simulator (NS2) / Java

5. Implementation of Error Detection / Error Correction Techniques.
6. Implementation of Stop and Wait Protocol and Sliding Window Protocol.
7. Implementation and study of Goback-N and Selective Repeat protocols.
8. Implementation of High-Level Data Link Control.
9. Study of Socket Programming and Client – Server model using java.
10. Write a socket Program for Echo/Ping/Talk commands using java.
11. Implementation of Distance Vector Routing (DVR) algorithm.
12. Implementation of Link State Routing (LSR) algorithm.
13. Study of Network simulator (NS) and simulation of Congestion Control Algorithms using NS2.

EXPERIMENT: 01

AIM: Study of different types of network cables and to practically implement the cross-wired cable and straight through cable using clamping tool.

H/W & S/W REQUIRED: RJ 45 Connector, Crimp Tool, Ethernet Cable (Twisted Pair Cable).

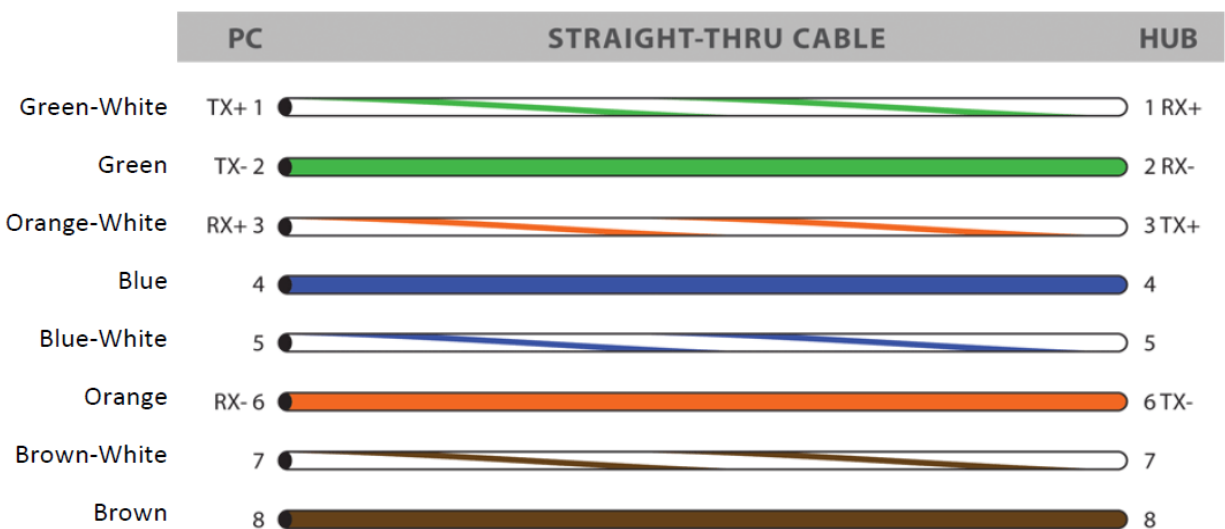
THEORY:

A twisted pair cable consists of two insulated conductors twisted together in the shape of a spiral. It can be shielded or unshielded. The unshielded twisted pair cables are very cheap and easy to install. But they are very badly affected by the electromagnetic noise interference. Twisting of wires will reduce the effect of noise or external interference. The induced emf into the two wires due to interference tends to cancel each other due to twisting. Number of twists per unit length will determine the quality of cable. More twists mean better quality. There are 3 types of UTP cables:

1. Straight-through cable
2. Crossover cable
3. Roll-over cable

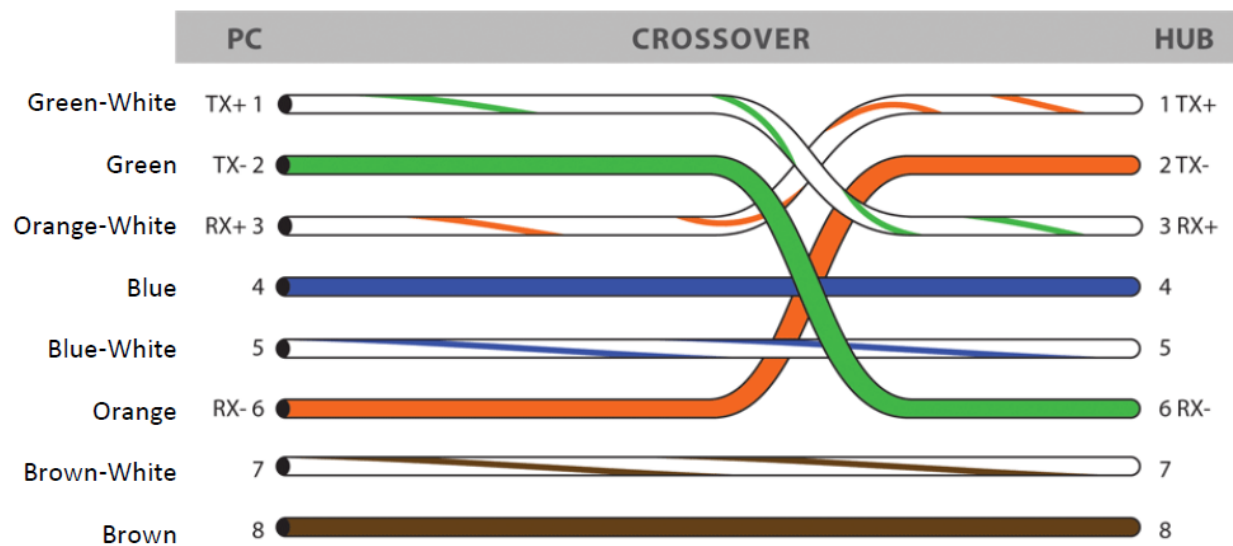
Straight-through cable

Straight-Through refers to cables that have the pin assignments on each end of the cable. In other words, Pin 1 connector A goes to Pin 1 on connector B, Pin 2 to Pin 2 etc. Straight-Through wired cables are most commonly used to connect a host to client. When we talk about cat5e patch cables, the Straight-through wired cat5e patch cable is used to connect computers, printers and other network client devices to the router, switch or hub (the host device in this instance).



Crossover cable

Crossover wired cables (commonly called crossover cables) are very much like Straight-through cables with the exception that TX and RX lines are crossed (they are at opposite positions on either end of the cable). Using the 568-B standard as an example, you will see that Pin 1 on connector A goes to Pin 3 on connector B. Pin 2 on connector A goes to Pin 6 on connector B etc. Crossover cables are most commonly used to connect two hosts directly. Examples would be connecting a computer directly to another computer, connecting a switch directly to another switch, or connecting a router to a router. Note: While in the past when connecting two host devices directly a crossover cable was required. Now days most devices have auto sensing technology that detects the cable and device and crosses pairs when needed.



Roll-over cable

Rollover wired cables most commonly called rollover cables, have opposite pin assignments on each end of the cable or in other words it is "rolled over". Pin 1 of connector A would be connected to Pin 8 of connector B. Pin 2 of connector A would be connected to Pin 7 of connector B and so on. Rollover cables, sometimes referred to as Yost cables are most commonly used to connect a devices console port to make programming changes to the device. Unlike crossover and straight-wired cables, rollover cables are not intended to carry data but instead create an interface with the device.

Crimp Tool

Following is the crimp tool for RJ 45 cable:



ACTIVITY:

STEPS

1. Start by stripping off about 2 inches of the plastic jacket off the end of the cable. Be very careful at this point, as to not nick or cut into the wires, which are inside. Doing so could alter the characteristics of your cable, or even worse render it useless. Check the wires, one more time for nicks or cuts. If there are any, just whack the whole end off, and start over.
2. Spread the wires apart, but be sure to hold onto the base of the jacket with your other hand. You do not want the wires to become untwisted down inside the jacket. Category 5 cable must only have 1/2 of an inch of 'untwisted' wire at the end; otherwise it will be 'out of spec'. At this point, you obviously have ALOT more than 1/2 of an inch of un-twisted wire.
3. You have 2 end jacks, which must be installed on your cable. If you are using a pre-made cable, with one of the ends whacked off, you only have one end to install - the crossed over end. Below are two diagrams, which show how you need to arrange the cables for each type of cable end.

RESULT:

Straight-through cable and crossover cable have been successfully implemented.

SUPPLEMENT MATERIAL:

- [Ethernet Cables, UTP vs STP, Straight vs Crossover, CAT 5,5e,6,7,8 Network Cables](#)
- [How to Make an Ethernet Cable! - FD500R](#)

EXPERIMENT: 02

AIM: To connect two or more PCs or Laptops in a Local Area Network (LAN).

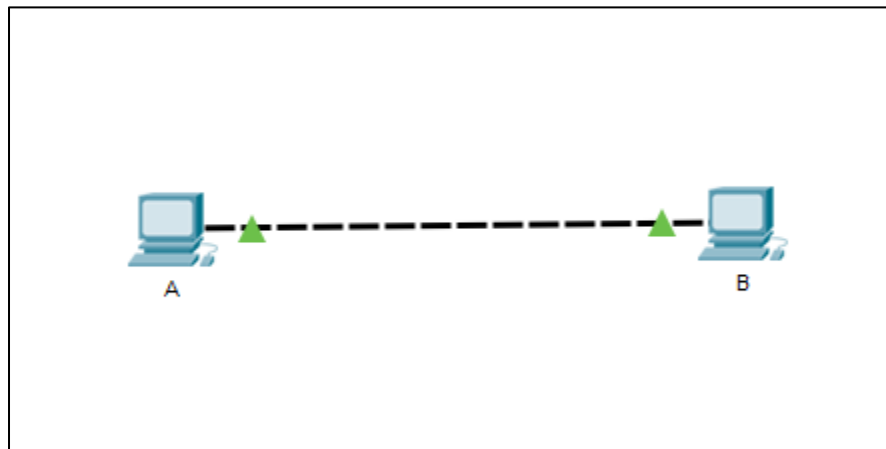
S/W REQUIRED: CISCO Packet Simulator

ACTIVITY 1:

Connecting two computers (point-to-point connection) using an Ethernet crossover cable.

STEPS

1. Open Cisco Packet Tracer (CPT).
2. Click on “End Devices” and select “PC”. Repeat this to create another “PC”.
3. Click on “Connections” and select “Copper Cross-Over” and connect the two PCs. The network design will be as shown below:



4. Click on the first PC to see the PC controls. Click on “Desktop” tab. Click on “IP Configuration”. Give the IP address value as “192.168.1.10”. Close the window.
5. Repeat the above step for the second PC and assign the IP address as “192.168.1.20”.
6. Enter “Simulation” mode.
7. Click on “Add Simple PDU” and click on one PC as the source and click on another PC to make it the destination.
8. Run the simulation.

RESULT:

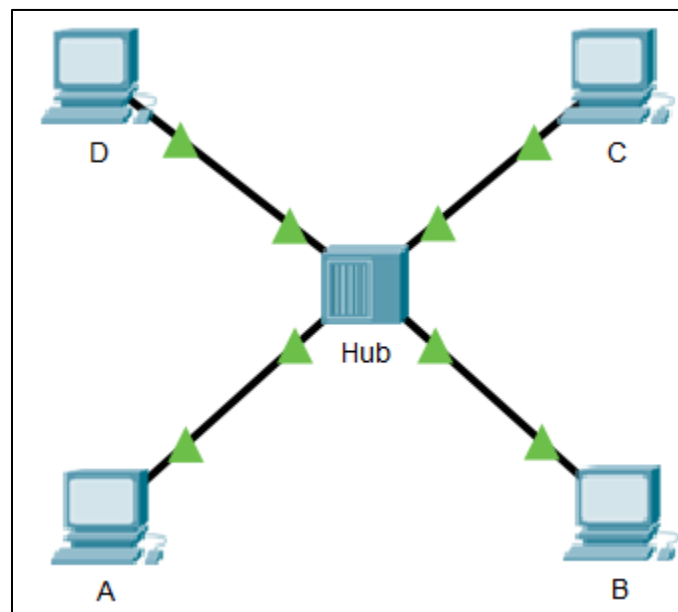
Two PCs were connected using a crossover cable and communication between the two PCs was observed.

ACTIVITY 2:

Creating a star topology using 4 PCs and a hub.

STEPS

1. Open CPT.
2. Click on “End Devices” and select “PC”. Repeat this to create another 3 PCs.
3. Click on “Network Devices” and select “Hubs” sub category. Now select “PT-Hub” and place it on the workspace.
4. Name the hub as “Hub” and the 4 PCs as “A”, “B”, “C”, and “D”.
5. Connect the 4 PCs with the hub using a Copper Straight-Through cable. The network design will be as shown below:



6. Now assign IP address to the PCs A, B, C, and D as 192.168.2.2, 192.168.2.3, 192.168.2.4, and 192.168.2.5 respectively.
7. Click on “Add Simple PDU” and click on PC “A” as the source and click on another PC “D” to make it the destination.
8. Run the simulation and observe the behavior of the hub.

RESULT:

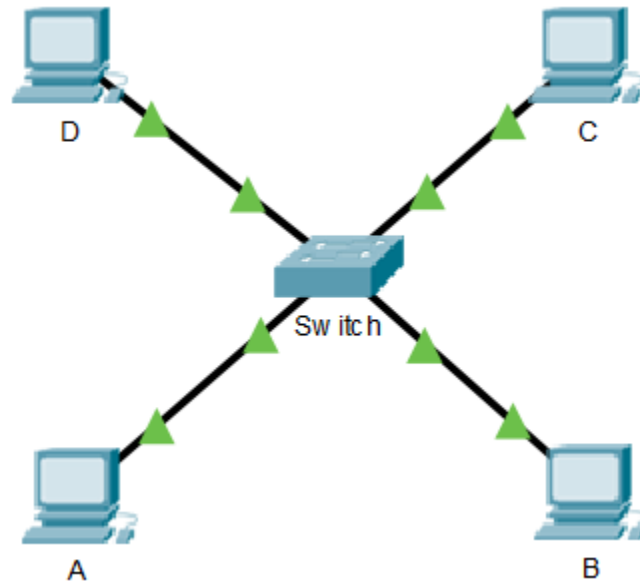
4 PCs were connected to a hub using a straight-through cable forming a star topology and communication between the two PCs was observed. Also, behavior of the hub was observed.

ACTIVITY 3:

Creating a star topology using 4 PCs and a switch.

STEPS

Repeat the steps of the above activity. But, instead of a hub use a switch. While selecting a switch, you can select 2960 switch which a well-known and used switch. The network design will look as shown below:



RESULT:

4 PCs were connected to a switch using a straight-through cable forming a star topology and communication between the two PCs was observed. Also, behavior of the switch was observed.

EXPERIMENT: 03

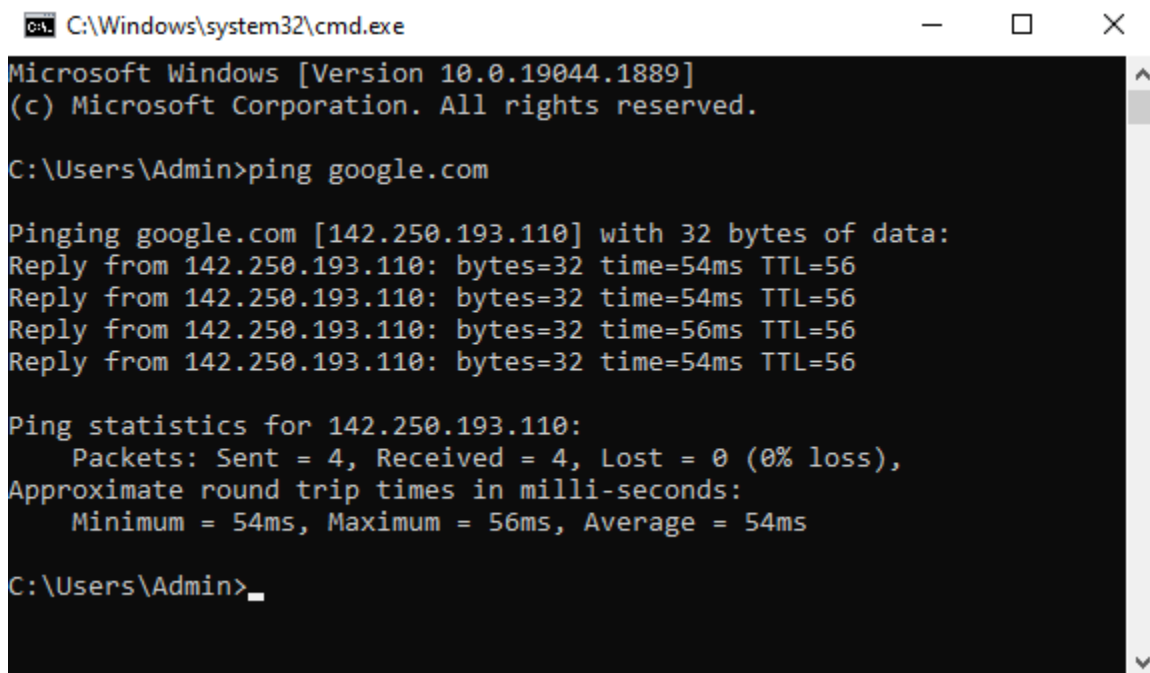
AIM: To study basic network commands and network configuration commands.

S/W REQUIRED: Command Prompt/Terminal, Cisco Packet Tracer

ACTIVITY 1:

STEPS

1. Make sure that your system is connected to Internet.
2. Open command prompt and type “ping google.com”. The output would be as shown below:



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19044.1889]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>ping google.com

Pinging google.com [142.250.193.110] with 32 bytes of data:
Reply from 142.250.193.110: bytes=32 time=54ms TTL=56
Reply from 142.250.193.110: bytes=32 time=54ms TTL=56
Reply from 142.250.193.110: bytes=32 time=56ms TTL=56
Reply from 142.250.193.110: bytes=32 time=54ms TTL=56

Ping statistics for 142.250.193.110:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 54ms, Maximum = 56ms, Average = 54ms

C:\Users\Admin>
```

3. Open command prompt and type “tracert google.com”. The output would be as shown below:

```
C:\Windows\system32\cmd.exe

C:\Users\Admin>tracert google.com

Tracing route to google.com [142.250.193.110]
over a maximum of 30 hops:

  1     1 ms     1 ms     1 ms  192.168.2.1
  2    15 ms     3 ms     3 ms  192.168.18.1
  3     2 ms     2 ms     2 ms  10.0.0.1
  4     7 ms     2 ms     4 ms  14.139.82.99
  5     4 ms   1163 ms     3 ms  10.119.237.93
  6     *         *         *    Request timed out.
  7     *         *         *    Request timed out.
  8    56 ms    62 ms    61 ms  10.119.73.122
  9    60 ms    55 ms    54 ms  72.14.195.128
 10    56 ms    56 ms    56 ms  142.251.227.211
 11    54 ms    54 ms    57 ms  142.251.55.223
 12    55 ms    56 ms    56 ms  maa05s24-in-f14.1e100.net [142.250.193.110]

Trace complete.

C:\Users\Admin>
```

4. Open command prompt and type “nslookup google.com”. The output would be as shown below:

```
C:\Windows\system32\cmd.exe

C:\Users\Admin>nslookup google.com
Server:  dns.google
Address:  8.8.8.8

Non-authoritative answer:
Name:     google.com.domain.name
Address:  95.216.67.149

C:\Users\Admin>
```

5. Open command prompt and type “pathping google.com”. The output would be as shown below:

```
C:\Windows\system32\cmd.exe

C:\Users\Admin>pathping google.com

Tracing route to google.com [142.250.196.46]
over a maximum of 30 hops:
 0  DESKTOP-FBFPE7R.domain.name [192.168.2.2]
 1  192.168.2.1
 2  192.168.18.1
 3  10.0.0.1
 4  14.139.82.99
 5  10.119.237.93
 6  * * *
Computing statistics for 125 seconds...
Hop  RTT      Source to Here   This Node/Link   Address
 0                                Lost/Sent = Pct  Lost/Sent = Pct  DESKTOP-FBFPE7R.domain.
name [192.168.2.2]
 1    3ms      0/ 100 = 0%      0/ 100 = 0%      | 192.168.2.1
 2   11ms      0/ 100 = 0%      0/ 100 = 0%      | 192.168.18.1
 3    6ms      0/ 100 = 0%      0/ 100 = 0%      | 10.0.0.1
 4    6ms      1/ 100 = 1%      0/ 100 = 0%      | 14.139.82.99
 5   ---      100/ 100 =100%   99/ 100 = 99%    | 10.119.237.93
Trace complete.
```

ACTIVITY 2:

STEPS

1. Open Cisco Packet Tracer.
2. Click on “Network Devices” and click on “Routers” and then click on the router “2901” to create a router in the logical workspace.
3. Click on the router again to open the options panel.
4. Click on “CLI” (Command Line Interface) tab.
5. Type the different commands given above and observe the output.

RESULT:

Networking commands and network configuration commands were successfully practiced.

EXPERIMENT: 04

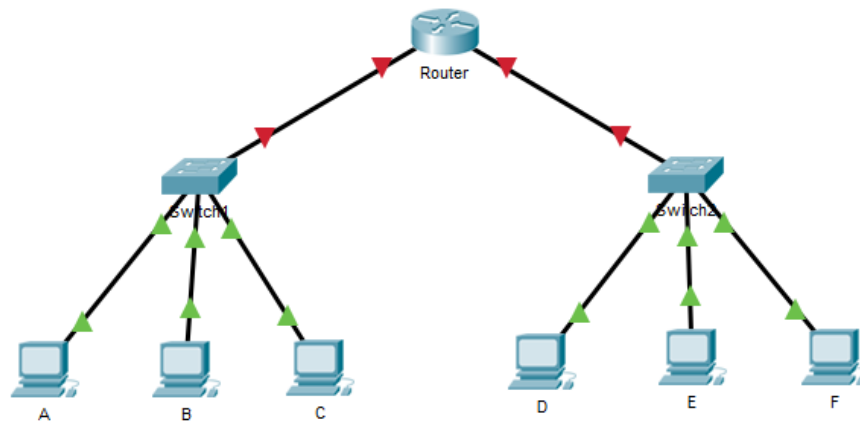
AIM: To configure a network topology using packet tracer.

S/W REQUIRED: Cisco Packet Tracer

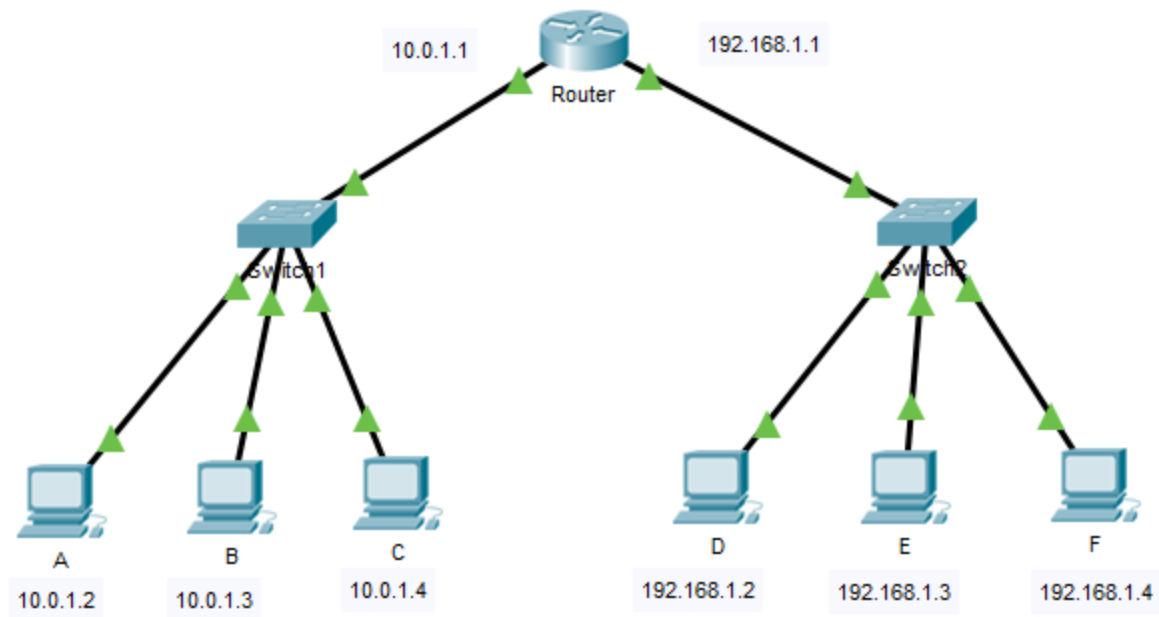
ACTIVITY:

STEPS

1. Open Cisco Packet Tracer.
2. Create a LAN containing a switch (2960) and 3 PCs. Name the PCs as A, B, and C.
3. Assign IP addresses to PCs created in the above step as 10.0.1.2, 10.0.1.3, and 10.0.1.4.
4. Connect all the 3 PCs to the switch.
5. Create another LAN containing a switch and 3 PCs. Name the PCs as D, E, and F.
6. Assign IP addresses to PCs created in the above step as 192.168.1.2, 192.168.1.3, and 192.168.1.4.
7. Connect all the 3 PCs to the switch.
8. Create a router (2901).
9. Connect both the switches to the router. Take care that while connecting the switches to the router, GigabitEthernet ports are selected. The network design looks as shown below:



10. Now, we have to configure the router interfaces. Click on router to open the options. Click on "Config". Click on GigabitEthernet0/0 interface on the left-hand side. Give the IP address as 10.0.1.1 and change (click) the "Port Status" as "On".
11. Repeat the above step for GigabitEthernet0/1 with IP address as 192.168.1.1.
12. For PCs A, B, and C change the "Default Gateway" in "IP Configuration" to 10.0.1.1.
13. For PCs D, E, and F change the "Default Gateway" in "IP Configuration" to 192.168.1.1. Final network design will be as shown below:



14. Go to simulation mode and add a PDU from the source as PC A and the destination as PC F. Click on play and watch the data flow.

RESULT:

Two LANs were connected using a router and the topology has been tested.

EXPERIMENT: 05

AIM: To implement error detection and error correction techniques.

S/W REQUIRED: Java

ACTIVITY:

Program: To implement error detection using even parity.

```
import java.util.*;
import java.io.*;
class Driver
{
    //Method to count number of ones in the data
    static int countOnes(int data[], int b)
    {
        int count = 0;
        for(int i=0; i<b; i++)
        {
            if(data[i] == 1)
                count++;
        }
        return count;
    }
    //Method to check whether the count is even or not
    static boolean isEven(int count)
    {
        if(count % 2 == 0)
            return true;
        else
            return false;
    }
}
```

```
//Method to calculate the parity bit of received data
static int checkEvenParity(int data[], int b)
{
    int count = countOnes(data, b);
    int bit = 0;
    if(isEven(count) == false)
        bit = 1;
    else
        bit = 0;
    return bit;
}

public static void main(String[] args) throws IOException
{
    System.out.println("How many bits you want to send? ");
    Scanner input = new Scanner(System.in);
    int b = input.nextInt();
    //One extra bit to store parity
    int data[] = new int[b+1];
    //Read input data
    System.out.println("Enter the bits, each bit must be separated by a space: ");
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    //Read input data as single line of text
    StringTokenizer bits = new StringTokenizer(br.readLine());
    int i = 0;
    //Extract each bit from the string
    while(bits.hasMoreTokens())
    {
        data[i] = Integer.parseInt(bits.nextToken());
        i++;
    }
}
```

```

int count = countOnes(data, b);

//Set the parity bit
if(isEven(count) == false)
    data[b] = 1;
else
    data[b] = 0;

//Print the data
System.out.println("Original data to be sent is: ");
for(i=0; i<b+1; i++)
    System.out.print(data[i] + " ");

//Read the bit position to change
System.out.println("\nEnter the bit position to change it.");
System.out.println("Position should be less than " + b + ": ");
int pos = input.nextInt();
if(data[pos] == 0)
    data[pos] = 1;
else
    data[pos] = 0;

//Print the data
System.out.println("Data to be sent is: ");
for(i=0; i<b+1; i++)
    System.out.print(data[i] + " ");

//Check for even parity
int bit = checkEvenParity(data, b);
if(data[b] == bit)
    System.out.println("\nNo Errors. Data is accepted.");
else
    System.out.println("\nError found. Data is corrupted.");
}

```

```

}

```


INPUT AND OUTPUT

How many bits you want to send?

5

Enter the bits, each bit must be separated by a space:

1 0 0 1 1

Original data to be sent is:

1 0 0 1 1 1

Enter the bit position to change it.

Position should be less than 5:

2

Data to be sent is:

1 0 1 1 1 1

Error found. Data is corrupted.

Program: To implement error detection using checksum.

```
import java.util.*;
class Driver
{
    //Code to complement the t value to get checksum
    static String complement(String sum, int m)
    {
        char bits[] = sum.toCharArray();
        for(int i=0; i<m; i++)
        {
            if(bits[i] == '1')
                bits[i] = '0';
            else
                bits[i] = '1';
        }
    }
}
```

```

        return new String(bits);
    }

    static String calChecksum(String data[], int k, int m)
    {
        int a = Integer.parseInt(data[0], 2);
        int b = 0;
        int c = 0;

        //Code to add the segments in data array
        for(int i=1; i<k; i++)
        {
            b = Integer.parseInt(data[i], 2);
            c = a+b;
            String temp = Integer.toBinaryString(c);
            if(temp.length() > m)
            {
                temp = temp.substring(1);
                c = Integer.parseInt(temp, 2);
                c = c + 1;
            }
            a = c;
        }

        String sum = Integer.toBinaryString(c);

        //Code to insert 0's before the value in sum to make it 8 bits
        String t = sum;
        if(sum.length() < m)
        {
            int diff = m - sum.length();
            for(int i=0; i<diff; i++)
                t = "0" + t;
        }
    }

```

```

        sum = t;
        return sum;
    }

    static boolean validateChecksum(String data[], int k, int m, String senderChecksum)
    {
        String sum = calChecksum(data, k, m);
        int s = Integer.parseInt(sum, 2);
        int sc = Integer.parseInt(senderChecksum, 2);
        s = s + sc;
        String finalsum = complement(Integer.toBinaryString(s), m);
        if(finalsum.indexOf('1') > -1)
            return false;
        else
            return true;
    }

    public static void main(String[] args)
    {
        System.out.println("How many segments of data? ");
        Scanner input = new Scanner(System.in);
        int k = input.nextInt();
        System.out.println("How many bits per segment? ");
        int m = input.nextInt();
        String data[] = new String[k];
        for(int i=0; i<k; i++)
        {
            System.out.println("Enter data segment " + (i+1) + ": ");
            data[i] = input.next();
        }
        String senderChecksum = complement(calChecksum(data, k, m), m);
        System.out.println("Sender side checksum value: " + senderChecksum);
    }
}

```

```

        System.out.println("Which segment you want to change?");
        System.out.println("Enter a number less than " + k);
        int segment = input.nextInt();
        System.out.println("Data in the segment is: " + data[segment]);
        System.out.println("Which bit you want to change?");
        System.out.println("Enter a number less than " + m);
        int bit = input.nextInt();
        StringBuilder sb = new StringBuilder(data[segment]);
        if(sb.charAt(bit) == '1')
            sb.setCharAt(bit,'0');
        else
            sb.setCharAt(bit,'1');
        data[segment] = new String(sb);
        System.out.println("Data being sent to receiver is:");
        for(int i=0; i<k; i++)
        {
            System.out.print(data[i] + " ");
        }
        System.out.print("\n");
        if(validateChecksum(data, k, m, senderChecksum))
            System.out.println("Data received has no errors!");
        else
            System.out.println("Data received is corrupted!");
    }
}

```

INPUT AND OUTPUT

How many segments of data?

4

How many bits per segment?

8

Enter data segment 1:

10011001

Enter data segment 2:

11100010

Enter data segment 3:

00100100

Enter data segment 4:

10000100

Sender side checksum value: 11011010

Which segment you want to change?

Enter a number less than 4

1

Data in the segment is: 11100010

Which bit you want to change?

Enter a number less than 8

1

Data being sent to receiver is:

10011001 10100010 00100100 10000100

Data received is corrupted!

RESULT:

Error detection and correction techniques were explored and implemented successfully.

EXPERIMENT: 06

AIM: To implement client-server communication using socket programming.

S/W REQUIRED: Java

ACTIVITY:

Program:

```
server.java
import java.net.*;
import java.io.*;
class NewServer
{
    public static void main(String[] args) throws Exception
    {
        ServerSocket ss = new ServerSocket(6000);
        Socket s = ss.accept();
        DataInputStream dis = new DataInputStream(s.getInputStream());
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String stream="";
        String input="";
        while(!stream.equals("stop"))
        {
            stream = dis.readUTF();
            System.out.println("Client: " + stream);
            input = br.readLine();
            dos.writeUTF(input);
            dos.flush();
        }
        dis.close();
    }
}
```

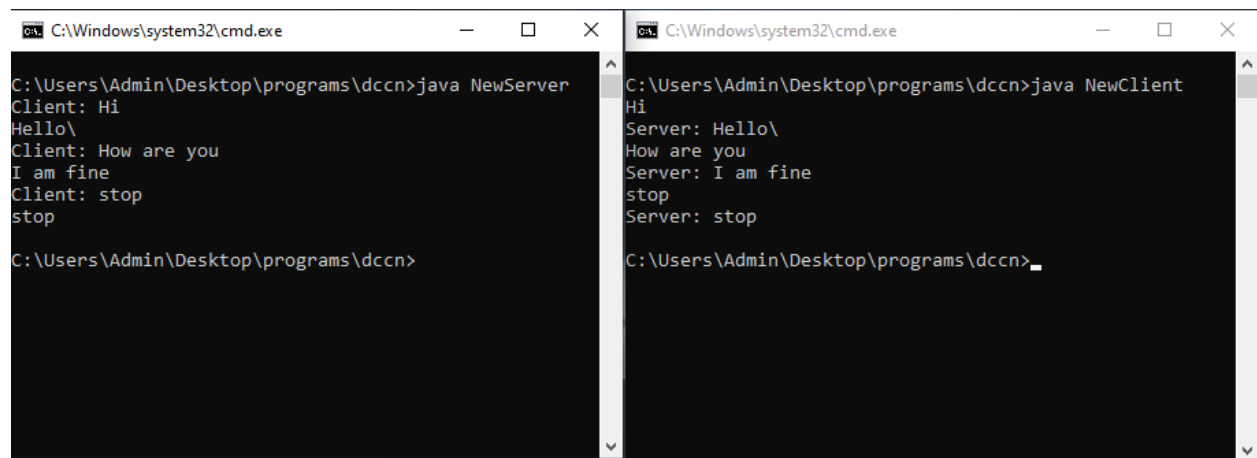
```
        dos.close();  
        s.close();  
        ss.close();  
    }  
}
```

client.java

```
import java.net.*;  
import java.io.*;  
  
class NewClient  
{  
    public static void main(String[] args) throws Exception  
    {  
        Socket s = new Socket("localhost", 6000);  
        DataInputStream dis = new DataInputStream(s.getInputStream());  
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());  
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
        String stream="";  
        String input="";  
        while(!stream.equals("stop"))  
        {  
            input = br.readLine();  
            dos.writeUTF(input);  
            dos.flush();  
            stream = dis.readUTF();  
            System.out.println("Server: " + stream);  
        }  
        dis.close();  
        dos.close();  
    }  
}
```

```
}  
  
    s.close();  
  
}  
  
}
```

INPUT AND OUTPUT



```
C:\Windows\system32\cmd.exe
C:\Users\Admin\Desktop\programs\dccn>java NewServer
Client: Hi
Hello\
Client: How are you
I am fine
Client: stop
stop
C:\Users\Admin\Desktop\programs\dccn>

C:\Windows\system32\cmd.exe
C:\Users\Admin\Desktop\programs\dccn>java NewClient
Hi
Server: Hello\
How are you
Server: I am fine
stop
Server: stop
C:\Users\Admin\Desktop\programs\dccn>
```

RESULT:

Concepts of socket programming were understood and implemented successfully.

EXPERIMENT: 07

AIM: To write a socket program for implementing ping command in Java.

S/W REQUIRED: Java

ACTIVITY:

Program:

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

class NewServer
{
    public static void main(String[] args) throws Exception
    {
        ServerSocket ss = new ServerSocket(6500);
        Socket s = ss.accept();
        DataInputStream dis = new DataInputStream(s.getInputStream());
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());
        String stream="";
        String input="";
        for(int i=1; i<=4; i++)
        {
            stream = dis.readUTF();
            System.out.println("Request " + i + ": " + stream);
            dos.writeUTF("Hello");
            dos.flush();
            Thread.currentThread().sleep(1000);
        }
        dis.close();
    }
}
```

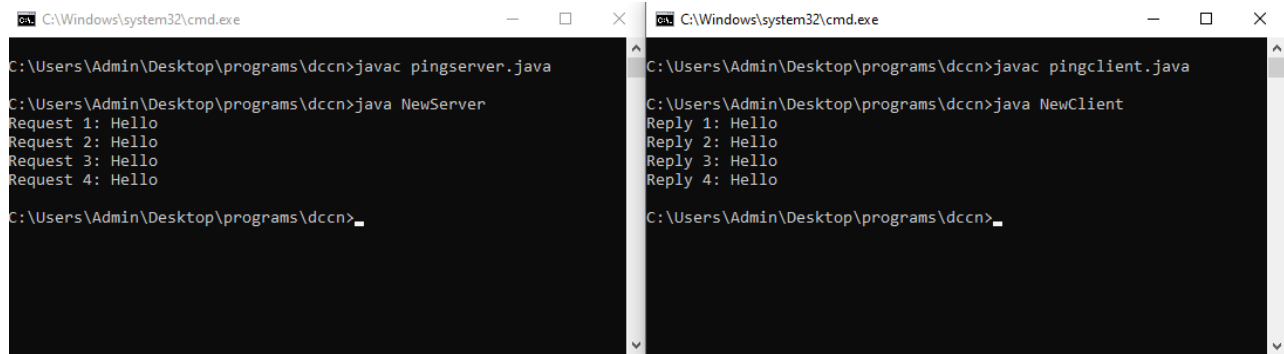
```
        dos.close();  
        s.close();  
        ss.close();  
    }  
}
```

pingclient.java

```
import java.net.*;  
import java.io.*;  
  
class NewClient  
{  
    public static void main(String[] args) throws Exception  
    {  
        Socket s = new Socket("localhost", 6500);  
        DataInputStream dis = new DataInputStream(s.getInputStream());  
        DataOutputStream dos = new DataOutputStream(s.getOutputStream());  
        String stream="";  
        String input="";  
        for(int i=1; i<=4; i++)  
        {  
            dos.writeUTF("Hello");  
            dos.flush();  
            Thread.currentThread().sleep(1000);  
            stream = dis.readUTF();  
            System.out.println("Reply " + i + ": " + stream);  
        }  
        dis.close();  
        dos.close();  
        s.close();  
    }  
}
```

```
}  
  
}
```

INPUT AND OUTPUT



```
C:\Windows\system32\cmd.exe
C:\Users\Admin\Desktop\programs\dccn>javac pingserver.java
C:\Users\Admin\Desktop\programs\dccn>java NewServer
Request 1: Hello
Request 2: Hello
Request 3: Hello
Request 4: Hello
C:\Users\Admin\Desktop\programs\dccn>_

C:\Windows\system32\cmd.exe
C:\Users\Admin\Desktop\programs\dccn>javac pingclient.java
C:\Users\Admin\Desktop\programs\dccn>java NewClient
Reply 1: Hello
Reply 2: Hello
Reply 3: Hello
Reply 4: Hello
C:\Users\Admin\Desktop\programs\dccn>_
```

RESULT:

Ping command was explored and implemented successfully.

EXPERIMENT: 08

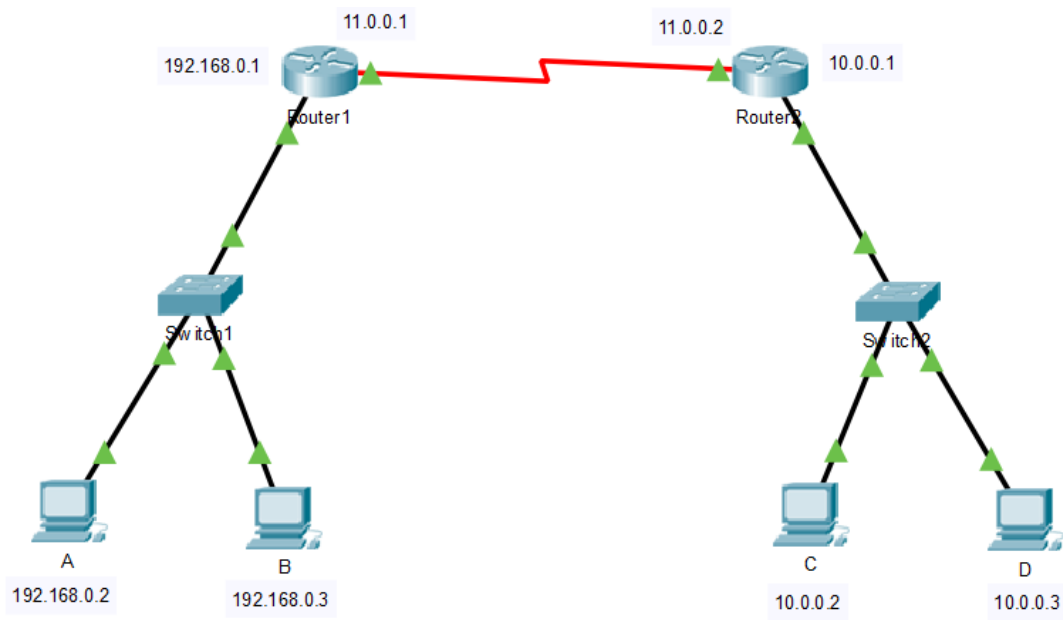
AIM: To implement High-Level Data Link Control (HDLC) protocol.

S/W REQUIRED: Cisco Packet Tracer

ACTIVITY:

STEPS

1. Open Cisco Packet Tracer and create a topology as shown in the below figure. Use **PT-Router** while creating the routers. The red connection (cable) between the two routers is a **Serial DTE** cable.



2. Assign IP addresses to the PCs A, B, C, and D as shown in the above figure.
3. Assign Default Gateway IP address for PCs A and B as 192.168.0.1 and 10.0.0.1 for PCs C and D.
4. Open **Router1** and configure its **FastEthernet0/0** port as shown in the below figure.

Router1

Physical **Config** CLI Attributes

GLOBAL

Settings

Algorithm Settings

ROUTING

Static

RIP

INTERFACE

FastEthernet0/0

FastEthernet1/0

Serial2/0

Serial3/0

FastEthernet4/0

FastEthernet5/0

FastEthernet0/0

Port Status ☒ On

Bandwidth ☒ 100 Mbps ☐ 10 Mbps ☒ Auto

Duplex ☐ Half Duplex ☒ Full Duplex ☒ Auto

MAC Address 000D.BD92.90B6

IP Configuration

IP Address 192.168.0.1

Subnet Mask 255.255.255.0

Tx Ring Limit 10

Equivalent IOS Commands

```

state to down

%LINEPROTO-5-UPDOWN: Line protocol on Interface Serial2/0, changed
state to up

Router(config-if)#exit
Router(config)#interface FastEthernet0/0
Router(config-if)#
  
```

☐ Top

5. Open **Router2** and configure its **FastEthernet0/0** port as shown in the below figure.

Router2

Physical **Config** CLI Attributes

GLOBAL

Settings

Algorithm Settings

ROUTING

Static

RIP

INTERFACE

FastEthernet0/0

FastEthernet1/0

Serial2/0

Serial3/0

FastEthernet4/0

FastEthernet5/0

FastEthernet0/0

Port Status ☒ On

Bandwidth ☒ 100 Mbps ☐ 10 Mbps ☒ Auto

Duplex ☐ Half Duplex ☒ Full Duplex ☒ Auto

MAC Address 0060.2F9A.C193

IP Configuration

IP Address 10.0.0.1

Subnet Mask 255.0.0.0

Tx Ring Limit 10

Equivalent IOS Commands

```

Router(config-router)#network 11.0.0.0
Router(config-router)#network 10.0.0.0
Router(config-router)#
Router(config-router)#end
Router#configure terminal
Enter configuration commands, one per line. End with CNTL/Z.
Router(config)#interface FastEthernet0/0
Router(config-if)#
  
```

☐ Top

6. Open **Router1** and configure its **Serial2/0** port as shown in the below figure. Also, make sure that the **Clock Rate** is as shown in the below figure.

The screenshot shows the configuration window for Router1, specifically the 'Config' tab. The left sidebar shows the 'Serial2/0' interface selected under the 'INTERFACE' section. The main area displays the configuration for 'Serial2/0'. The 'Port Status' is checked 'On'. The 'Duplex' is set to 'Full Duplex'. The 'Clock Rate' is set to '1200'. The 'IP Configuration' section shows 'IP Address' as '11.0.0.1' and 'Subnet Mask' as '255.0.0.0'. The 'Tx Ring Limit' is set to '10'. Below the configuration fields, there is a section for 'Equivalent IOS Commands' which contains the following commands:

```
state to up
Router(config-if)#exit
Router(config)#interface FastEthernet0/0
Router(config-if)#
Router(config-if)#exit
Router(config)#interface Serial2/0
Router(config-if)#
```

At the bottom left, there is a 'Top' button.

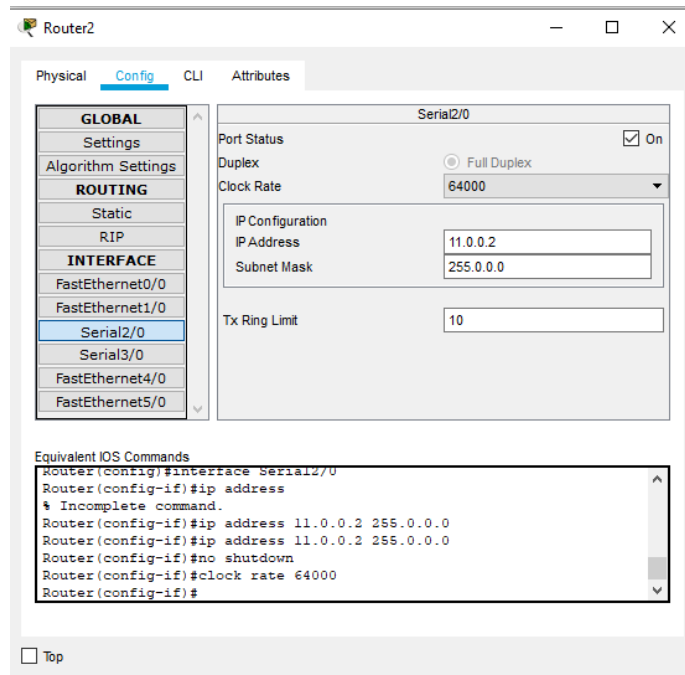
7. Open **Router1** and configure the **RIP** (Routing Information Protocol) settings. Add the networks 11.0.0.0 and 192.168.0.0 as shown in the below figure.

The screenshot shows the configuration window for Router1, specifically the 'Config' tab. The left sidebar shows the 'RIP' protocol selected under the 'ROUTING' section. The main area displays the 'RIP Routing' configuration. There is a 'Network' field with an 'Add' button next to it. Below this, a list of 'Network Address' is shown, containing '11.0.0.0' and '192.168.0.0'. At the bottom right of this list is a 'Remove' button. Below the configuration fields, there is a section for 'Equivalent IOS Commands' which contains the following commands:

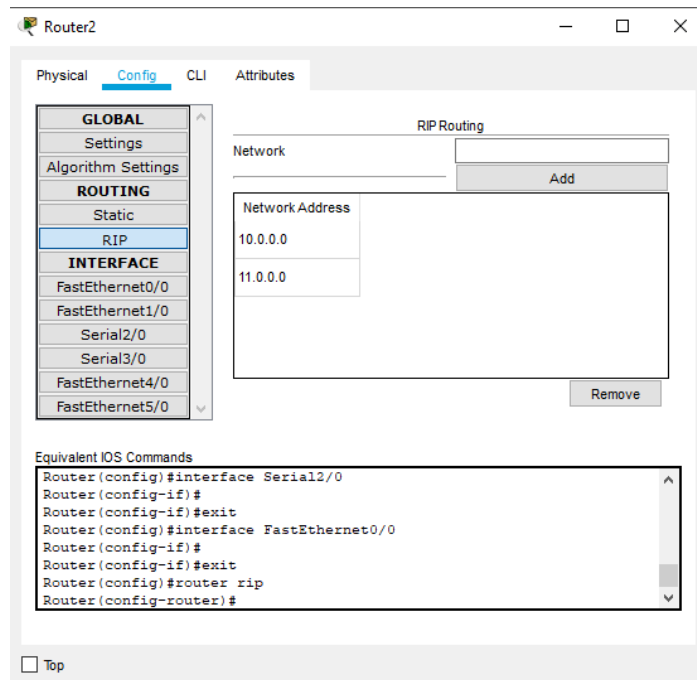
```
Router(config)#interface FastEthernet0/0
Router(config-if)#
Router(config-if)#exit
Router(config)#interface Serial2/0
Router(config-if)#
Router(config-if)#exit
Router(config)#router rip
Router(config-router)#
```

At the bottom left, there is a 'Top' button.

8. Open **Router2** and configure its **Serial2/0** port as shown in the below figure. Also, make sure that the **Clock Rate** is as shown in the below figure.



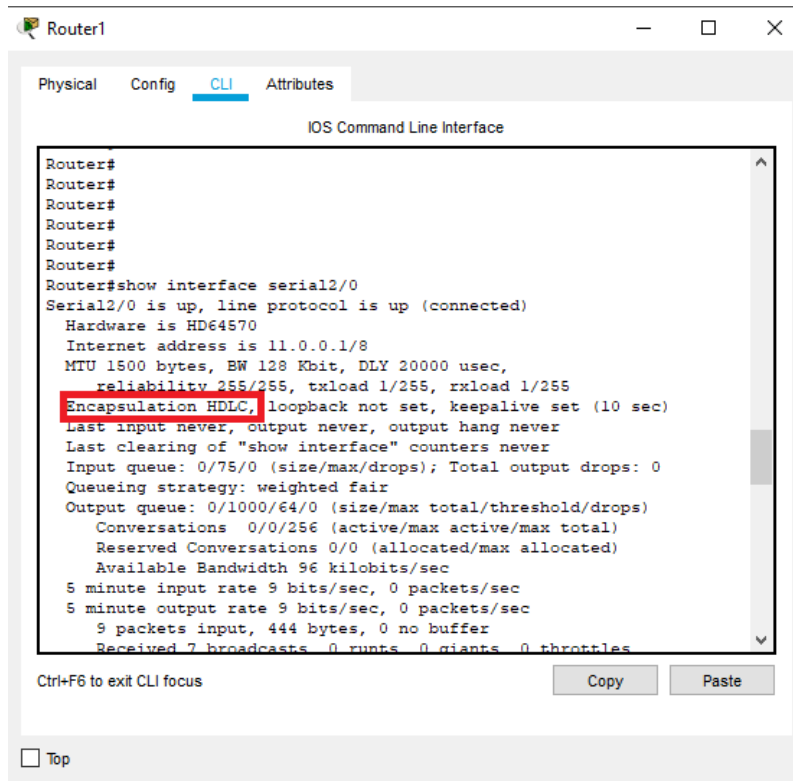
9. Open **Router2** and configure the **RIP** (Routing Information Protocol) settings. Add the networks 11.0.0.0 and 192.168.0.0 as shown in the below figure.



10. By default, the protocol used at data link layer using the serial DTE cable between Router1 and Router2 is HDLC. We can verify it by using the following command in any router's CLI.

```
show interface serial2/0
```

In the result of the above command we see that the encapsulation is already set to HDLC as shown in below figure.



```
Router1
Physical Config CLI Attributes
IOS Command Line Interface
Router#
Router#
Router#
Router#
Router#
Router#
Router#show interface serial2/0
Serial2/0 is up, line protocol is up (connected)
Hardware is HD64570
Internet address is 11.0.0.1/8
MTU 1500 bytes, BW 128 Kbit, DLY 20000 usec,
  reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation HDLC, loopback not set, keepalive set (10 sec)
Last input never, output never, output hang never
Last clearing of "show interface" counters never
Input queue: 0/75/0 (size/max/drops); Total output drops: 0
Queueing strategy: weighted fair
Output queue: 0/1000/64/0 (size/max total/threshold/drops)
  Conversations 0/0/256 (active/max active/max total)
  Reserved Conversations 0/0 (allocated/max allocated)
  Available Bandwidth 96 kilobits/sec
5 minute input rate 9 bits/sec, 0 packets/sec
5 minute output rate 9 bits/sec, 0 packets/sec
  9 packets input, 444 bytes, 0 no buffer
  Received 7 broadcasts 0 runs 0 giants 0 throttles
```

In order to set the encapsulation to HDLC, we need to type the following commands in router's EXEC mode or privileged mode:

```
configure terminal
interface serial2/0
encapsulation hdlc
```

RESULT:

HDLC concepts were understood and simulated successfully using Cisco Packet Tracer.

EXPERIMENT: 09

AIM: To implement stop-and-wait protocol.

S/W REQUIRED: NS2

ACTIVITY:

Program:

```
#Implmentation of Stop-and-wait protocol

#Create Simulator Object
set ns [new Simulator]

#Initialization code
set namfile [open out.nam w]
$ns namtrace-all $namfile

#Termination code
proc finish {} {
    global ns namfile
    $ns flush-trace
    close $namfile
    puts "Executing NAM..."
    exec nam out.nam &
    exit 0
}

#Creating nodes
set n1 [$ns node]
set n2 [$ns node]

#Assign labels to nodes
```

```
$ns at 0.0 "$n1 label Sender"
$ns at 0.0 "$n2 label Receiver"

#Creating links
$ns duplex-link $n1 $n2 10Mb 10ms DropTail

#Setting queue limit
$ns queue-limit $n1 $n2 10

#Creating TCP agent and TCP sink
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink

#Connect TCP agent and TCP sink
$ns connect $tcp $sink

#Creating FTP application
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#Setting TCP window size and congestion window size
#window_ is the upper bound on the advertised window for the TCP
connection.
$tcp set window_ 1
#maxcwnd_ is the upper bound on the congestion window for the TCP
connection.
$tcp set maxcwnd_ 1

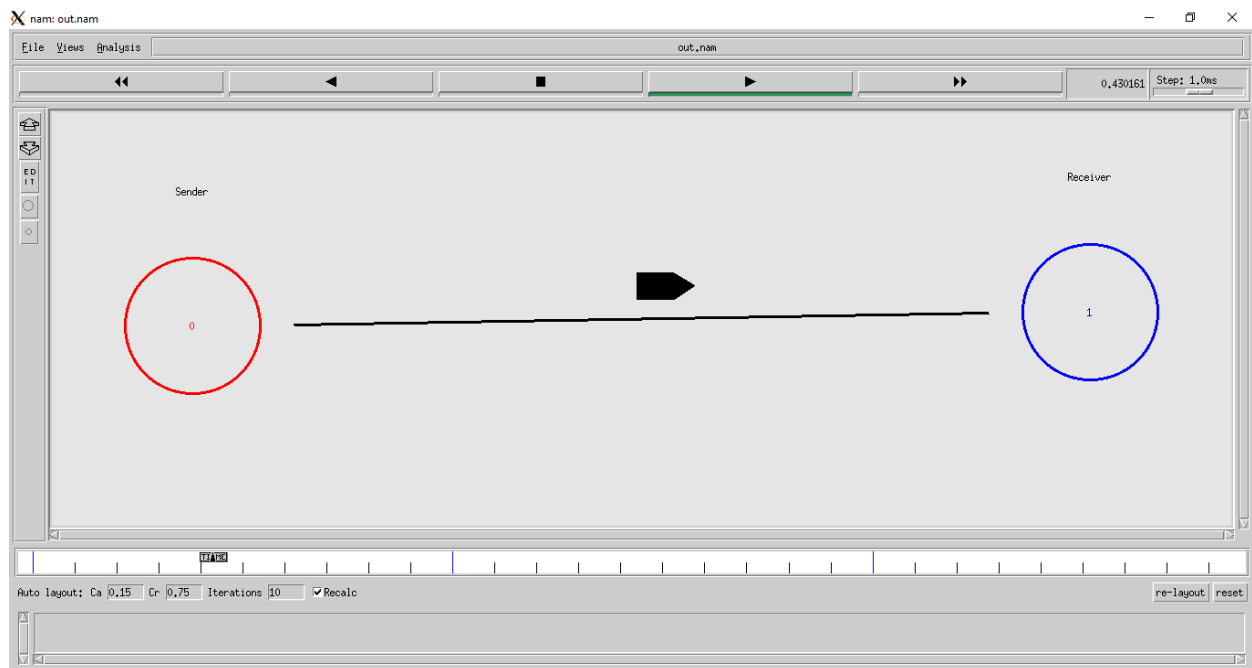
#Set the location of node labels
$ns at 0.0 "$n1 label-at up"
```

```
$ns at 0.0 "$n1 color Red"
$ns at 0.0 "$n2 label-at up"
$ns at 0.0 "$n2 color Blue"

#Set the timing of events
$ns at 0.3 "$ftp start"
$ns at 2.8 "$ftp stop"
$ns at 3.0 "finish"

$ns run
```

INPUT AND OUTPUT



RESULT:

Stop-and-wait protocol was understood and implemented successfully.

EXPERIMENT: 10

AIM: To implement Sliding Window, Go Back N, and Selective Repeat protocols.

S/W REQUIRED: NS2

ACTIVITY:

Program:

```
#Implementation of Go-Back-N protocol

#Create Simulator Object
set ns [new Simulator]

#Initialization code
set namfile [open out.nam w]
$ns namtrace-all $namfile

#Termination code
proc finish {} {
    global ns namfile
    $ns flush-trace
    close $namfile
    puts "Executing NAM..."
    exec nam out.nam &
    exit 0
}

#Creating nodes
set n1 [$ns node]
set n2 [$ns node]
```

```
#Assign labels to nodes
$ns at 0.0 "$n1 label Sender"
$ns at 0.0 "$n2 label Receiver"

#Creating links
$ns duplex-link $n1 $n2 10Mb 10ms DropTail

#Setting queue limit
$ns queue-limit $n1 $n2 10

#Creating TCP agent and TCP sink
set tcp [new Agent/TCP]
$ns attach-agent $n1 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n2 $sink

#Connect TCP agent and TCP sink
$ns connect $tcp $sink

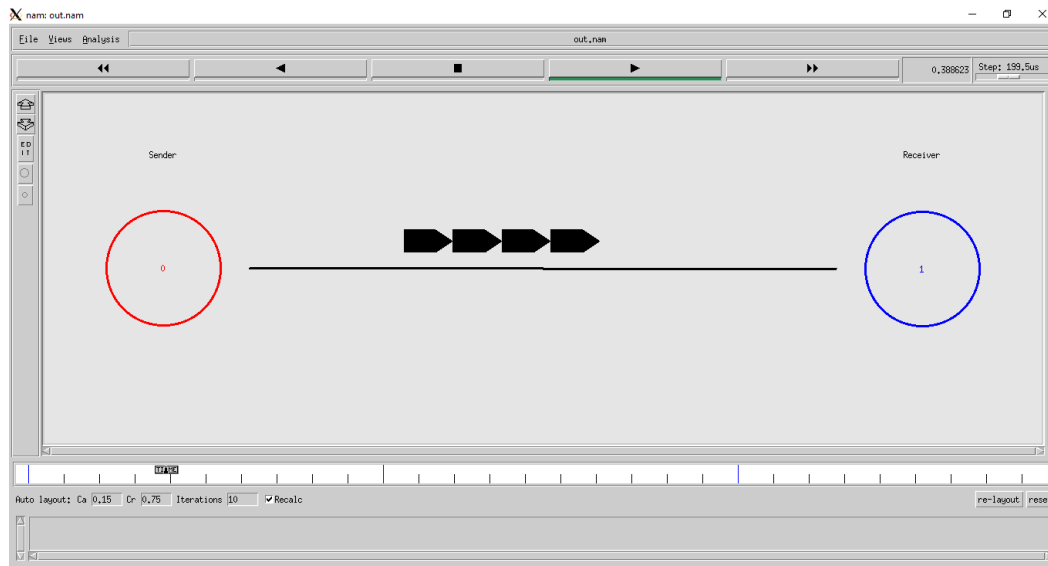
#Creating FTP application
set ftp [new Application/FTP]
$ftp attach-agent $tcp

#Setting TCP window size and congestion window size
#window_ is the upper bound on the advertised window for the TCP
connection.
$tcp set window_ 4
#maxcwnd_ is the upper bound on the congestion window for the TCP
connection.
$tcp set maxcwnd_ 4

#Set the location of node labels
```

```
$ns at 0.0 "$n1 label-at up"  
$ns at 0.0 "$n1 color Red"  
$ns at 0.0 "$n2 label-at up"  
$ns at 0.0 "$n2 color Blue"  
  
#Set the timing of events  
$ns at 0.3 "$ftp start"  
$ns at 2.8 "$ftp stop"  
$ns at 3.0 "finish"  
  
$ns run
```

INPUT AND OUTPUT



RESULT:

Sliding window, Go Back N, and Selective Repeat protocols were understood and implemented successfully.

REFERENCES:

- https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

EXPERIMENT: 11

AIM: To implement Distance Vector Routing algorithm.

S/W REQUIRED: NS2

ACTIVITY:

Program:

```
#Create object for Simulator class
set ns [new Simulator]

#Initialization script
set tracefile [open out.tr w]
$ns trace-all $tracefile
set namfile [open out.nam w]
$ns namtrace-all $namfile

#Assign colors to traffic flows
$ns color 1 Red
$ns color 2 Blue

#Termination script
proc finish { } {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}

#Creating nodes
```

```

for {set i 0} {$i < 12} {incr i 1} {
    set n($i) [$ns node]
}

#Creating links
for {set i 0} {$i < 8} {incr i 1} {
    $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

#Creating UDP, Null agents and CBR applications
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $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 $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500

```



```
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

#Set routing protocol to DVR
$ns rtproto DV

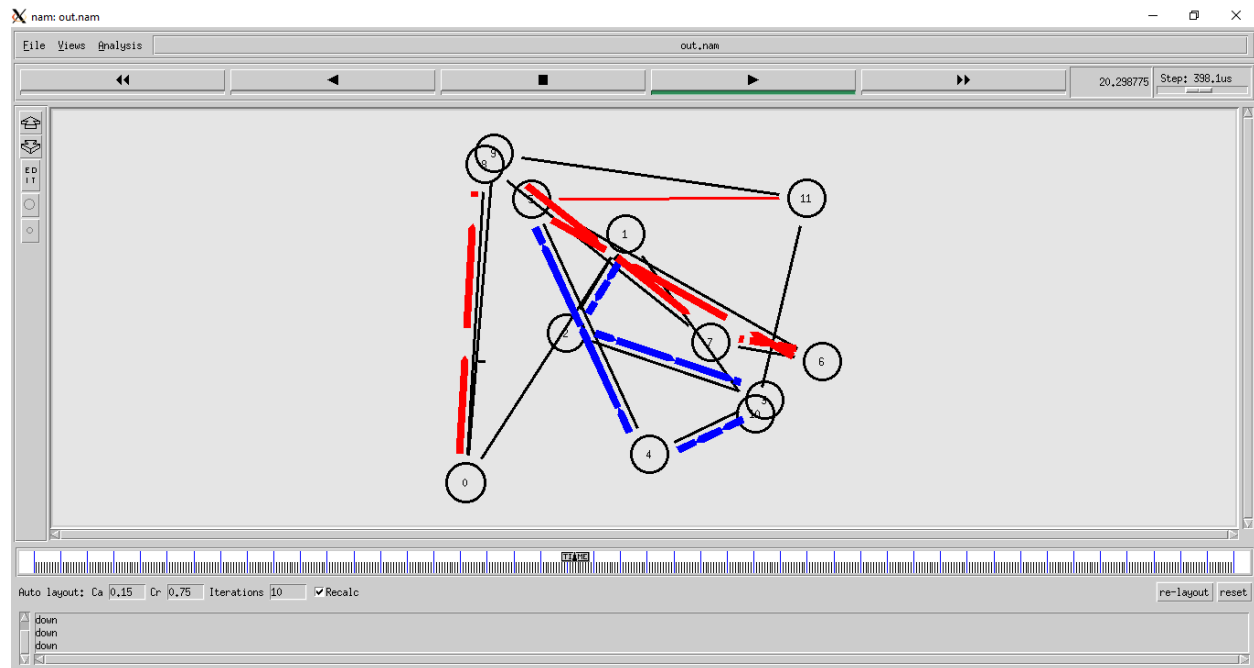
#Make links up and down to test DVR
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2

#Assign time to events
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45.0 "finish"

#Execute the script
$ns run
```

INPUT AND OUTPUT



RESULT:

Distance Vector Routing has been understood and implemented successfully.

REFERENCES:

- <https://www.scaler.com/topics/computer-network/distance-vector-routing-algorithm/>

EXPERIMENT: 12

AIM: To implement Link State Routing algorithm.

S/W REQUIRED: NS2

ACTIVITY:

Program:

```
#Create object for Simulator class
set ns [new Simulator]

#Initialization script
set tracefile [open out.tr w]
$ns trace-all $tracefile
set namfile [open out.nam w]
$ns namtrace-all $namfile

#Assign colors to traffic flows
$ns color 1 Red
$ns color 2 Blue

#Termination script
proc finish { } {
    global ns tracefile namfile
    $ns flush-trace
    close $tracefile
    close $namfile
    exec nam out.nam &
    exit 0
}

#Creating nodes
```

```

for {set i 0} {$i < 12} {incr i 1} {
    set n($i) [$ns node]
}

#Creating links
for {set i 0} {$i < 8} {incr i 1} {
    $ns duplex-link $n($i) $n([expr $i+1]) 1Mb 10ms DropTail
}

$ns duplex-link $n(0) $n(8) 1Mb 10ms DropTail
$ns duplex-link $n(1) $n(10) 1Mb 10ms DropTail
$ns duplex-link $n(0) $n(9) 1Mb 10ms DropTail
$ns duplex-link $n(9) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(10) $n(11) 1Mb 10ms DropTail
$ns duplex-link $n(11) $n(5) 1Mb 10ms DropTail

#Creating UDP, Null agents and CBR applications
set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $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 $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500

```

```
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp1 $null0

#Set routing protocol to LSR
$ns rtproto LS

#Make links up and down to test LSR
$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)
$ns rtmodel-at 20.0 up $n(7) $n(6)

$udp0 set fid_ 1
$udp1 set fid_ 2

#Assign time to events
$ns at 1.0 "$cbr0 start"
$ns at 2.0 "$cbr1 start"
$ns at 45.0 "finish"

#Execute the script
$ns run
```

INPUT AND OUTPUT

EXPERIMENT: 13

AIM: To implement congestion control algorithm.

S/W REQUIRED: NS2

ACTIVITY:

Program:

```
#create simulator
set ns [new Simulator]

#initialization script
set namfile [open out.nam w]
$ns namtrace-all $namfile
set outfile [open congestion.xg w]

#termination script
proc finish {} {
    global ns namfile
    $ns flush-trace
    close $namfile
    exec nam out.nam &
    #exec xgraph congestion.xg -geometry 300x300 &
    exit 0
}

#to create nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
```

```

set n5 [$ns node]

# to create the link between the nodes with bandwidth, delay and queue
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 0.3Mb 200ms DropTail
$ns duplex-link $n3 $n4 0.5Mb 40ms DropTail
$ns duplex-link $n3 $n5 0.5Mb 30ms DropTail

# Sending node is 0 with agent as Reno Agent
set tcp1 [new Agent/TCP/Reno]
$ns attach-agent $n0 $tcp1

# receiving (sink) node is n4
set sink1 [new Agent/TCPSink]
$ns attach-agent $n4 $sink1

# establish the traffic between the source and sink
$ns connect $tcp1 $sink1

# Setup a FTP traffic generator on "tcp1"
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP

# procedure to plot the congestion window
proc plotWindow {tcpSource outfile} {
    global ns
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]

    # the data is recorded in a file called congestion.xg (this can be plotted # using xgraph or gnuplot.
    # this example uses xgraph to plot the cwnd_

```



```

    puts $outfile "$now $cwnd"
    $ns at [expr $now+0.1] "plotWindow $tcpSource $outfile"
}

#timing events
$ns at 0.0 "plotWindow $tcp1 $outfile"
# start/stop the traffic
$ns at 0.1 "$ftp1 start"
$ns at 40.0 "$ftp1 stop"
# Set simulation end time
$ns at 50.0 "finish"

# Run simulation
$ns run

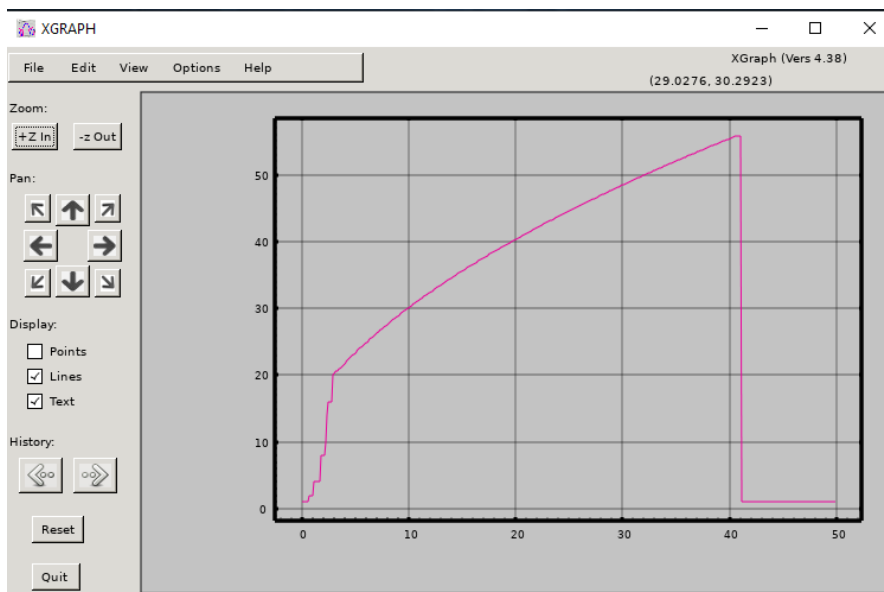
```

INPUT AND OUTPUT

Note: For observing output, we need to install *xgraph*, a third-party software. See the provided instructions in Google Classroom on how to install and configure *xgraph* for observing the output.

Output can be observed by using the following command:

`xgraph congestion.xg`



RESULT:

Congestion control algorithm is simulated and output is observed successfully.

REFERENCES:

- <https://www.scaler.com/topics/computer-network/tcp-congestion-control/>