COIMBATORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Autonomous | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with "A" Grade

DEPARTMENT OF

COMPUTER SCIENCE AND ENGINEERING

NETWORKS LABORATORY (U23CSP10)

2025 - 2026

CIET COIMBATORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Autonomous | Approved by AICTE | Affiliated to Anna University | Accredited by NAAC with "A" Grade

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of work done by Mr. /Ms ……..…..... in the **NETWORKS LABORATORY (U23CSP10)** of this institution, as prescribed by the Anna University for the Fifth Semester Computer Science and Engineering, during the year 2025 - 2026.

_____ _____

Ms. A. SUBHA PRIYADHARSHINI B.Tech., M.E.,
ASSISTANT PROFESSOR
Department of Computer Science & Engineering,
Coimbatore Institute of Engineering and Technology,
Coimbatore – 641 109.

Dr. K. PUSHPALATHA M.E., Ph.D.,
HEAD OF THE DEPARTMENT
Department of Computer Science & Engineering,
Coimbatore Institute of Engineering and Technology,
Coimbatore – 641 109.

Submitted for the University Practical Examination held on ……..…….. at Coimbatore Institute of Engineering and Technology, Coimbatore – 641 109.

Register Number: | | |

Internal Examiner **External Examiner**

INDEX

S.NO.	DATE	LIST OF EXPERIMENTS	PAGE NO.	MARKS	SIGNATURE
1		NETWORK TOPOLOGY AND SIMPLE LAN			
2		STUDY OF BASIC NETWORK COMPONENTS AND COMMANDS			
3A		ECHO CLIENT AND ECHO SERVER USING TCP SOCKETS			
3B		CHAT CLIENT AND CHAT SERVER USING TCP SOCKETS			
4		SIMULATIONS OF DNS USING UDP SOCKETS			
5A		IMPLEMENTATION OF STOP-AND-WAIT PROTOCOL USING TCP SOCKET			
5B		IMPLEMENTATION OF SLIDING WINDOW PROTOCOL USING TCP SOCKET			
6A		IMPLEMENTATION OF GO-BACK-N PROTOCOL USING TCP SOCKET			
6B		IMPLEMENTATION OF SELECTIVE REPEAT PROTOCOL USING TCP SOCKET			
7		VLAN CONFIGURATION USING CISCO PACKET TRACER			
8A		DHCP CONFIGURATION IN A SIMULATED ENVIRONMENT			
8B		DNS CONFIGURATION IN A SIMULATED ENVIRONMENT			
9A		INTERCONNECTING TWO LANS USING A SINGLE ROUTER IN CISCO PACKET TRACER			
9B		INTERCONNECTING TWO LANS USING TWO ROUTERS IN CISCO PACKET TRACER			

9C		INTERCONNECTING FOUR LANS USING TWO ROUTERS IN CISCO PACKET TRACER			
10 A		STUDY OF WIRE SHARK TOOL			
10 B		WIRE SHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS			

EXP.NO:1 **NETWORK TOPOLOGY AND SIMPLE LAN**

DATE:

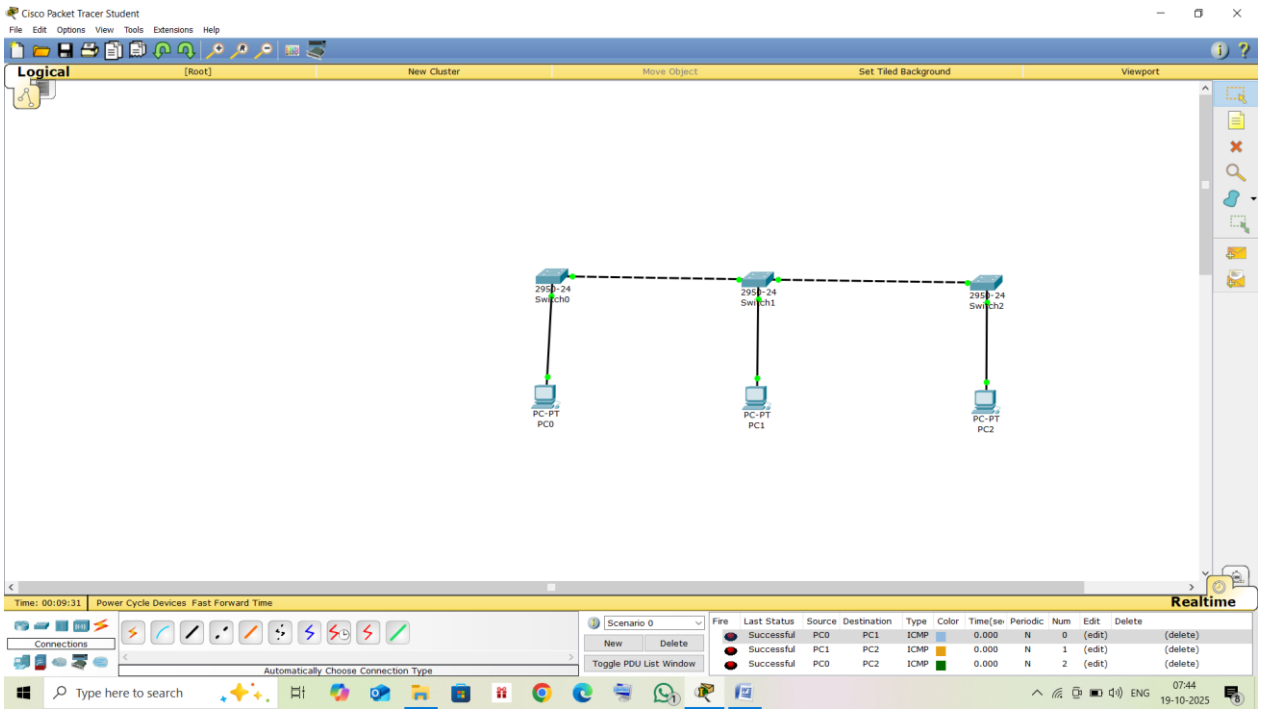
AIM:

To design a bus, star, ring topology and simple LAN using Cisco packet tracker

1. **Bus topology:**

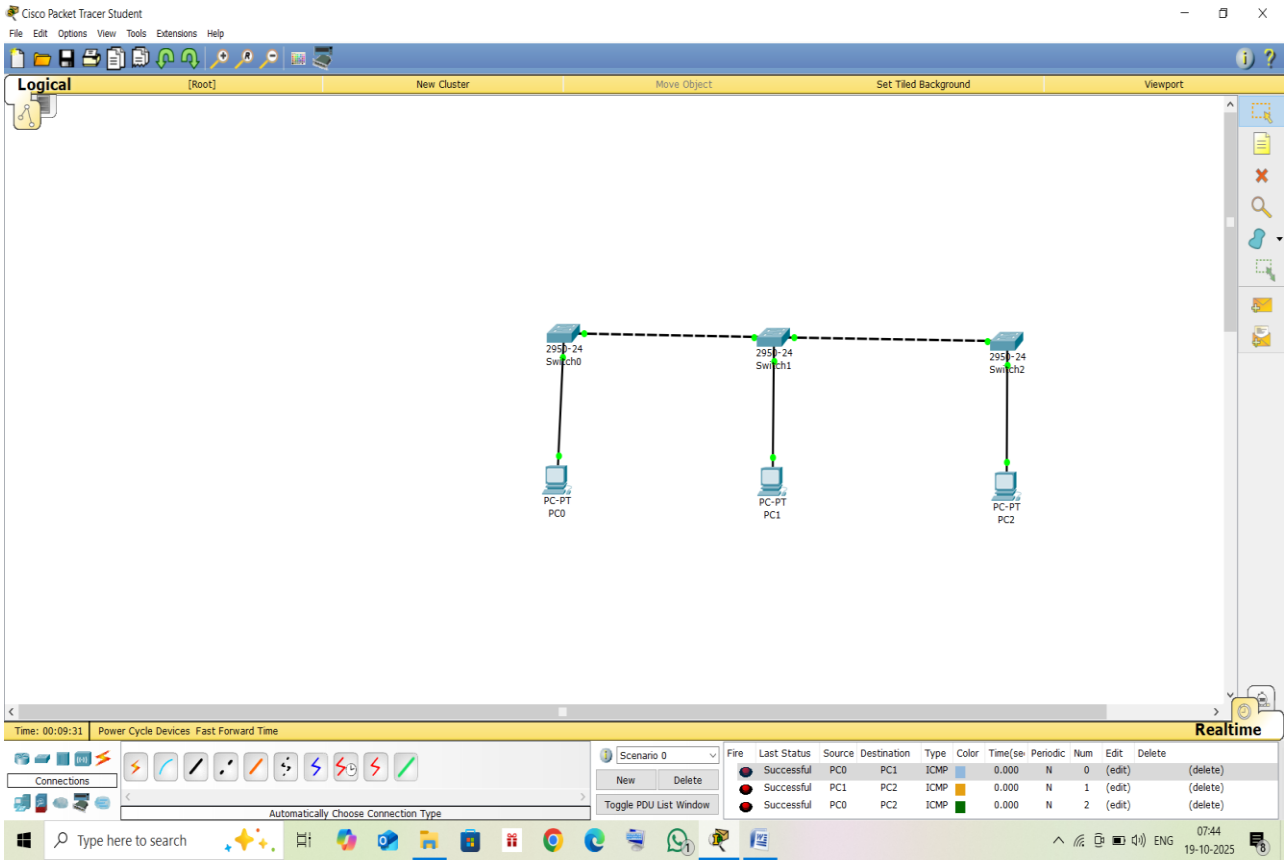
In local area network, it is a single network cable runs in the building or campus and all nodes are connected along with this communication line with two endpoints called bus are backbone .In other words, it is a multipoint data point communication circuit that is easily control data flow between the computer because this configuration allows all station to receive every transmission over the network. For bus topology we build network using three generic pc which are serially connected with three switches using copper straight through cable and switches are interconnected using copper cross over cable.

Design of bus topology:



To configure the IP address of an interface, we configure all PC one by one click on PC, Open DESKTOP window, Fill IP address, Fill subnet mask. After that, simulate the network using simulation.

Simulation modal for Bus Topology:



2. Star Topology:

In star topology all the computers are connected to a single hub through a cable. This hub is the central node and all other nodes are connected to the central node.

Step 1- Choose the end devices to be configured with IP address and add them to the screen.

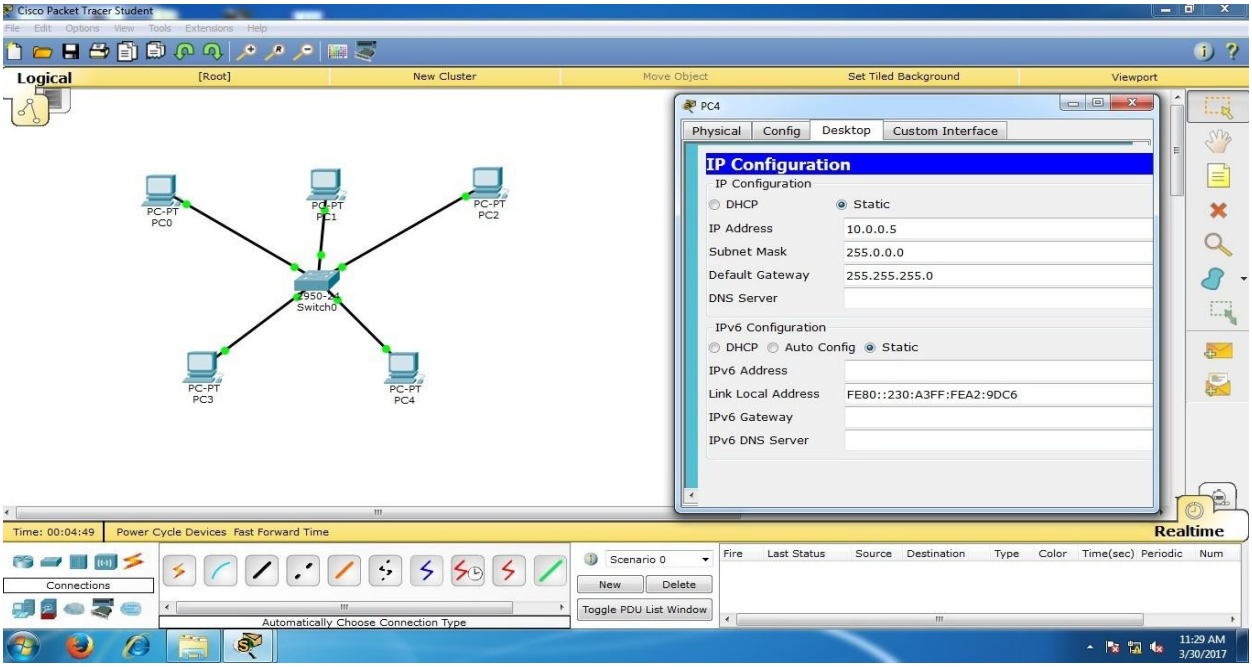
Step 2- Place more such devices to form a star pattern and configure each of them with IP address.

Step 3- Connect all the devices to a common hub.

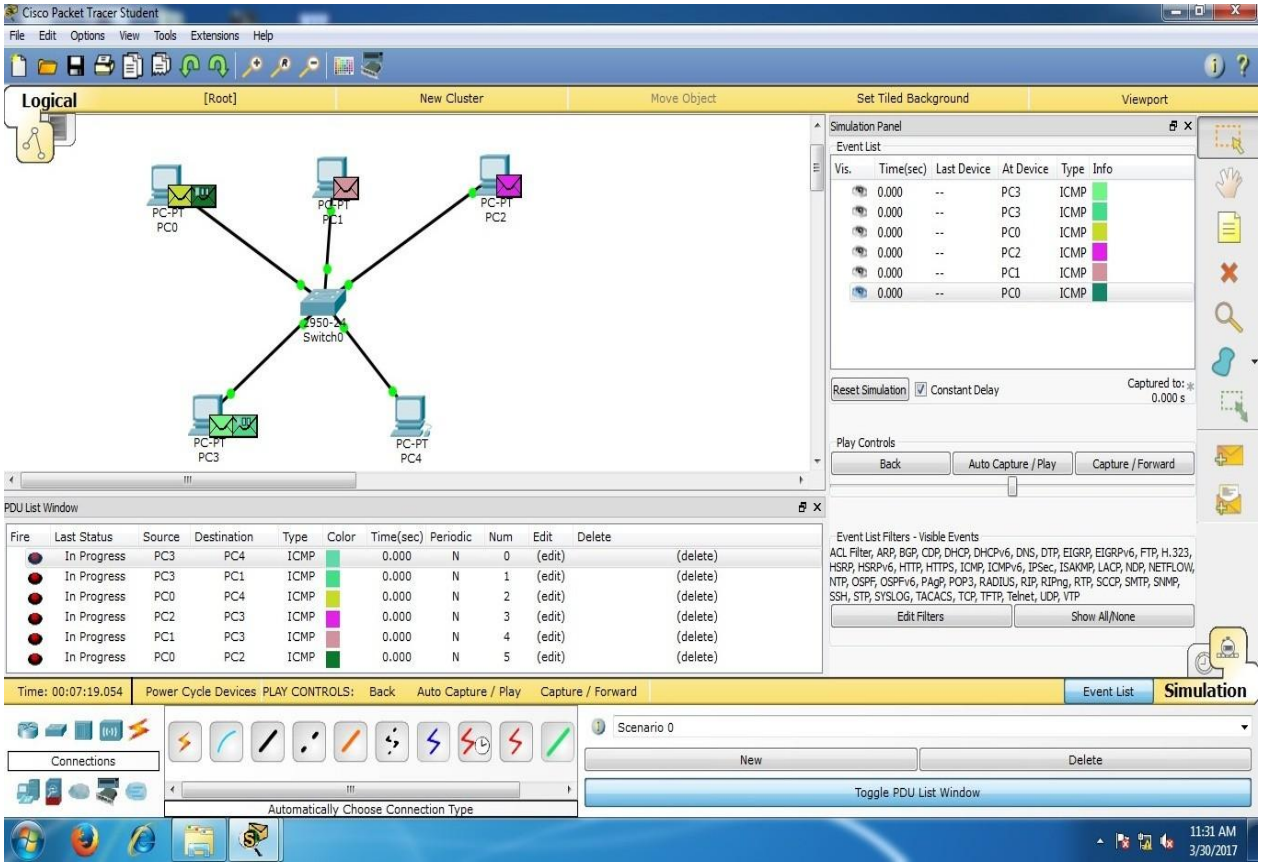
Step 4- Click the message packets and ‘add simple PDU’ and select the receiver and sender within the star connected devices.

Step 5-Click on simulation-Auto capture/Play to observe the transfer of data in the network system.

Design of Star Topology:



Simulation modal of Star Topology:



3. Ring Topology:

It is called ring topology because it forms a ring as each computer is connected to another computer, with the last one connected to the first. Exactly two neighbors for each device.

Step 1: Choose the end device to be configured with IP address and add them to the screen.

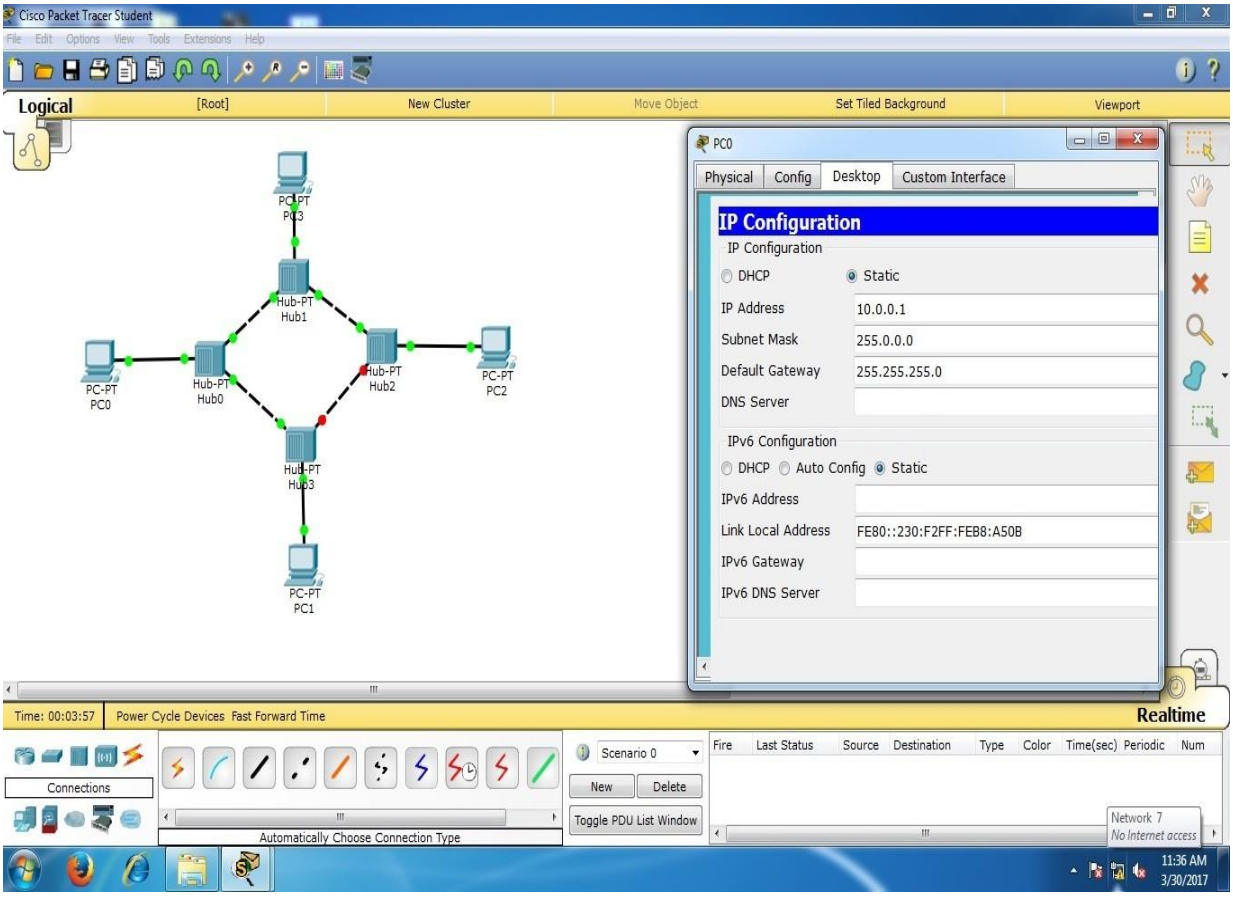
Step 2: Place more such devices to form a ring pattern and configure each of them with IP address.

Step 3: Connect each device with another device adjacent to it such that it form a ring pattern.

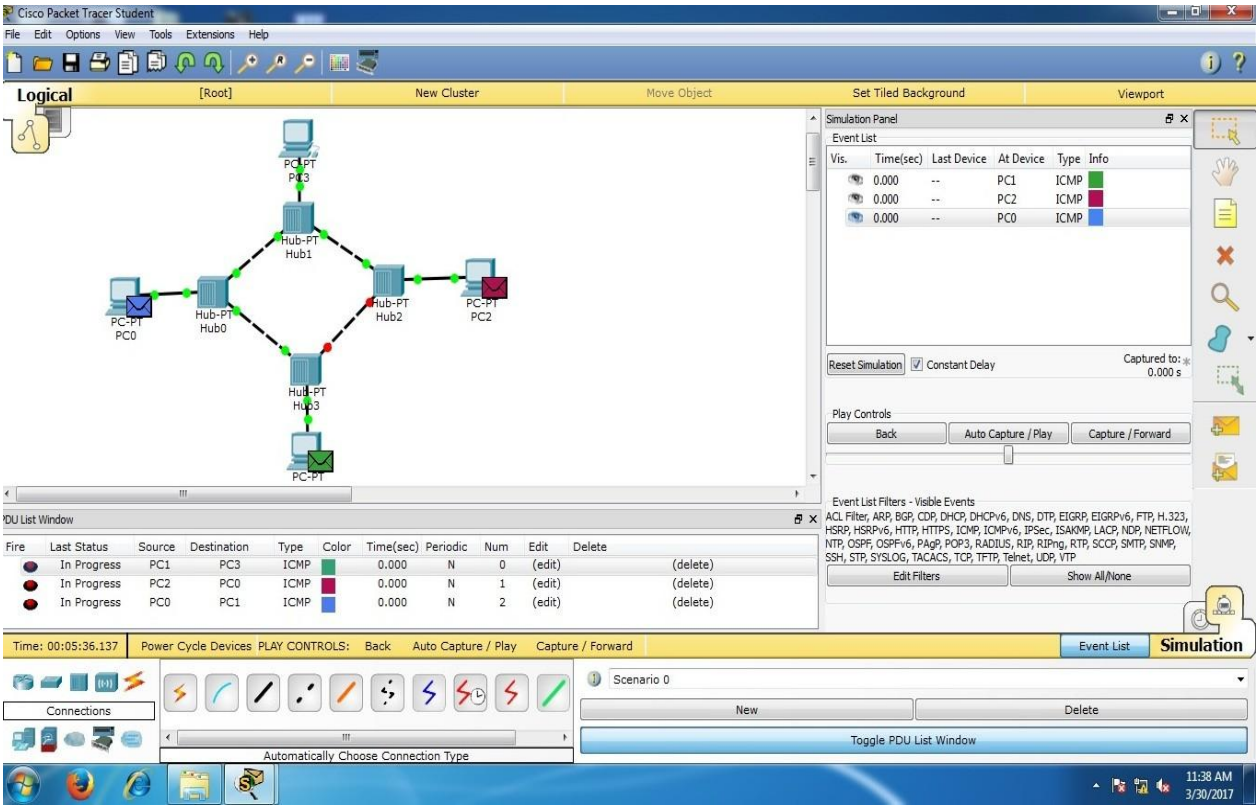
Step 4: Click the message packets from 'Add simple PDU' and select the receiver and the sender within the ring connected devices.

Step 5- Click on Simulation- Auto capture/ Play to observe the transfer of data in the network system.

<u>Design of Ring Topology:</u>



Simulation modal of Star Topology:



LAN:

A local area network (LAN) is a collection of devices connected together in one physical location, such as a building, office, or home. A LAN can be small or large, ranging from a home network with one user to an enterprise network with thousands of users and devices in an office or school.

A LAN comprises cables, access points, switches, routers, and other components that enable devices to connect to internal servers, web servers, and other LANs via wide area networks.

The rise of virtualization has also fueled the development of virtual LANs, which enable network administrators to logically group network nodes and partition their networks without a need for major infrastructure changes.

For example, in an office with multiple departments, such as accounting, IT support, and administration, each department's computers could be logically connected to the same switch but segmented to behave as if they are separate.

Algorithm:

Step1: Open the Cisco Packet Tracer. Click and drag the Switch and PCs as required. Connect all

the PCs to the Switch using the Copper Straight-Through Cable. Connect switches using the Copper Cross Over wire.

Step2: Assign the IP address , Subnet mask and default gateway to all the PCs

Click PC →Desktop → IP configuration → assign IP4 address , subnet mask.

PC0	10.10.10.1	255.0.0.0	255.255.255.0
PC1	10.10.10.2	255.0.0.0	255.255.255.0
PC2	10.10.10.3	255.0.0.0	255.255.255.0
PC3	10.10.10.4	255.0.0.0	255.255.255.0
PC4	10.10.10.5	255.0.0.0	255.255.255.0
PC5	10.10.10.6	255.0.0.0	255.255.255.0
PC6	10.10.10.7	255.0.0.0	255.255.255.0
PC7	10.10.10.8	255.0.0.0	255.255.255.0
PC8	10.10.10.9	255.0.0.0	255.255.255.0
PC9	10.10.10.10	255.0.0.0	255.255.255.0

Step 3: Drop the PDUs to source and destination and check whether the packets are send from source to destination.

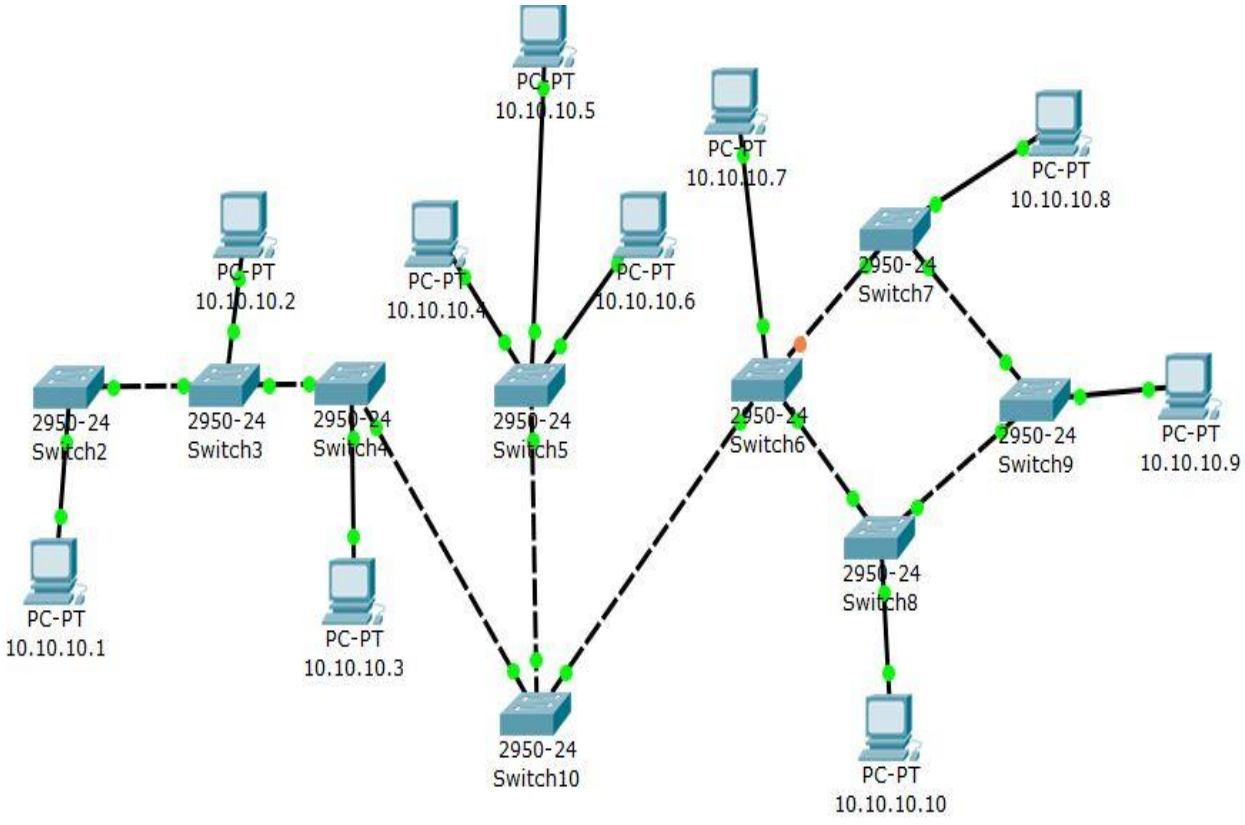
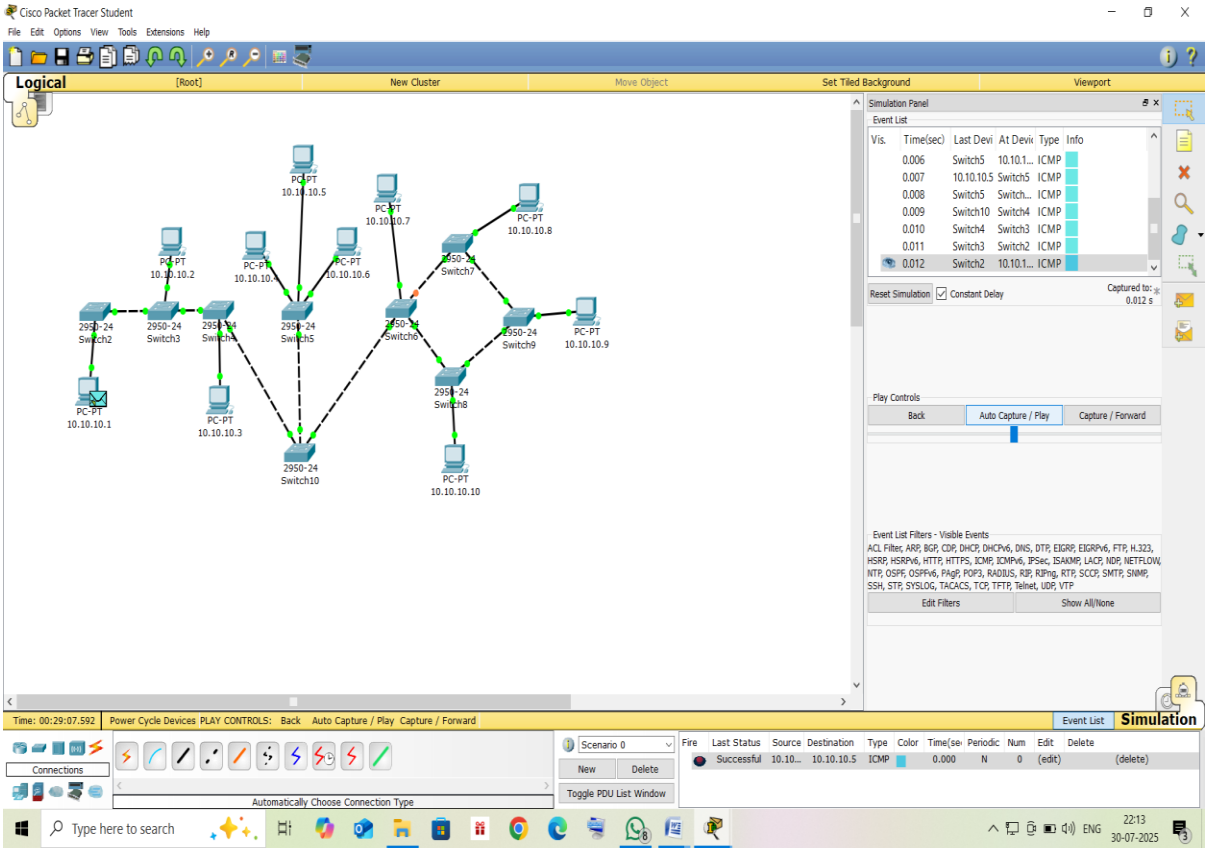


Fig: Design of Simple LAN using Switch

Output:



Algorithm:

Step1: Open the Cisco Packet Tracer. Click and drag the Switch and PCs as required. Connect all the PCs to the Switch using the Copper Straight-Through Cable. Connect switches and Hub using the Copper Cross Over wire.

Step2: Assign the IP address , Subnet mask and default gateway to all the PCs

Click PC →Desktop → IP configuration → assign IP4 address, subnet mask.

PC0	10.10.10.1	255.0.0.0	255.255.255.0
PC1	10.10.10.2	255.0.0.0	255.255.255.0
PC2	10.10.10.3	255.0.0.0	255.255.255.0
PC3	10.10.10.4	255.0.0.0	255.255.255.0
PC4	10.10.10.5	255.0.0.0	255.255.255.0
PC5	10.10.10.6	255.0.0.0	255.255.255.0
PC6	10.10.10.7	255.0.0.0	255.255.255.0
PC7	10.10.10.8	255.0.0.0	255.255.255.0
PC8	10.10.10.9	255.0.0.0	255.255.255.0
PC9	10.10.10.10	255.0.0.0	255.255.255.0

Step 3: Drop the PDUs to source and destination and check whether the packets are send from source to destination.

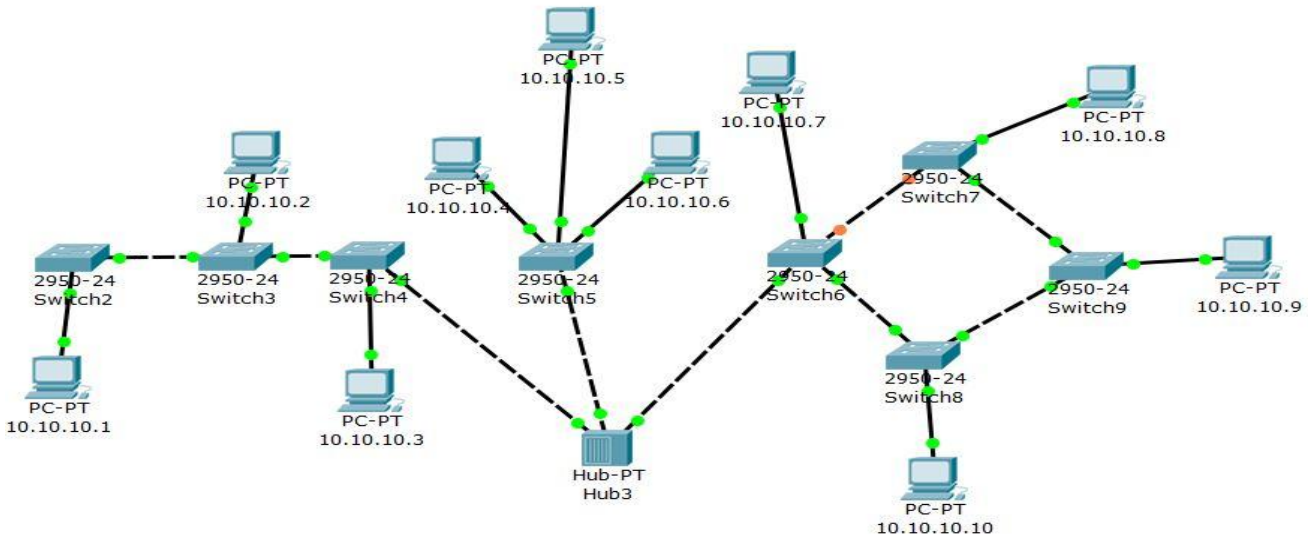
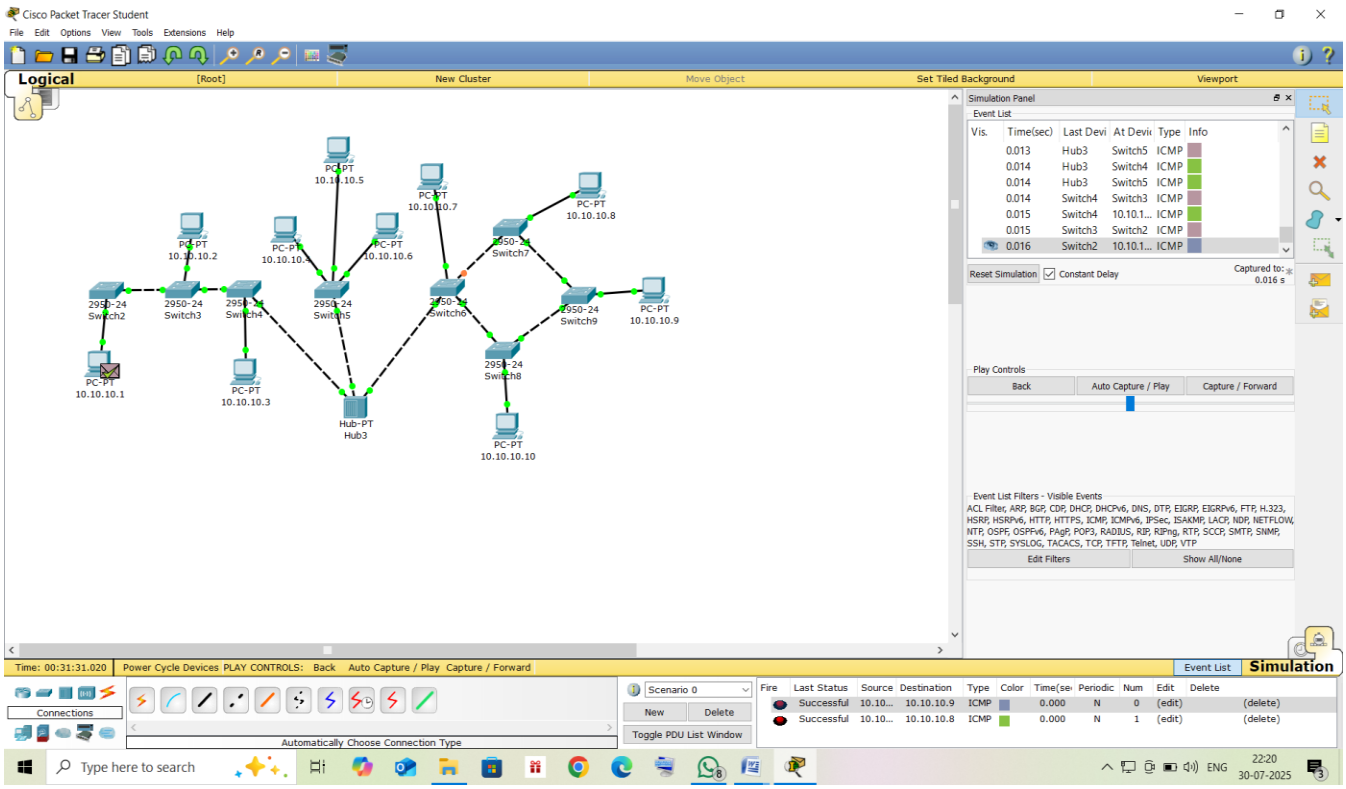


Fig: Design of Simple LAN using Hub

Output:



RESULT:

Thus the bus, star, ring topologies and simple LAN has been designed and executed successfully.

EXP.NO:2 **STUDY OF BASIC NETWORK COMPONENTS AND COMMANDS**

DATE:

Aim:

To learn the basic network components like Hub, Switch and basic network commands like NETSTAT, ipconfig, Ping, nslookup and Traceroute.

Cisco Packet Tracer:

Cisco Packet Tracer is a comprehensive networking simulation software tool for teaching and learning how to create network topologies and imitate modern computer networks. The tool offers a unique combination of realistic simulation and visualization experiences, assessment and activity authoring capabilities, and multi-user collaboration and competition opportunities.

HUB

A hub is a basic network device that connects multiple devices in a local area network (LAN) and broadcasts data to all connected devices, regardless of the intended recipient.

- Operates at the Physical Layer (Layer 1**)** of the OSI model.
- Has no intelligence, does not filter or forward data to specific devices.
- When one device sends data, the hub sends it to all other devices on the network.
- Increases the chance of collisions, making it less efficient than switches.

Algorithm:

Step1: Open the Cisco Packet Tracer. Click and drag the Hub and PCs as required. Connect all the PCs to the Hub using the Copper Straight-Through Cable.

Step2: Assign the IP address, Subnet mask and default gateway to all the PCs

Click PC →Desktop → IP configuration → assign IP4 address, subnet mask, default gateway

PC0	10.10.10.1	255.0.0.0	255.255.255.0
PC1	10.10.10.2	255.0.0.0	255.255.255.0
PC2	10.10.10.3	255.0.0.0	255.255.255.0
PC3	10.10.10.4	255.0.0.0	255.255.255.0
PC4	10.10.10.5	255.0.0.0	255.255.255.0

Step 3: Drop the PDUs to source and destination and check whether the packets are send from source to destination.

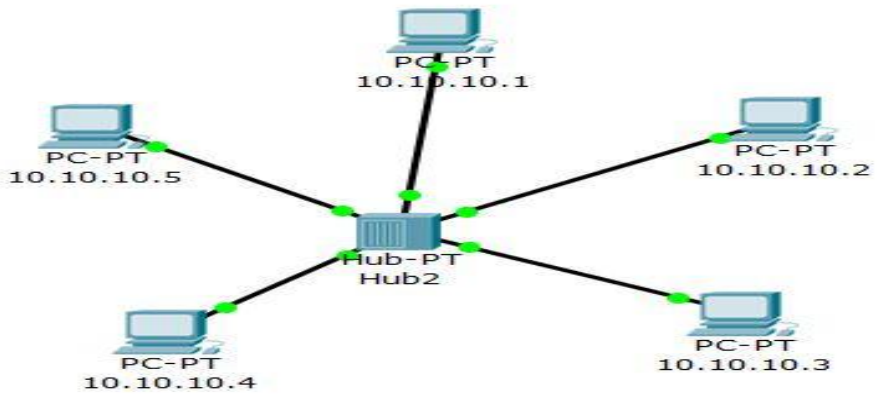
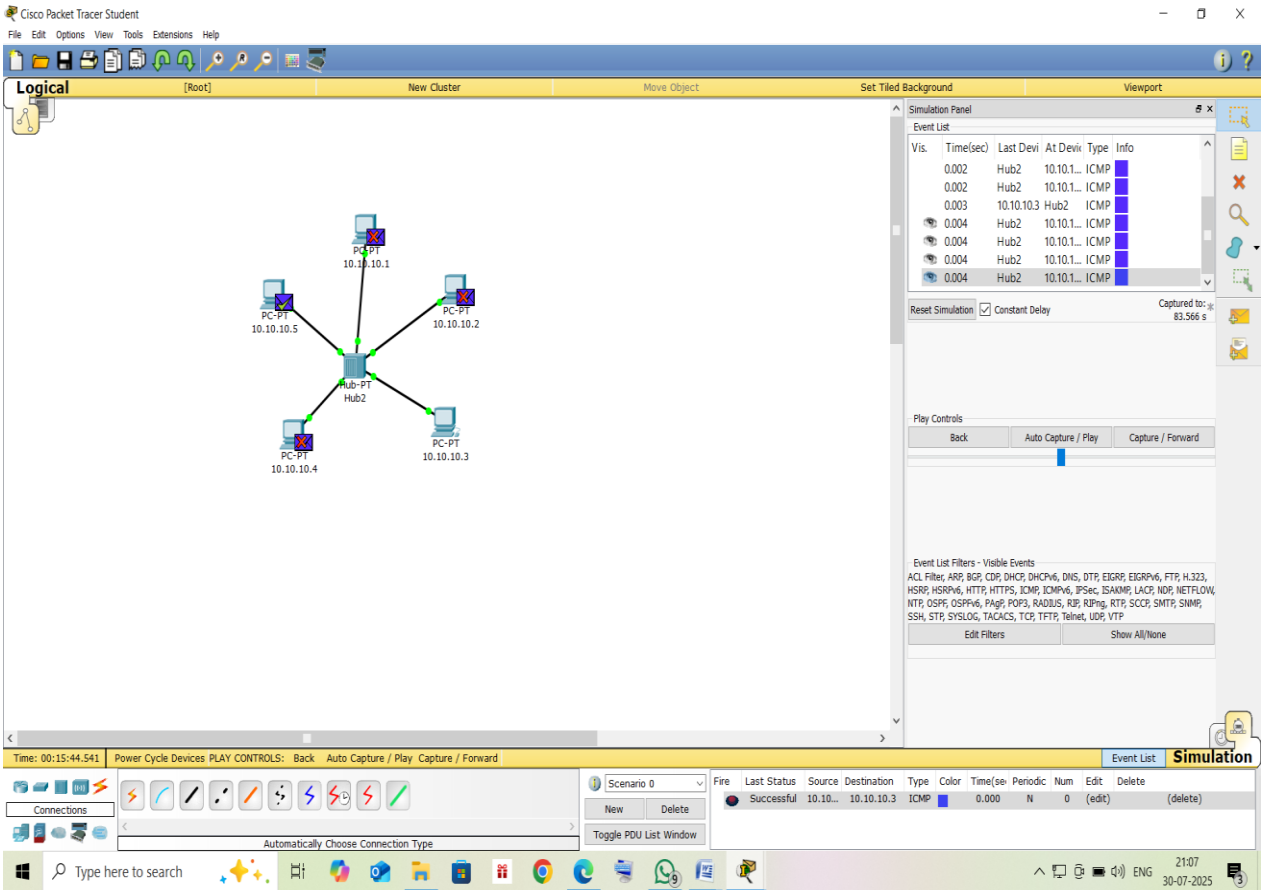


Fig: Design of Network using Hub

Output:



SWITCH

A network switch is a hardware device that connects devices within a local area network (LAN) and uses MAC addresses to forward data only to the specific device intended to receive it.

- Operates at the Data Link Layer (Layer 2) of the OSI model (some can also operate at Layer 3).
- Unlike hubs, switches do not broadcast data to all devices; they send it only to the correct destination.
- Maintains a MAC address table to map devices to their respective ports.
- Improves network efficiency and reduces traffic.

Algorithm:

Step1: Open the Cisco Packet Tracer. Click and drag the Switch and PCs as required. Connect all the PCs to the Switch using the Copper Straight-Through Cable.

Step2: Assign the IP address, Subnet mask and default gateway to all the PCs

Click PC →Desktop → IP configuration → assign IP4 address , subnet mask.

PC0	10.10.10.1	255.0.0.0	255.255.255.0
PC1	10.10.10.2	255.0.0.0	255.255.255.0
PC2	10.10.10.3	255.0.0.0	255.255.255.0
PC3	10.10.10.4	255.0.0.0	255.255.255.0
PC4	10.10.10.5	255.0.0.0	255.255.255.0

Step 3: Drop the PDUs to source and destination and check whether the packets are send from source to destination.

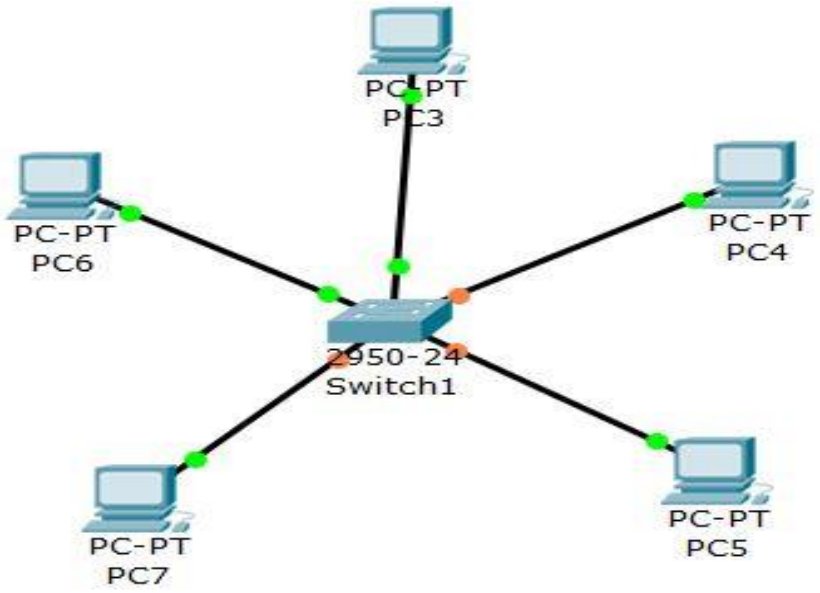
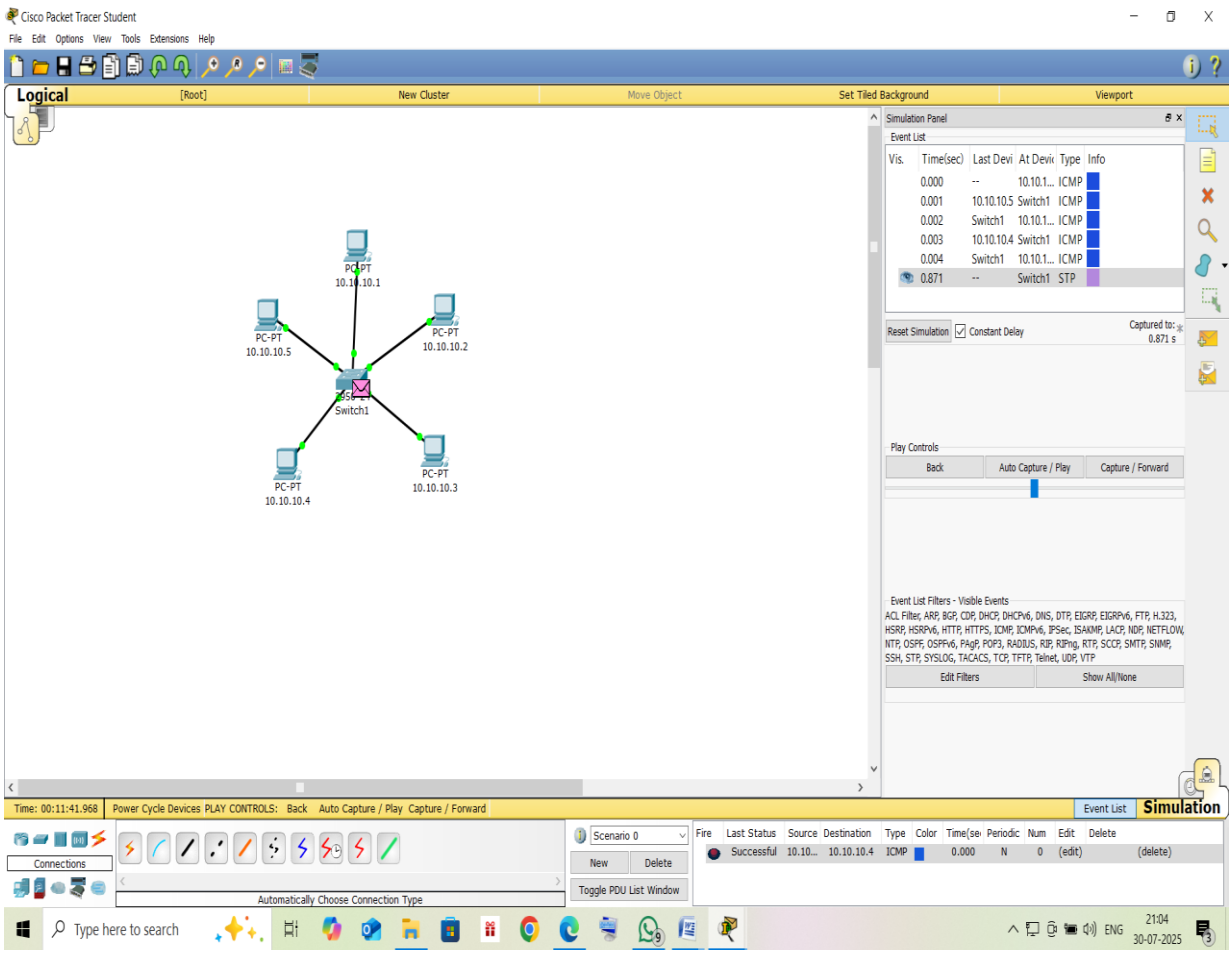


Fig: Design of Network using Switch

Output:

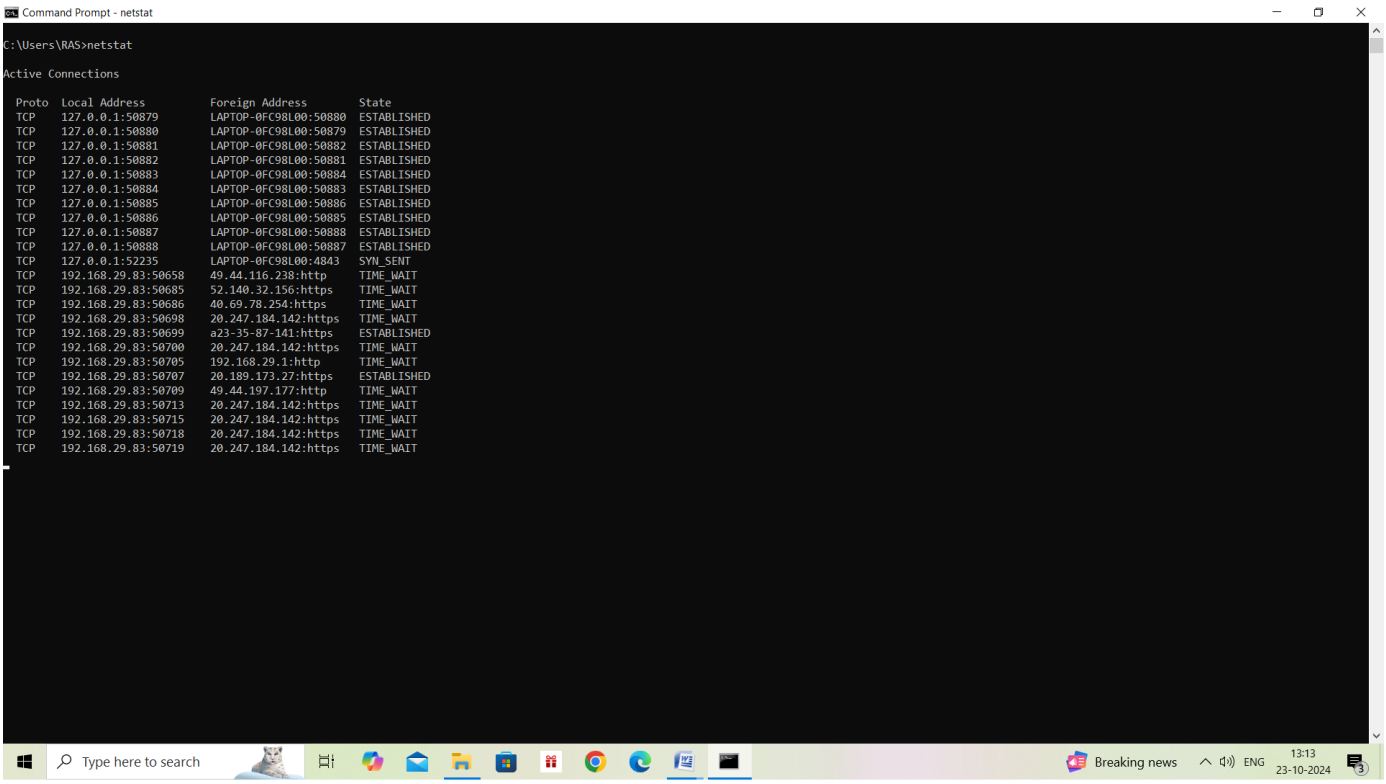


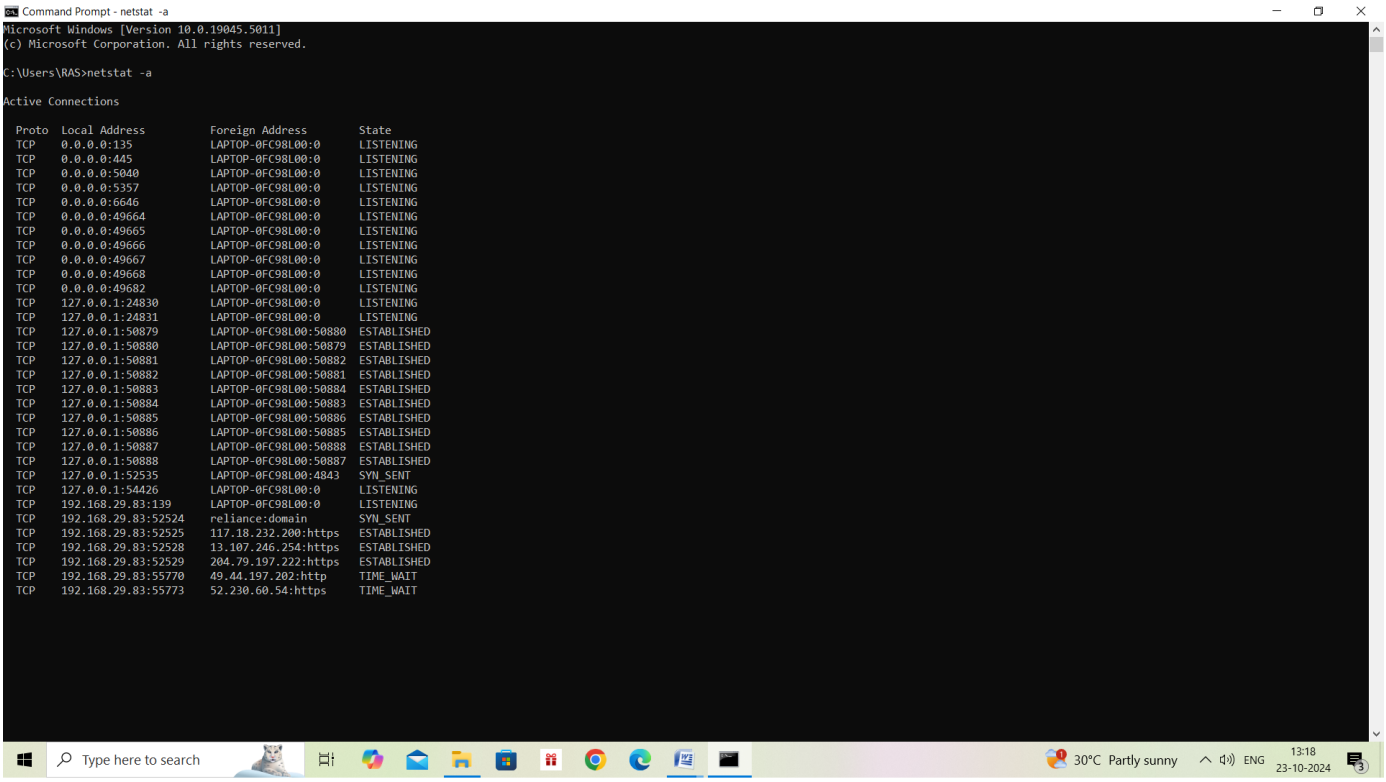
1. NETSTAT

Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems. Net stat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen (analogous to a Linux or UNIX) for netstat reads as follows:

Displays protocol statistics and current TCP/IP network connections. NETSTAT -a - b -e -n -o -p proto -r -s -v interval

-a	Displays all connections and listening ports.
-b	Displays the executable involved in creating each connection or listening port. In some cases well- knownexecutableshostmultipleindependentcomponents,andinthesecases the sequence of components involved in creating the connection or listening port is displayed.Inthiscasetheexecutablenameisin[]atthebottom,ontopisthecomponent it called, and so forth until TCP/IP was reached. Note that this option can be time-consuming and will fail unless you have sufficient permissions.
-e	Displays Ethernet statistics. This may be combined with the -s option.
-n	Displays addresses and port numbers in numerical form.
-o	Displays the owning process ID associated with each connection.
-p proto	Showsconnectionsfortheprotocolspecifiedbyproto;protomaybeanyof:TCP,UDP, TCPv6, or UDPv6. If used with the -s option to display per-protocol statistics, proto may be any of: IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, or UDPv6.
-r	Displays the routing table.
-s	Displays per-protocol statistics. By default, statistics are shown for IP, IPv6, ICMP, ICMPv6,TCP,TCPv6, UDP, and UDPv6;the-poptionmaybeusedtospecifyasubset of the default.
-v	When used in conjunction with -b, will display sequence of components involved in creating the connection or listening port for all executables.
interval	Redisplays selected statistics, pausing interval seconds between each display. Press CTRL+C to stop redisplaying statistics. If omitted, netstat will print the current Configuration information once.





2.IPCONFIG

In Windows, ipconfig is a console application designed to run from the Windows command prompt. This utility allows you to get the IP address information of a Windows computer. It also allows

some control over active TCP/IP connections. Ipconfig replaced the older win ipcfg utility.

From the command prompt, type **ipconfig** to run the utility with default options. The output of the default command contains the IP address, network mask, and gateway for all physical and virtual network adapter Syntax

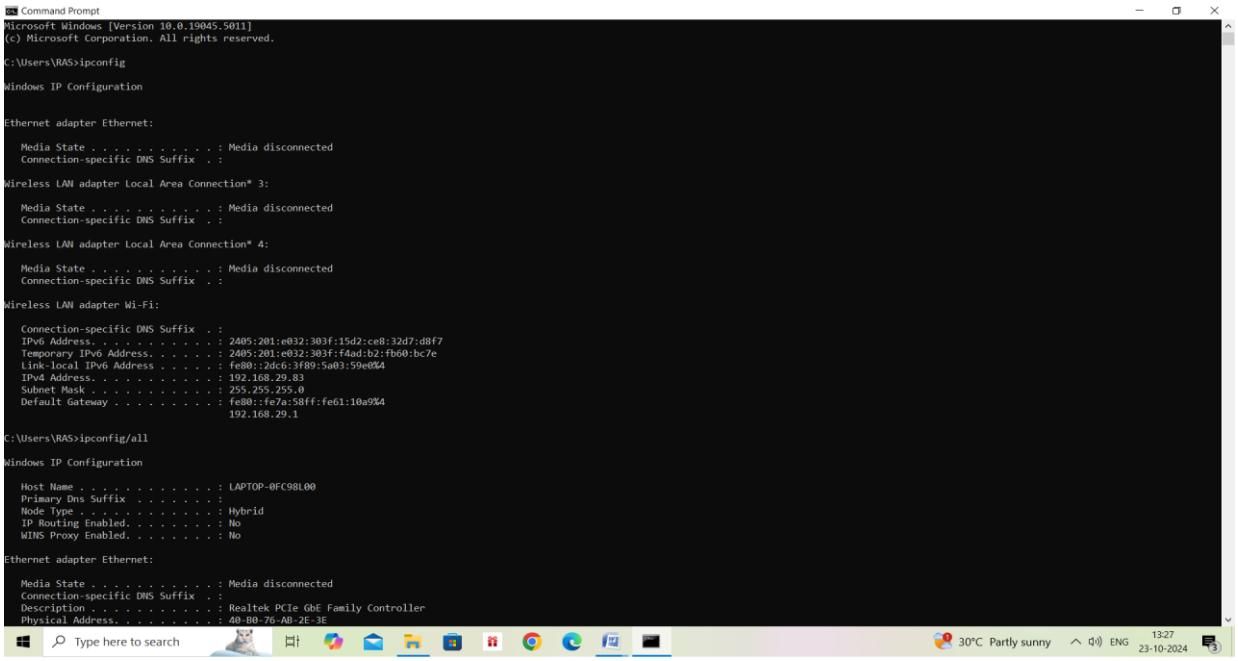
ipconfig [/all] [/renew [Adapter]] [/release [Adapter]] [/flushdns] [/displaydns] [/registerdns] [/showclassid Adapter] [/setclassid Adapter [ClassID]]

Parameters

Used without parameters	displays the IP address, subnet mask, and default gateway for all adapters.
/all	Displays full tcp/ip configurations
/renew [Adapter]	Renews DHCP configuration for all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically. To specify an adapter name, type the adapter name that appears when you use ipconfig without parameters.
/release [Adapter]	Sends a DHCPRELEASE message to the DHCP server to release the current DHCP configuration and discard the IP address configuration for either all adapters (if an adapter is not specified) or for a specific adapter if the Adapter parameter is included. This parameter disables TCP/IP for adapters configured to obtain an IP address automatically. To specify an adapter name, type the adapter name that appears when you use ipconfig without parameters.
/flushdns	Flushes and resets the contents of the DNS client resolver cache. During DNS troubleshooting, you can use this procedure to discard negative cache entries from the cache, as well as any other entries that have been added dynamically.

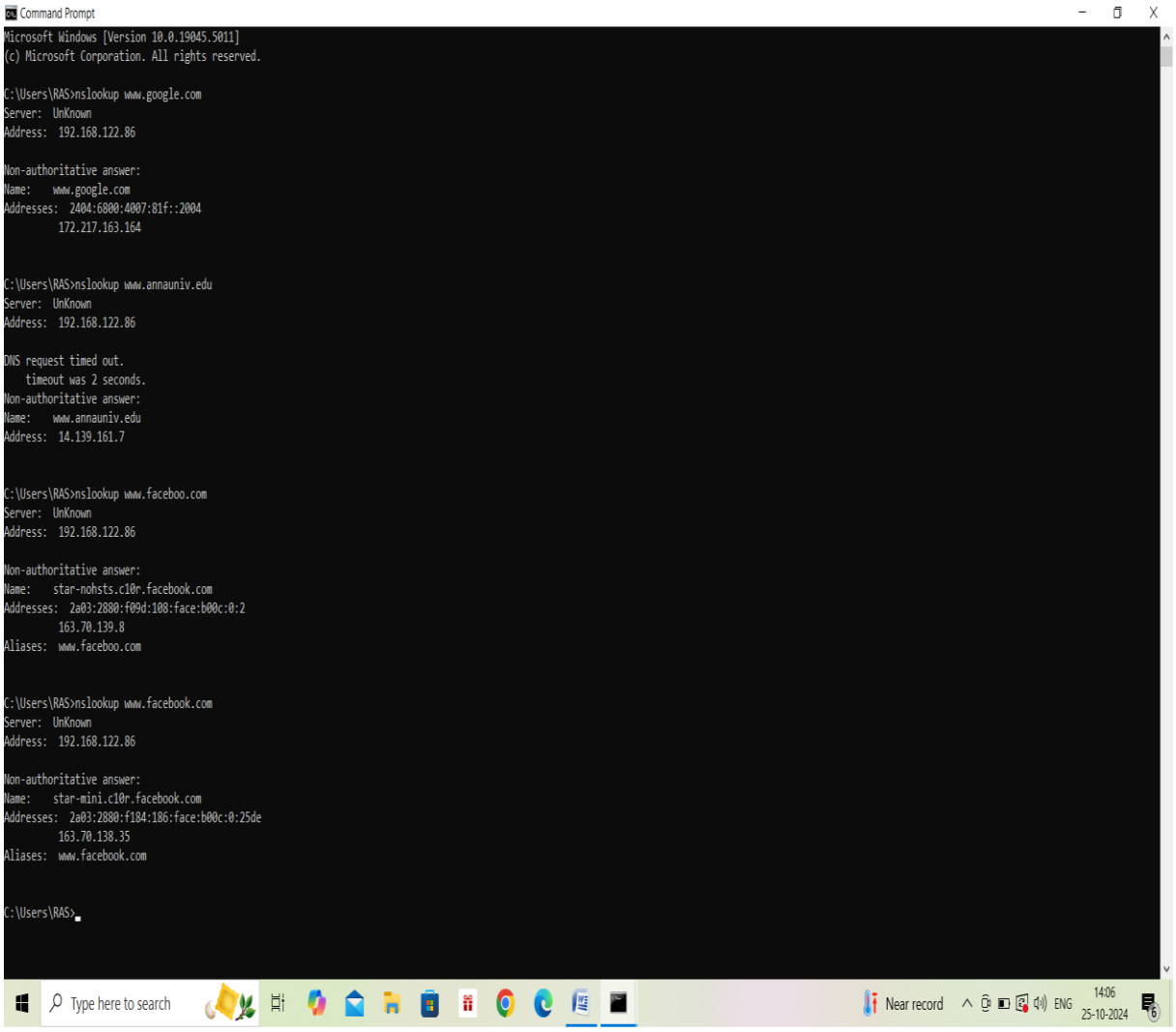
/displaydns	Displays the contents of the DNS client resolver cache, which includes both entries preloaded from the local Hosts file and any recently obtained resource records for name queries resolved by the computer. The DNS Client service uses this information to resolve frequently queried names quickly, before querying its configured DNS servers.

/registerdns	Initiates manual dynamic registration for the DNS names and IP addresses that are configured at a computer. You can use this parameter to troubleshoot a failed DNS name registration or resolve a dynamic update problem between a client and the DNS server without rebooting the client computer. The DNS settings in the advanced properties of the TCP/IP protocol determine which names are registered in DNS.
/showclassid	Adapter Displays the DHCP class ID for a specified adapter. To see the DHCP class ID for all adapters, use the asterisk (*) wildcard character in place of Adapter. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically.
/setclassid	Adapter [ClassID] Configures the DHCP class ID for a specified adapter. To set the DHCP class ID for all adapters, use the asterisk (*) wildcard character in place of Adapter. This parameter is available only on computers with adapters that are configured to obtain an IP address automatically. If a DHCP class IDis not specified, the current class ID is removed.



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.

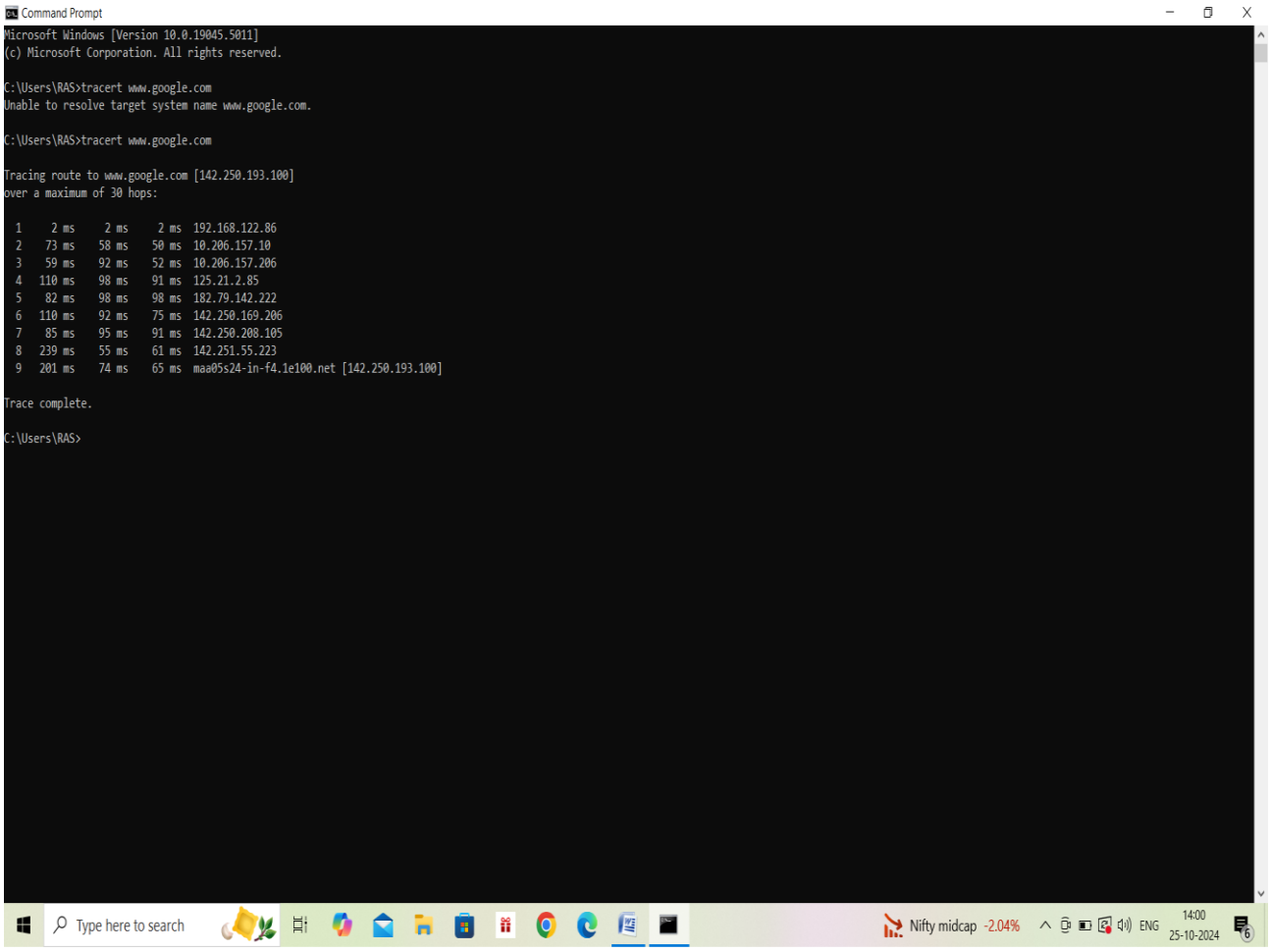


4. TRACE ROUTE

Trace route is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Trace route also records the time taken for each hop the packetmakesduringitsroutetothedestination.TracerouteusesInternetControl Message Protocol

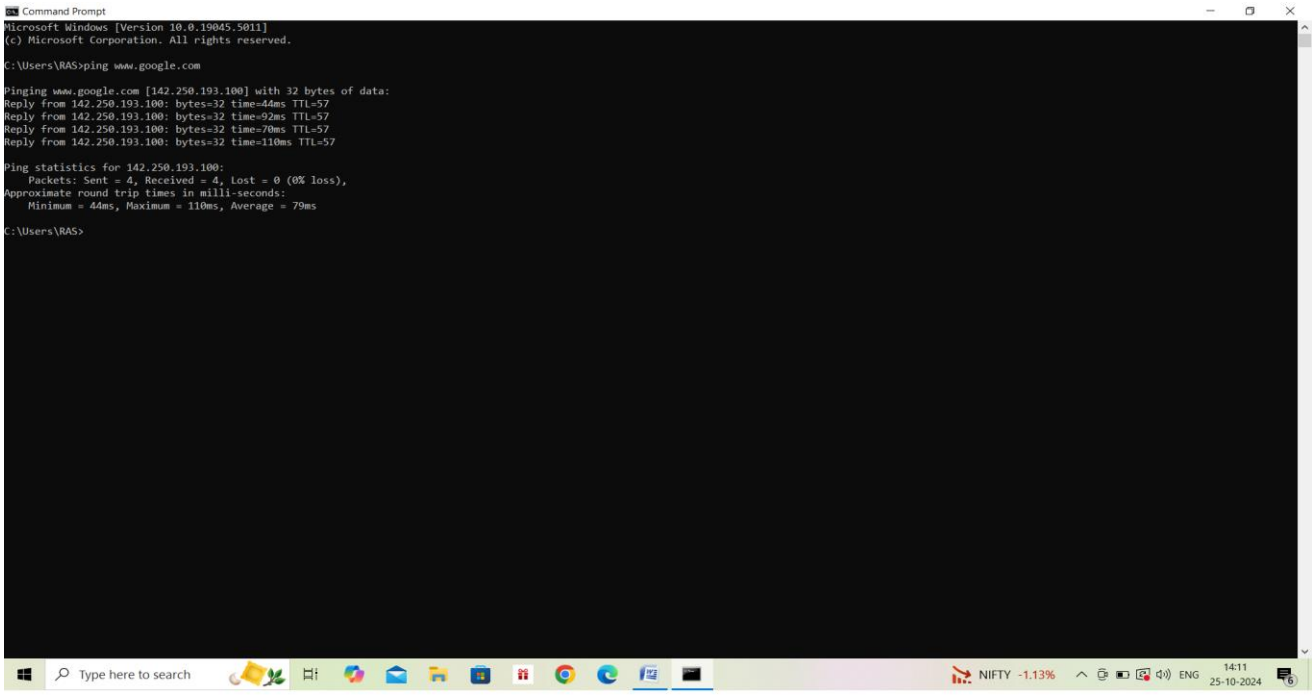
(ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

With the tracert command shown above, we're asking tracert to show us the path from the local computer all the way to the network device with the hostname www.google.com.



5.PING

The ping command in Windows is used to test network connectivity and diagnose issues with reachability, connectivity, and name resolution.



Result:

Thus the basic network components like Hub, Switch and basic network commands like NETSTAT, ipconfig, Ping, nslookup and Traceroute are studied successfully.

EXP.NO:3A ECHO CLIENT AND ECHO SERVER USING TCP SOCKETS
DATE:

Aim:

To write a program in Java to implement an applications using TCP Sockets like echo client and echo server

Algorithm

1. Start the Program
2. In Server
    a) Create a server socket and bind it to port.
    b) Listen for new connection and when a connection arrives, accept it.
    c) Read the data from client.
    d) Echo the data back to the client.
    e) Close all streams.
    f) Close the server socket.
    g) Stop.
3. In Client
    a) Create a client socket and connect it to the server's port number.
    b) Send user data to the server.
    c) Display the data echoed by the server.
    d) Close the input and output streams.
    e) Close the client socket.
    f) Stop.
4. Stop the program

Program

EchoServer

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
```

```
import java.net.Socket;
public class EchoServer
{
public static void main(String[] args) throws Exception {
ServerSocket serverSocket = null;
try {
serverSocket = new ServerSocket(10007);
} catch (IOException e) {
System.err.println("Could not listen on port: 10007.");
System.exit(1);
}
Socket clientSocket = null;
System.out.println("Waiting for connection.....");
try {
clientSocket = serverSocket.accept();
} catch (IOException e) {
System.err.println("Accept failed.");
System.exit(1);
}
System.out.println("Connection successful");
System.out.println("Waiting for input.....");

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
String inputLine;
while ((inputLine = in.readLine()) != null) {
System.out.println("Server: " + inputLine);
out.println(inputLine);
if (inputLine.equals("Bye."))
break;
```

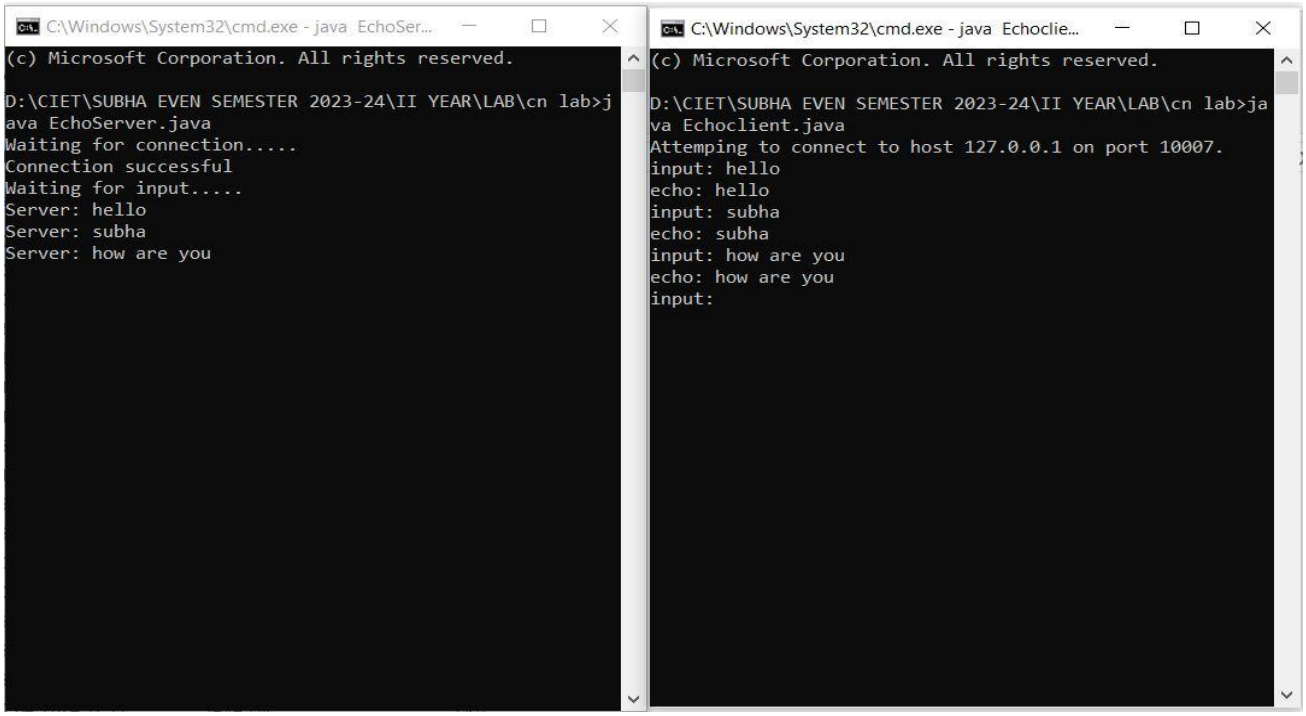
```
}
out.close();
in.close();
clientSocket.close();
serverSocket.close();
}
}
```

EchoClient

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.net.UnknownHostException;

public class EchoClient
 {
public static void main(String[] args) throws Exception {
String serverHostname = new String("127.0.0.1");
if (args.length > 0)
serverHostname = args[0];
System.out.println("Attemping to connect to host " + serverHostname + " on port 10007.");
Socket echoSocket = null;
PrintWriter out = null;
BufferedReader in = null;
try
{
// echoSocket = new Socket("taranis", 7);
echoSocket = new Socket(serverHostname, 10007);
out = new PrintWriter(echoSocket.getOutputStream(), true);
```

```
in = new BufferedReader(new InputStreamReader(echoSocket.getInputStream()));
} catch (UnknownHostException e) {
System.err.println("Don't know about host: " + serverHostname);
System.exit(1);
} catch (IOException e) {
System.err.println("Couldn't get I/O for " + "the connection to: " + serverHostname);
System.exit(1);
}
BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
String userInput;
System.out.print("input: ");
while (!(userInput = stdIn.readLine()).equals("bye")) {
out.println(userInput);
System.out.println("echo: " + in.readLine());
System.out.print("input: ");
}

out.close();
in.close();
stdIn.close();
echoSocket.close();
}
}
```

Output:



Result:

Thus a program in Java implemented an applications using TCP Sockets like echo client and echo server.

EXP.NO:3B CHAT CLIENT AND CHAT SERVER USING TCP SOCKETS

DATE:

Aim

To write a program in Java to implement an applications using TCP Sockets like chat.

Algorithm

1. Start the Program
2. In Server
   a) Create a server socket and bind it to port.
   b) Listen for new connection and when a connection arrives, accept it.
   c) Read Client's message and display it
   d) Get a message from user and send it to client
   e) Repeat steps 3-4 until the client sends "end"
   f) Close all streams
   g) Close the server and client socket
   h) Stop
3. In Client
   a) Create a client socket and connect it to the server's port number
   b) Get a message from user and send it to server
   c) Read server's response and display it
   d) Repeat steps 2-3 until chat is terminated with "end" message
   e) Close all input/output streams
   f) Close the client socket
   g) Stop
4. Stop the program

Program

ChatServer

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

```
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;
public class ChatServer {

public static void main(String[] args) throws IOException {
ServerSocket server = null; // server socket for handling server operation
try {
server = new ServerSocket(90); // establish connection port
} catch (IOException ioEx) {
System.out.println("Error in port 90");
System.exit(1);
}

Socket serverSocket = null; // handle client request
try {
System.out.println("Waiting for client!");
serverSocket = server.accept(); // listen to the client request
System.out.println("connection accepted at " + serverSocket);
} catch (IOException ioEx) {
System.out.println("Failed to Connect");
System.exit(1);
}
// output to the client
DataOutputStream frmServer = new DataOutputStream(serverSocket.getOutputStream());
// get input from client
BufferedReader toServer = new BufferedReader(new
InputStreamReader(serverSocket.getInputStream()));
// get input from keyboard;
BufferedReader kybd = new BufferedReader(new InputStreamReader(System.in));
String clientMsg, serverMsg;
```

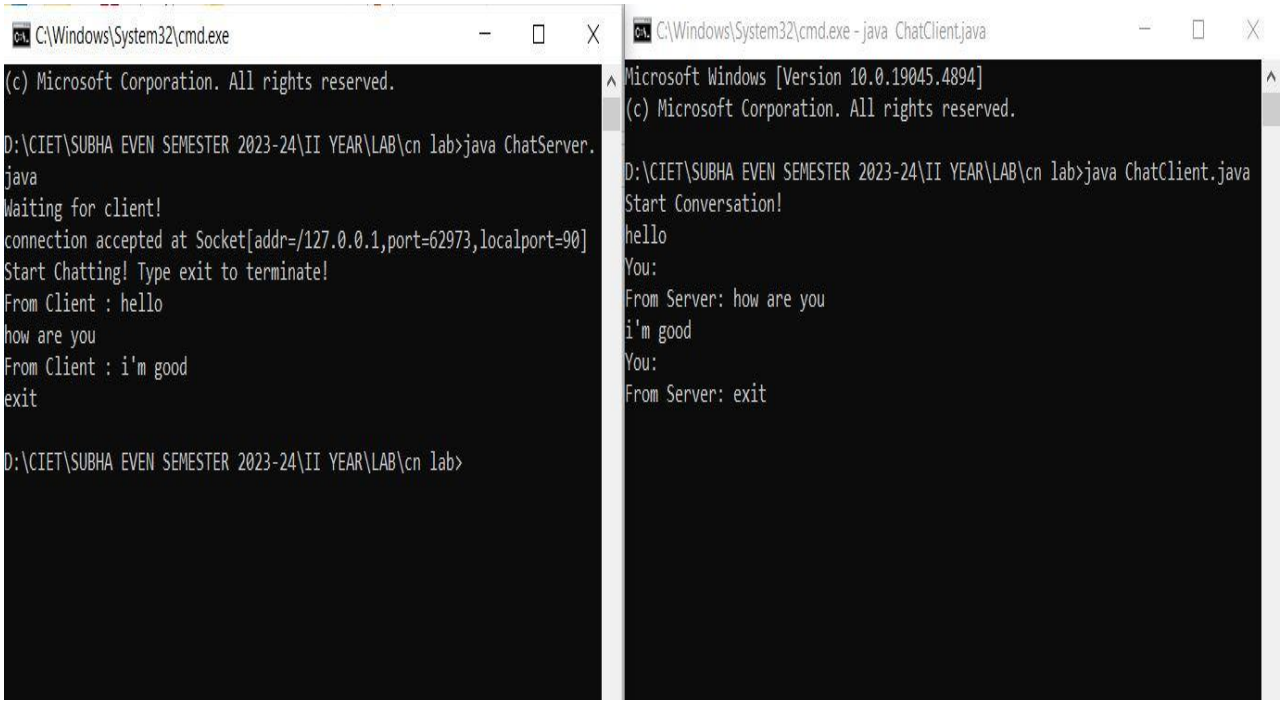
```
System.out.println("Start Chatting! Type exit to terminate!");
boolean end = false;
try {
do {
clientMsg = toServer.readLine(); // accept client's message;
System.out.println("From Client : " + clientMsg);
// int len = msgClient.length();
// accept msg and send to client
serverMsg = kybd.readLine();
frmServer.writeBytes(serverMsg);
frmServer.write(13);
frmServer.write(10);
frmServer.flush();
if (serverMsg.equals("exit")) {
end = true;
}
} while (!end);
}
catch (Exception e)
{
System.out.println("Exiting the chat....");
}
try
{
frmServer.close();server.close();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

ChatClient

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.InetAddress;
import java.net.Socket;
import java.net.UnknownHostException;

public class ChatClient {

public static void main(String[] args) throws IOException {
Socket client = null;
BufferedReader toClient = null;
DataOutputStream frmClient = null;
BufferedReader kybd = new BufferedReader(new InputStreamReader(System.in));

try {
client = new Socket(InetAddress.getLocalHost(), 90); // send request to server;

toClient = new BufferedReader(new InputStreamReader(client.getInputStream()));
frmClient = new DataOutputStream(client.getOutputStream());
} catch (UnknownHostException unknwnEx) {
System.out.println("Server not Found");
System.exit(1);
}
System.out.println("Start Conversation!");
boolean end = false;
String clientMsg, serverMsg;
try {
do {
// read message from the keyboard;
clientMsg = kybd.readLine();
// send to server;
```

```
frmClient.writeBytes(clientMsg);
frmClient.write(13);
frmClient.write(10);
frmClient.flush();

// receive message from server
System.out.println("You: ");
serverMsg = toClient.readLine();
System.out.println("From Server: " + serverMsg);
if (clientMsg.equals("exit")) {
end = true;

}
} while (!end);
}
catch(Exception e) {
System.out.println("Exiting the chat....");
}
try {
toClient.close();
frmClient.close();
client.close();

}
catch(Exception e) {

System.out.println("Something wrong...");
}
}
```

Output:



Result:

Thus a program in Java implemented an application using TCP Sockets like chat client and chat server.

EXP.NO:4 **SIMULATIONS OF DNS USING UDP SOCKETS**

DATE:

Aim:

To write a program in Java to perform Simulation of DNS using UDP sockets.

Algorithm

1. Start the Program
2. In Server
   a) Create an array of hosts and its ip address in another array
   b) Create a datagram socket and bind it to a port
   c) Create a datagram packet to receive client request
   d) Read the domain name from client to be resolved
   e) Lookup the host array for the domain name
   f) If found then retrieve corresponding address
   g) Create a datagram packet and send ip address to client
   h) Repeat steps 3-7 to resolve further requests from clients
   i) Close the server socket
   j) Stop
3. In Client
   a) Create a datagram socket
   b) Get domain name from user
   c) Create a datagram packet and send domain name to the server
   d) Create a datagram packet to receive server message
   e) Read server's response
   f) If ip address then display it else display "Domain does not exist"
   g) Close the client socket
   h) Stop
4. Stop the program

Program:

UDPServer

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

```
import java.net.*;
public class UDPServer
{
private static int indexOf(String[] array, String str)
{
str = str.trim();
for (int i=0; i < array.length; i++)
{
if (array[i].equals(str)) return i;
}
return -1;
}
public static void main(String arg[])throws IOException
{
String[] hosts = {"zoho.com", "gmail.com","google.com", "facebook.com"};
String[] ip = {"172.28.251.59", "172.217.11.5","172.217.11.14","31.13.71.36"};
System.out.println("Press Ctrl + C to Quit");
while (true)
{
DatagramSocket serversocket=new DatagramSocket(1362);
byte[] senddata = new byte[1021];
byte[] receivedata = new byte[1021];
DatagramPacket recvpack = new DatagramPacket(receivedata,
receivedata.length);
serversocket.receive(recvpack);
```

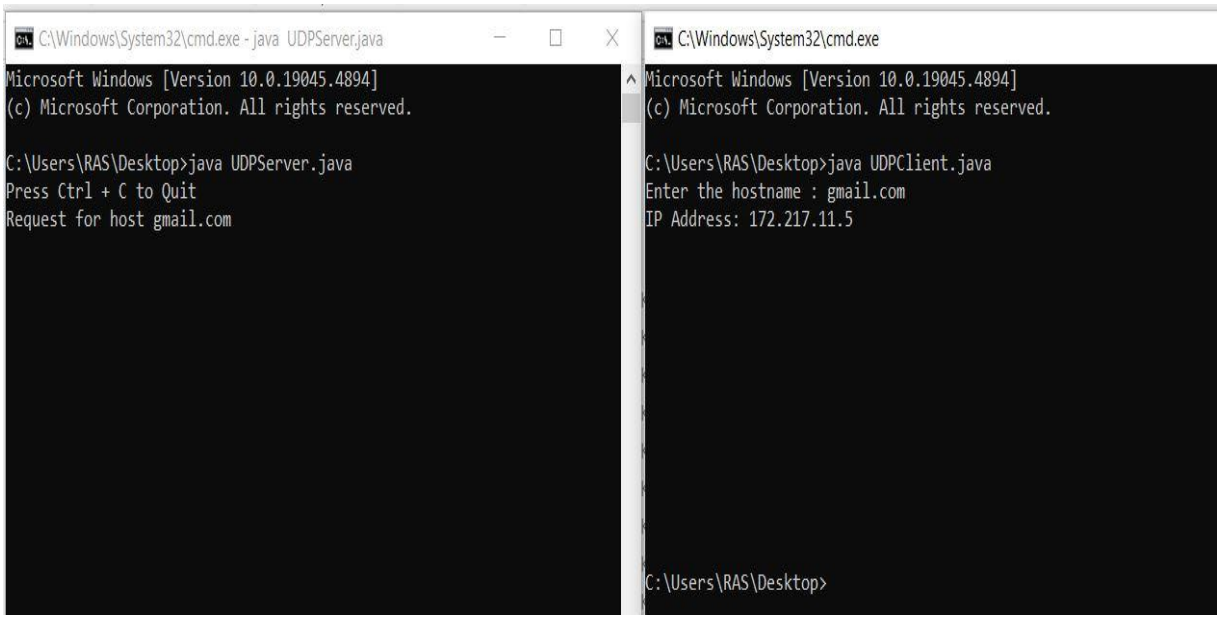
```
        String sen = new String(recvpack.getData());
        InetAddress ipaddress = recvpack.getAddress();
        int port = recvpack.getPort();
        String capsent;
        System.out.println("Request for host " + sen);
        if(indexOf (hosts, sen) != -1)
        capsent = ip[indexOf (hosts, sen)];
        else
        capsent = "Host Not Found"; senddata = capsent.getBytes();
      DatagramPacket pack = new DatagramPacket (senddata,senddata.length,ipaddress,port);
        serversocket.send(pack);
        serversocket.close();
        }
        }
}
```

UDPClient

```
        import java.io.*;
        import java.net.*;
        public class UDPClient
        {
        public static void main(String args[])throws IOException
        {
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        DatagramSocket clientsocket = new DatagramSocket();
```

```
        InetAddress ipaddress;
        if (args.length == 0)
        ipaddress = InetAddress.getLocalHost();
        else
        ipaddress = InetAddress.getByName(args[0]);
        byte[] senddata = new byte[1024];
        byte[] receivedata = new byte[1024];
        int portaddr = 1362;
        System.out.print("Enter the hostname : ");
        String sentence = br.readLine();
        senddata = sentence.getBytes();
        DatagramPacket pack = new
        DatagramPacket(senddata,senddata.length,ipaddress,portaddr);
        clientsocket.send(pack);
        DatagramPacket recvpack =new
        DatagramPacket(receivedata,receivedata.length);
        clientsocket.receive(recvpack);
        String modified = new String(recvpack.getData());
        System.out.println("IP Address: " + modified);
        clientsocket.close();
        }
}
```

Output:



Result:

Thus a program in Java performed Simulation of DNS using UDP sockets.

EXP.NO:5 A

IMPLEMENTATION OF STOP-AND-WAIT PROTOCOL USING TCP SOCKET

DATE:

Aim:

To write a program in Java to implement stop-and-wait protocol using TCP socket

Algorithm

Server

1. Start.
2. Create a ServerSocket on a specific port (e.g., 5000).
3. Wait for a client connection using `accept()`.
4. Once the connection is established:
   Create input and output streams for communication.
5. Repeat the following steps until termination:
   a. Receive a frame from the client.
   b. If the frame is `"exit"`, then
   → Display message and break the loop.
   c. Display the received frame to the user.
   d. Ask the user:
   "Accept this frame? (yes/no)"
   e. If the user enters yes:
   → Send `"ACK"` (Acknowledgment) to client.
   f. Else (user enters no):
   → Send `"NACK"` (Negative Acknowledgment) to client.
   g. Go back to step (a).
6. Close all connections and sockets.
7. Stop.

Client

1. Start.
2. Establish a connection to the server socket (e.g., on port 5000).
3. Create input and output streams for communication.
4. Ask the user how many frames to send (n).
5. For each frame i from 1 to n, do:
   a. Create a frame name (e.g., "Frame–i").
   b. Set a flag acknowledged = false.
   c. While acknowledged is false:
   i. Send the frame to the server.
   ii. Wait to receive a response from the server.
   iii. If response is "ACK":
   - Display message: ACK received.
   - Set acknowledged = true (proceed to next frame).

iv. Else if response is "NACK":
- Display message: NACK received.
- Resend the same frame.

6. After all frames are sent successfully, send "exit" message to the server.
7. Close all connections and sockets.
8. Stop.

Program:

Server

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class StopAndWaitServer {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(5000);
        System.out.println("Server started. Waiting for client...");
        Socket socket = serverSocket.accept();
        System.out.println("Client connected.");
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        Scanner sc = new Scanner(System.in);
        String frame;
        while (true) {
            frame = in.readLine();
            if (frame.equalsIgnoreCase("exit")) {
                System.out.println("Client ended the transmission.");
                break;
            }
            System.out.println("Received Frame: " + frame);
            System.out.print("Accept this frame? (yes/no): ");
            String choice = sc.nextLine().trim().toLowerCase();
            if (choice.equals("yes")) {
                out.println("ACK");
                System.out.println("ACK sent.\n");
            } else {
```

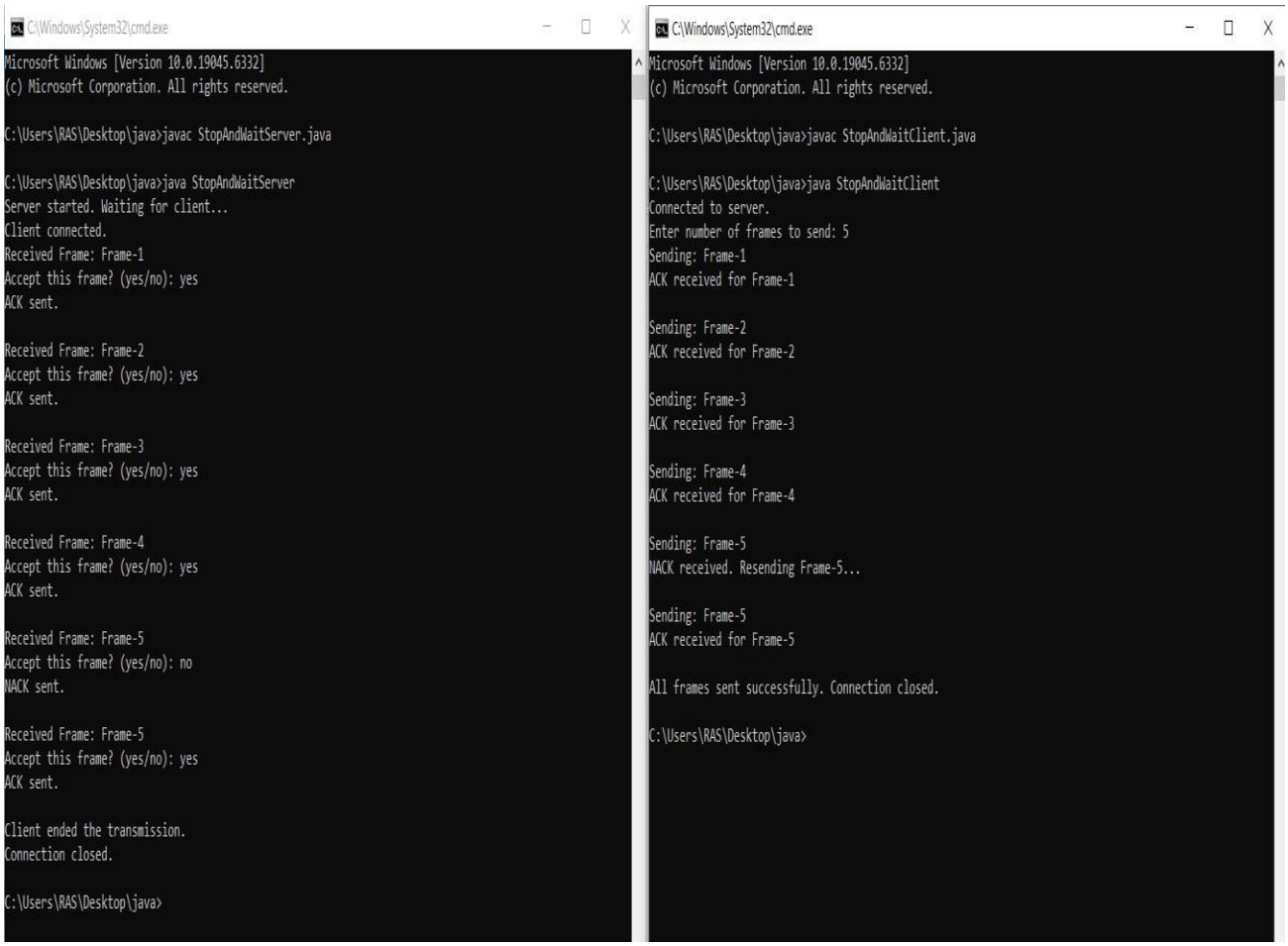
```
                out.println("NACK");
                System.out.println("NACK sent.\n");
            }
        }
        socket.close();
        serverSocket.close();
        sc.close();
        System.out.println("Connection closed.");
    }
}
```

Client

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class StopAndWaitClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        Scanner sc = new Scanner(System.in);
        System.out.println("Connected to server.");
        System.out.print("Enter number of frames to send: ");
        int n = sc.nextInt();
        sc.nextLine(); // consume newline
        for (int i = 1; i <= n; i++) {
            String frame = "Frame-" + i;
            boolean acknowledged = false;
```

```
                while (!acknowledged) {
                    System.out.println("Sending: " + frame);
                    out.println(frame);
                    String response = in.readLine();
                    if (response.equalsIgnoreCase("ACK")) {
                        System.out.println("ACK received for " + frame + "\n");
                        acknowledged = true;
                    } else {
                        System.out.println("NACK received. Resending " + frame + "...\n");
                    }
                }
            }
            out.println("exit");
            socket.close();
            sc.close();
            System.out.println("All frames sent successfully. Connection closed.");
        }
    }
```

Output:



Result:

Thus a program in Java to implement stop-and-wait protocol using TCP socket was executed successfully.

EXP.NO:5 B

IMPLEMENTATION OF SLIDING WINDOW PROTOCOL USING TCP SOCKET

DATE:

Aim:

To write a program in Java to implement sliding window protocol using TCP socket

Algorithm

Server:

1. Start.
2. Create a ServerSocket on a specific port (e.g., 5000).
3. Wait for a client connection using `accept()`.
4. Once the connection is established:
5. Repeat the following steps until termination:
   a. Receive a frame from the client.
   b. If the frame is `"exit"`, then
   → Display message "Client ended transmission."
   → Break the loop.
   c. Display the received frame to the user.
   d. Ask the user:
   "Accept this frame? (yes/no)"
   e. If the user enters **yes**:
   → Send `"ACK"` (Acknowledgment) to the client.
   f. Else if the user enters no:
   → Send `"NACK"` (Negative Acknowledgment) to the client.
   g. Go back to step (a).
6. Close all connections and sockets.
7. Stop.

Client:

1. Start.
2. Establish a connection to the server socket (e.g., on port 5000).
3. Create input and output streams for communication.
4. Ask the user to enter:
   i. The number of frames (N) to be sent.
   ii. The window size (W).
5. Initialize current = 1.
6. Repeat until all frames are sent (current ≤ N):
   a. Calculate the window range:
   end = min(current + W - 1, N)
   b. Display message:
   "Sending window: Frames current to end."

c. For each frame i from current to end, do:
i. Send Frame–i to the server.
ii. Wait for the response from the server.
iii. If response is "ACK":
- Display message: “ACK received for Frame–i.”
iv. Else if response is "NACK":
- Display message: “NACK received for Frame–i.”
- Set current = i (to retransmit the window starting from that frame).
- Break from the loop to resend the window.
d. If all frames in the current window are acknowledged:
→ Set current = end + 1 (slide the window forward).

7. After all frames are successfully acknowledged, send "exit" message to the server.
8. Close all connections and sockets.
9. Stop.

Program

Server:

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class SlidingWindowServer {
   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket = new ServerSocket(5000);
      System.out.println("Server started. Waiting for client...");
      Socket socket = serverSocket.accept();
      System.out.println("Client connected.");
      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
      Scanner sc = new Scanner(System.in);
      String frame;
      while (true) {
         frame = in.readLine();
         if (frame.equalsIgnoreCase("exit")) {
            System.out.println("Client ended transmission.");
            break;
         }
```

```
                System.out.println("Received Frame: " + frame);
                System.out.print("Accept this frame? (yes/no): ");
                String choice = sc.nextLine().trim().toLowerCase();
                if (choice.equals("yes")) {
                    out.println("ACK");
                    System.out.println("ACK sent.\n");
                } else {
                    out.println("NACK");
                    System.out.println("NACK sent.\n");
                }
            }
            socket.close();
            serverSocket.close();
            sc.close();
            System.out.println("Connection closed.");
        }
}
```

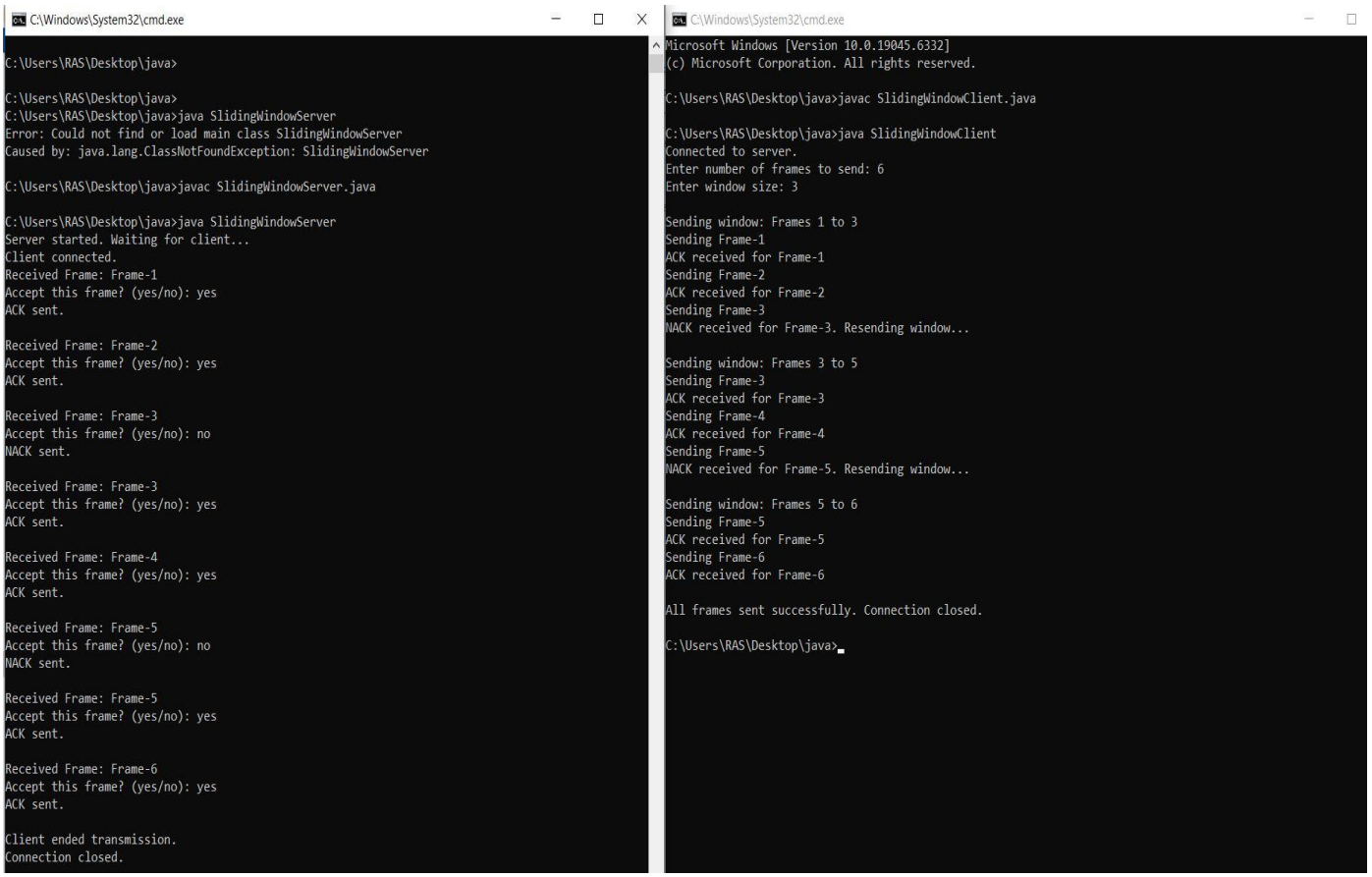
Client:

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
public class SlidingWindowClient {
    public static void main(String[] args) throws IOException {
        Socket socket = new Socket("localhost", 5000);
        BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
        Scanner sc = new Scanner(System.in);
        System.out.println("Connected to server.");
        System.out.print("Enter number of frames to send: ");
        int totalFrames = sc.nextInt();
```

```
        System.out.print("Enter window size: ");
        int windowSize = sc.nextInt();
        int current = 1;
        while (current <= totalFrames) {
            int end = Math.min(current + windowSize - 1, totalFrames);
            System.out.println("\nSending window: Frames " + current + " to " + end);
            for (int i = current; i <= end; i++) {
                System.out.println("Sending Frame-" + i);
                out.println("Frame-" + i);
                String response = in.readLine();
                if (response.equalsIgnoreCase("ACK")) {
                    System.out.println("ACK received for Frame-" + i);
                } else {
                    System.out.println("NACK received for Frame-" + i + ". Resending
window...");
                    // Go-Back-N style retransmission
                    current = i;
                    break;
                }

                if (i == end) {
                    current = end + 1;
                }
            }
        }

        out.println("exit");
        socket.close();
        sc.close();
        System.out.println("\nAll frames sent successfully. Connection closed.");
```

```
        }
}
```

Output:



```
C:\Windows\System32\cmd.exe

C:\Users\RAS\Desktop\java>

C:\Users\RAS\Desktop\java>
C:\Users\RAS\Desktop\java>java SlidingWindowServer
Error: Could not find or load main class SlidingWindowServer
Caused by: java.lang.ClassNotFoundException: SlidingWindowServer

C:\Users\RAS\Desktop\java>javac SlidingWindowServer.java

C:\Users\RAS\Desktop\java>java SlidingWindowServer
Server started. Waiting for client...
Client connected.
Received Frame: Frame-1
Accept this frame? (yes/no): yes
ACK sent.

Received Frame: Frame-2
Accept this frame? (yes/no): yes
ACK sent.

Received Frame: Frame-3
Accept this frame? (yes/no): no
NACK sent.

Received Frame: Frame-3
Accept this frame? (yes/no): yes
ACK sent.

Received Frame: Frame-4
Accept this frame? (yes/no): yes
ACK sent.

Received Frame: Frame-5
Accept this frame? (yes/no): no
NACK sent.

Received Frame: Frame-5
Accept this frame? (yes/no): yes
ACK sent.

Received Frame: Frame-6
Accept this frame? (yes/no): yes
ACK sent.

Client ended transmission.
Connection closed.
```

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.6332]
(c) Microsoft Corporation. All rights reserved.

C:\Users\RAS\Desktop\java>javac SlidingWindowClient.java

C:\Users\RAS\Desktop\java>java SlidingWindowClient
Connected to server.
Enter number of frames to send: 6
Enter window size: 3

Sending window: Frames 1 to 3
Sending Frame-1
ACK received for Frame-1
Sending Frame-2
ACK received for Frame-2
Sending Frame-3
NACK received for Frame-3. Resending window...

Sending window: Frames 3 to 5
Sending Frame-3
ACK received for Frame-3
Sending Frame-4
ACK received for Frame-4
Sending Frame-5
NACK received for Frame-5. Resending window...

Sending window: Frames 5 to 6
Sending Frame-5
ACK received for Frame-5
Sending Frame-6
ACK received for Frame-6

All frames sent successfully. Connection closed.

C:\Users\RAS\Desktop\java>
```

Result:

Thus a program in Java to implement Sliding Window protocol using TCP socket was executed successfully.

EXP.NO:6A **IMPLEMENTATION OF GO BACK N PROTOCOL USING TCP SOCKET**

DATE:

Aim:

To write a program in Java to implementation of Go-Back-N protocol using TCP socket.

Algorithm

Server

1. Start
2. Create a `ServerSocket` on a specific port (e.g., 5000).
3. Wait for a client connection using `accept()`.
4. Once connection is established:
   a. Create input and output streams for communication.
5. Initialize `expectedFrame = 1`.
6. Repeat until termination:
   a. Receive a frame from the client.
   b. If the received frame = `"exit"`, then
   → Display "Client requested termination"
   → Break the loop.
   c. Display the received frame number.
   d. Ask the user manually:
   → "Accept this frame? (yes/no)"
   e. If user input = `"yes"` then
   i. If the frame number == `expectedFrame`,
   → Display "Frame accepted."
   → Send `"ACK-frameNo"` to client.
   → Increment `expectedFrame` by 1.
   ii. Else (frame out of order),
   → Display "Out of order frame."
   → Send `"ACK-(expectedFrame-1)"`.
   f. Else (user input = `"no"`)
   → Display "Frame rejected."
   → Send `"NACK-frameNo"` to client.
7. Close input/output streams and sockets.
8. **Stop.**

Client

1. Start
2. Establish a connection to the server (e.g., port 5000).
3. Create input and output streams for communication.

4. Ask the user to enter:
   a. Total number of frames n.
   b. Window size w.
5. Initialize variables:
   a. base = 1
   b. nextFrame = 1
6. Repeat while base <= n:
   a. Send frames within current window:
   → While (nextFrame < base + w and nextFrame <= n):
   → Send "Frame–nextFrame" to server.
   → Increment nextFrame.
   b. Wait for response from server.
   c. If response starts with "ACK–":
   i. Extract acknowledgment number ackNo.
   ii. Display “ACK received for Frame–ackNo.”
   iii. Set base = ackNo + 1 (slide window forward).
   d. Else if response starts with "NACK–":
   i. Extract negative acknowledgment number nackNo.
   ii. Display “NACK received for Frame–nackNo.”
   iii. Set nextFrame = nackNo (retransmit from that frame).
   iv. Resend frames from nackNo up to window size.
7. After all frames sent successfully, send "exit" message to server.
8. Close all connections and sockets.
9. Stop.

Program:

Server

```
import java.io.*;
import java.net.*;
import java.util.*;
public class GoBackNServer {
   public static void main(String[] args) throws Exception {
      ServerSocket serverSocket = new ServerSocket(5000);
      System.out.println("Server started. Waiting for client connection...");
      Socket socket = serverSocket.accept();
      System.out.println("Client connected.");
      BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
      PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
      Scanner sc = new Scanner(System.in);
```

```
        int expectedFrame = 1;
        String frame;
        while (true) {
            frame = in.readLine();
            if (frame == null) continue;

            if (frame.equalsIgnoreCase("exit")) {
                System.out.println("\nAll frames received successfully. Closing
connection...");
                break;
            }
            System.out.println("\nReceived: " + frame);
            int frameNo = Integer.parseInt(frame.split("-")[1]);
            // If duplicate or out of order frame
            if (frameNo < expectedFrame) {
                System.out.println(" Duplicate Frame-" + frameNo + " ignored. Sending
ACK-" + (expectedFrame - 1));
                out.println("ACK-" + (expectedFrame - 1));
                continue;
            }
            // If out of order (skipped expected frame)
            if (frameNo > expectedFrame) {
                System.out.println(" Out of order frame! Expected Frame-" +
expectedFrame);
                out.println("ACK-" + (expectedFrame - 1));
                continue;
            }

            // Manual user ACK/NACK input
            System.out.print("Accept this frame? (yes/no): ");
            String response = sc.nextLine().trim();

            if (response.equalsIgnoreCase("yes")) {
                System.out.println("Frame-" + frameNo + " accepted. Sending ACK-" +
frameNo);
```

```
                out.println("ACK-" + frameNo);
                expectedFrame++;
            } else {
                System.out.println("Frame-" + frameNo + " rejected. Sending NACK-" +
        frameNo);
                out.println("NACK-" + frameNo);
            }
        }

        socket.close();
        serverSocket.close();
        sc.close();
        System.out.println("Server stopped.");
    }
}
```

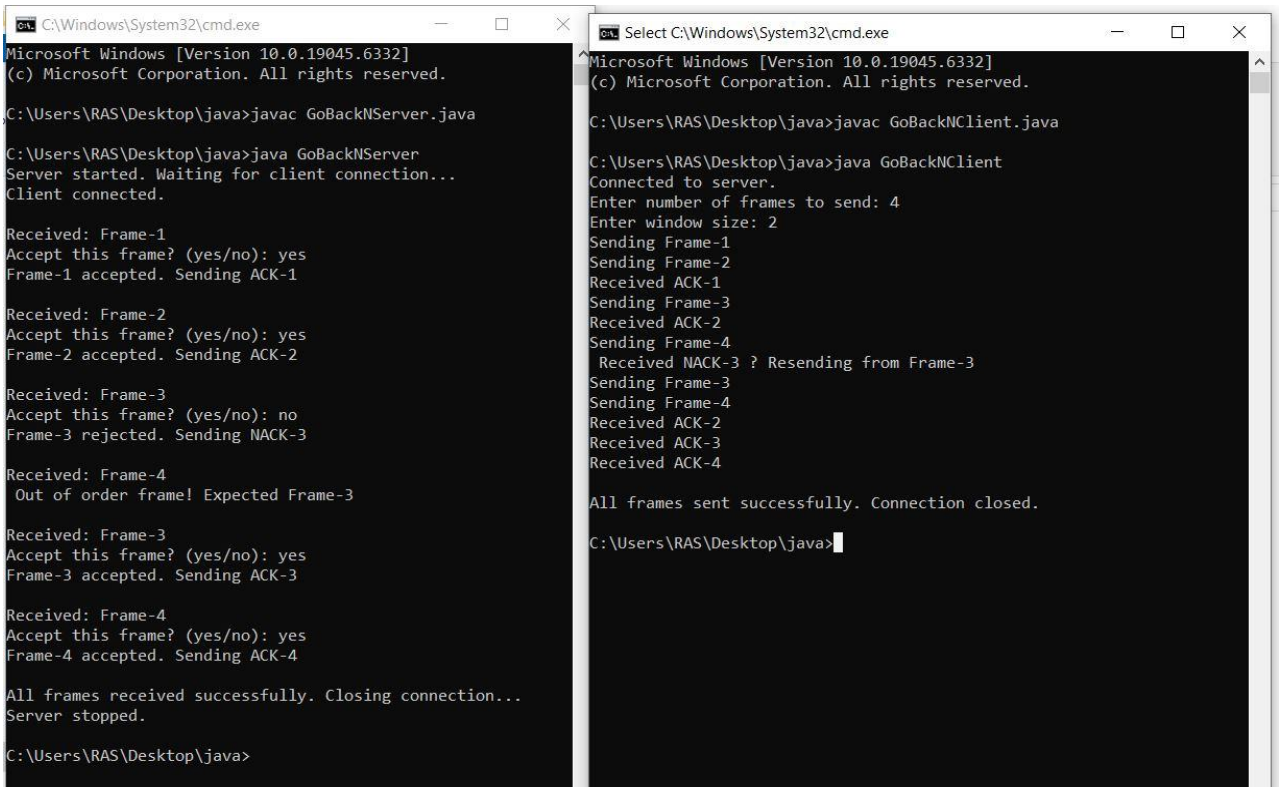
Client

```
    import java.io.*;
    import java.net.*;
    import java.util.*;
    public class GoBackNClient {
        public static void main(String[] args) throws Exception {
            Socket socket = new Socket("localhost", 5000);
            System.out.println("Connected to server.");
            BufferedReader in = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            Scanner sc = new Scanner(System.in);
            System.out.print("Enter number of frames to send: ");
            int totalFrames = sc.nextInt();
```

```
        System.out.print("Enter window size: ");
        int windowSize = sc.nextInt();
        int base = 1;
        int nextFrame = 1;
        String ackMsg;
        while (base <= totalFrames) {
            // Send all frames in current window
            while (nextFrame < base + windowSize && nextFrame <= totalFrames) {
                System.out.println("Sending Frame-" + nextFrame);
                out.println("Frame-" + nextFrame);
                nextFrame++;
            }
            // Wait for ACK or NACK
            ackMsg = in.readLine();
            if (ackMsg == null) continue;
            if (ackMsg.startsWith("ACK-")) {
                int ackNo = Integer.parseInt(ackMsg.split("-")[1]);
                System.out.println("Received " + ackMsg);
                base = ackNo + 1; // Slide window
            } else if (ackMsg.startsWith("NACK-")) {
                int nackNo = Integer.parseInt(ackMsg.split("-")[1]);
                System.out.println(" Received " + ackMsg + " → Resending from Frame-" +
nackNo);
                nextFrame = nackNo; // Retransmit from that frame
            }
```

```
            }
            // Inform server to close after successful transmission
            out.println("exit");
            socket.close();
            sc.close();
            System.out.println("\nAll frames sent successfully. Connection closed.");
        }
}
```

Output:



Result:

Thus a program in Java to implement Go-Back-N protocol using TCP socket was executed successfully.

EXP.NO:6 B

IMPLEMENTATION OF SELECTIVE REPEAT PROTOCOL USING TCP SOCKET

DATE:

Aim:

To write a program in Java to implement Selective Repeat protocol using TCP socket.

Algorithm

Server:

Step 1: Start

- Start the program.
- Create a `ServerSocket` on a specific port (e.g., 5000).
- Wait for a client connection using `accept()`.
- Once connected, create input and output streams for communication.

Step 2: Frame Reception

- Repeat until the client sends "exit":
  a. Receive a frame from the client.
  b. If the frame is `"exit"`, display a message and break the loop.
  c. Extract the frame number from the message (e.g., "Frame–i").
  d. Check if this frame has already been received:
    1. If yes, display "Duplicate Frame ignored", and resend `"ACK-i"`.
    2. If no, continue to step (e).

Step 3: Manual Acknowledgment

- Ask the user:
  `"Accept this frame? (yes/no)"`
- If the user enters:
  1. yes →
     i. Display "Frame accepted".
     ii. Send `"ACK-i"` to client.
     iii. Add the frame number to the set of received frames.
  2. no →
     i. Display "Frame rejected".
     ii. Send `"NACK-i"` to client.

Step 4: Termination

- After receiving `"exit"`, close:
  1. Input/output streams.
  2. Socket and ServerSocket.
- Display "Server stopped".
- Stop.

Client:

Step 1: Start

- Start the program.
- Create a `Socket` connection to the server (e.g., `localhost`, port 5000).

- Create input and output streams for communication.
- Get from the user:
    1. Total number of frames `n`.
    2. Window size `w`.

Step 2: Initialization

- Initialize an array `ackReceived[n]` to keep track of acknowledged frames.
- Set `base = 1`.

Step 3: Frame Transmission

- Repeat until all frames are acknowledged:

  a. For each frame from `base` to `base + w - 1`:
    1. If the frame is not yet acknowledged, send `"Frame-i"` to the server.

       b. Wait for response from server.

       c. If response is:
    2. `"ACK-i"` → mark `ackReceived[i] = true`.
    3. `"NACK-i"` → resend only `"Frame-i"`.

Step 4: Sliding Window Update

- After receiving ACKs:
    1. Slide the window forward while `ackReceived[base] == true`.
    2. Move base pointer to next unacknowledged frame.

Step 5: Termination

- When all frames are acknowledged:
    1. Send `"exit"` to the server.
    2. Close all connections.
- Display "All frames sent successfully. Connection closed."
- Stop.

Program

Server:

```
import java.io.*;
import java.net.*;
import java.util.*;
public class SelectiveRepeatServer {
   public static void main(String[] args) {
      try {
         ServerSocket serverSocket = new ServerSocket(5000);
         System.out.println("Server started. Waiting for client connection...");
         Socket socket = serverSocket.accept();
         System.out.println("Client connected.");
         BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
         PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
```

```
        Scanner scanner = new Scanner(System.in);

        Set<Integer> receivedFrames = new HashSet<>();
        while (true) {
            String frame = in.readLine();
            if (frame == null || frame.equalsIgnoreCase("exit")) {
                System.out.println("All frames received. Closing connection...");
                break;
            }
            int frameNo = Integer.parseInt(frame.split("-")[1]);

            // Check for duplicate frame
            if (receivedFrames.contains(frameNo)) {
                System.out.println("Received duplicate Frame-" + frameNo + " (ignored).
Sending ACK-" + frameNo);
                out.println("ACK-" + frameNo);
                continue;
            }
            System.out.println("\nReceived: " + frame);
            System.out.print("Accept this frame? (yes/no): ");
            String response = scanner.nextLine();

            if (response.equalsIgnoreCase("yes")) {
                System.out.println("Frame-" + frameNo + " accepted. Sending ACK-" +
frameNo);
                out.println("ACK-" + frameNo);
                receivedFrames.add(frameNo);
            }
else {
                System.out.println("Frame-" + frameNo + " rejected. Sending NACK-" +
frameNo);
                out.println("NACK-" + frameNo);
            }
        }
```

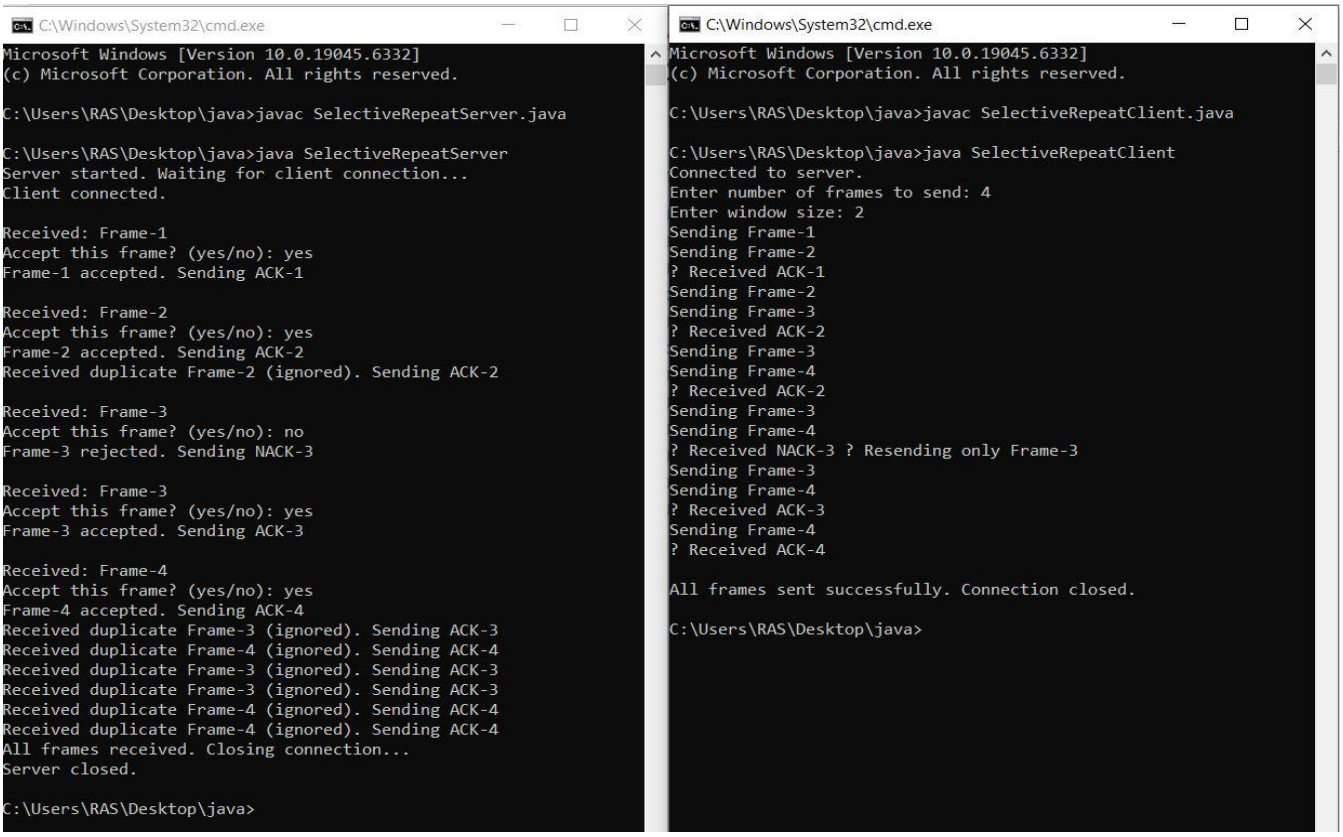
```
                socket.close();
                serverSocket.close();
                System.out.println("Server closed.");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

Client:

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

public class SelectiveRepeatClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 5000);
            System.out.println("Connected to server.");

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
            Scanner scanner = new Scanner(System.in);

            System.out.print("Enter number of frames to send: ");
            int totalFrames = scanner.nextInt();
            System.out.print("Enter window size: ");
            int windowSize = scanner.nextInt();

            boolean[] ackReceived = new boolean[totalFrames + 1];
            int base = 1;

            while (base <= totalFrames) {
                // Send frames within window that are not acknowledged
                for (int i = base; i < base + windowSize && i <= totalFrames; i++) {
                    if (!ackReceived[i]) {
                        System.out.println("Sending Frame-" + i);
```

```
                out.println("Frame-" + i);
            }
        }

        // Wait for ACK/NACK
        String response = in.readLine();
        if (response == null)
            break;

        if (response.startsWith("ACK")) {
            int frameNo = Integer.parseInt(response.split("-")[1]);
            ackReceived[frameNo] = true;
            System.out.println(" Received " + response);
        } else if (response.startsWith("NACK")) {
            int frameNo = Integer.parseInt(response.split("-")[1]);
            System.out.println(" Received " + response + " → Resending only Frame-" +
frameNo);
            out.println("Frame-" + frameNo);
        }

        // Slide window forward for acknowledged frames
        while (base <= totalFrames && ackReceived[base]) {
            base++;
        }
      }

      // Send exit signal
      out.println("exit");
      System.out.println("\nAll frames sent successfully. Connection closed.");

      socket.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Output:



Result:

Thus a program in Java to implement Selective Repeat protocol using TCP socket was executed successfully.

EXPT NO:07 **VLAN CONFIGURATION USING CISCO PACKET TRACER**

DATE:

Aim:

To configure VLANs on a switch using Cisco Packet Tracer.

VLAN:

A VLAN (Virtual Local Area Network) is a network configuration that allows a single physical network to be logically segmented into multiple, separate networks. This segmentation improves security, reduces network congestion, and simplifies management by grouping devices based on function, department, or project, rather than physical location. VLANs enable devices on different switches to communicate as if they were on the same local network, while isolating traffic between unrelated groups, making them essential in modern, scalable network design.

Algorithm:

1. Open Cisco Packet Tracer student version
2. Click and drag one switch and 15 PCs
3. Assign IP address and subnet mask for all PCs.
    - PC1→10.10.10.1 255.0.0.0
    - PC2→10.10.10.2 255.0.0.0
    - PC3→10.10.10.3 255.0.0.0
    - PC4→10.10.10.4 255.0.0.0
    - PC5→10.10.10.5 255.0.0.0
    - PC6→10.10.10.6 255.0.0.0
    - PC7→10.10.10.7 255.0.0.0
    - PC8→10.10.10.8 255.0.0.0
    - PC9→10.10.10.9 255.0.0.0
    - PC10→10.10.10.10 255.0.0.0
    - PC11→10.10.10.11 255.0.0.0
    - PC12→10.10.10.12 255.0.0.0
    - PC13→10.10.10.13 255.0.0.0
    - PC14→10.10.10.14 255.0.0.0
    - PC15→10.10.10.15 255.0.0.0
4. Test connectivity between all the PCs.
5. Create three VLANs in the switch and assign five PCs to each VLAN

    15 PCs – divided into 3 groups:

    a. VLAN 10 → PCs 1–5

    b. VLAN 20 → PCs 6–10

    c. VLAN 30 → PCs 11–15
6. Test connectivity within VLANs and across VLANs.

PROCEDURE (Step by Step)

Step 1: Create VLANs on Switch

Switch> enable
Switch# configure terminal
Switch(config)# vlan 10
Switch(config-vlan)# name Admin
Switch(config-vlan)# exit

Switch(config)# vlan 20
Switch(config-vlan)# name HR
Switch(config-vlan)# exit

Switch(config)# vlan 30
Switch(config-vlan)# name SALES
Switch(config-vlan)# exit

Step 2: Assign Switch Ports to VLANs

VLAN 10 → Ports 1-5
Switch(config)# interface range fa0/1 - 5
Switch(config-if-range)# switchport mode access
Switch(config-if-range)# switchport access vlan 10
VLAN 20 → Ports 6-10
Switch(config)# interface range fa0/6 - 10

Switch(config-if-range)# switchport mode access

Switch(config-if-range)# switchport access vlan 20

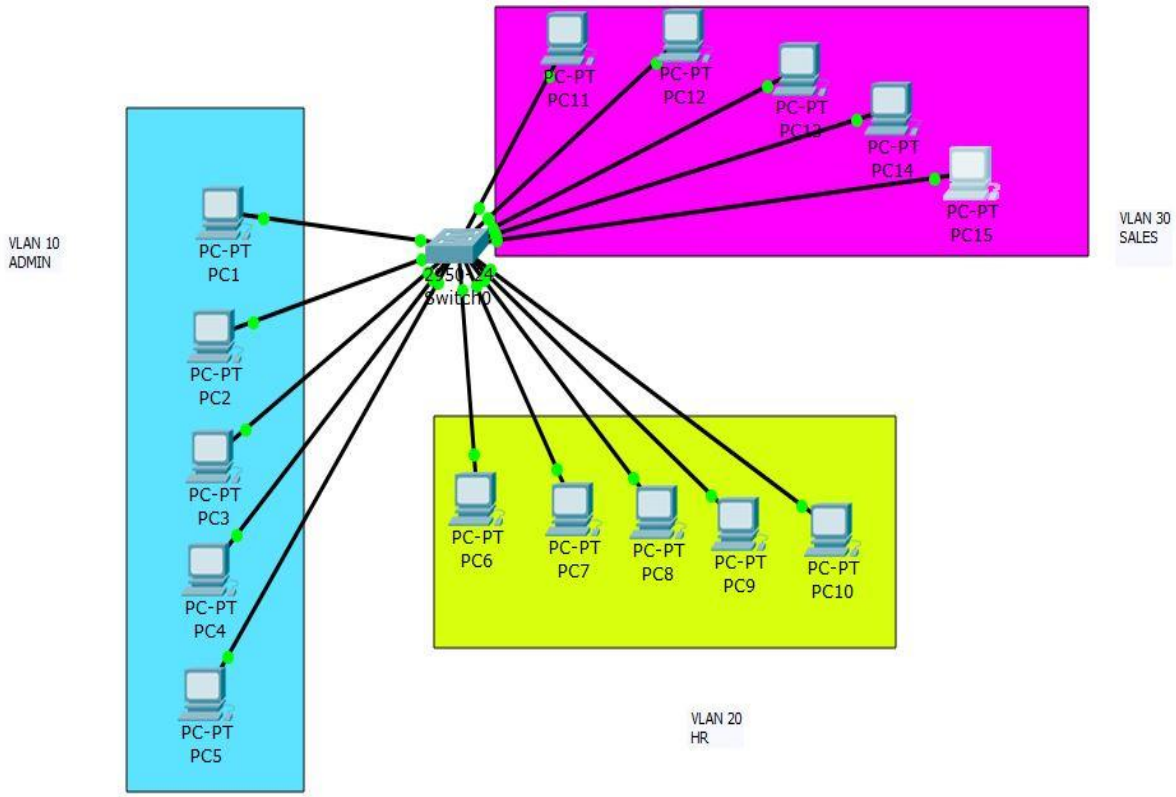
VLAN 30 → Ports 11-15

Switch(config)# interface range fa0/11 - 15

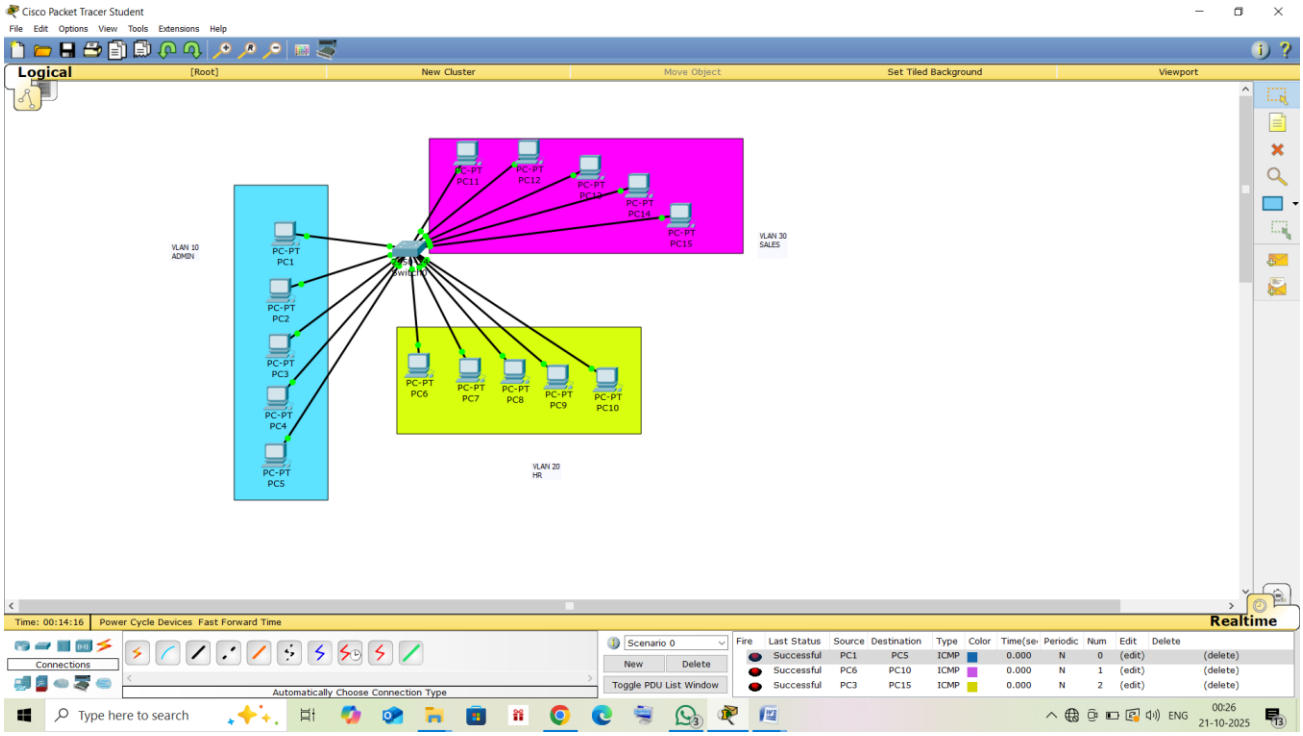
Switch(config-if-range)# switchport mode access

Switch(config-if-range)# switchport access vlan 30

VLAN Design



Output:



RESULT:

Thus, the VLANs were successfully created on the switch in Cisco packet tracer.

EXPT NO:8A **DHCP CONFIGURATION IN A SIMULATED ENVIRONMENT**

DATE:

Aim:

To configure a server as a DHCP server using Cisco Packet Tracer.

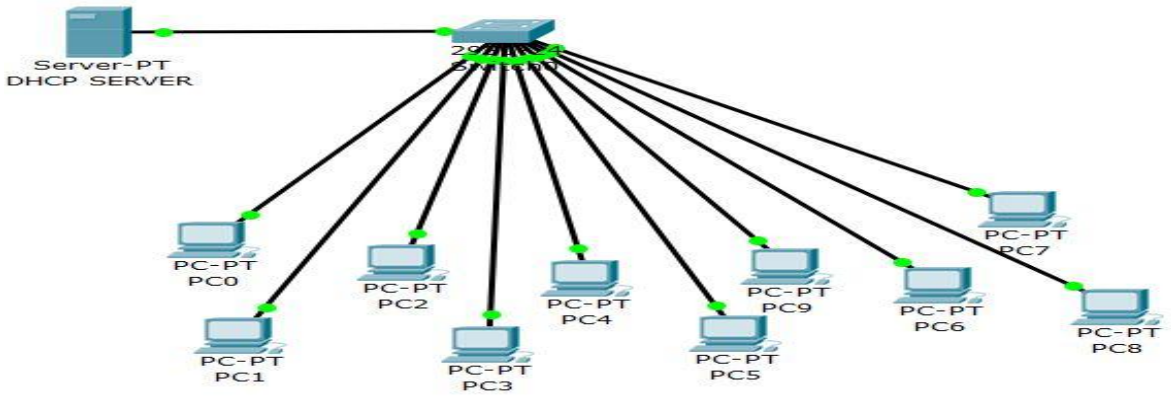
DHCP:

Dynamic Host Configuration Protocol (DHCP) is a network management protocol used to automatically assign IP addresses and other network configuration parameters (such as subnet mask, default gateway, and DNS server) to devices on a network. This eliminates the need for manual configuration of IP addresses, reducing errors and simplifying network administration. DHCP operates on a client-server model, where the DHCP server dynamically provides IP addresses to DHCP clients whenever they connect to the network.

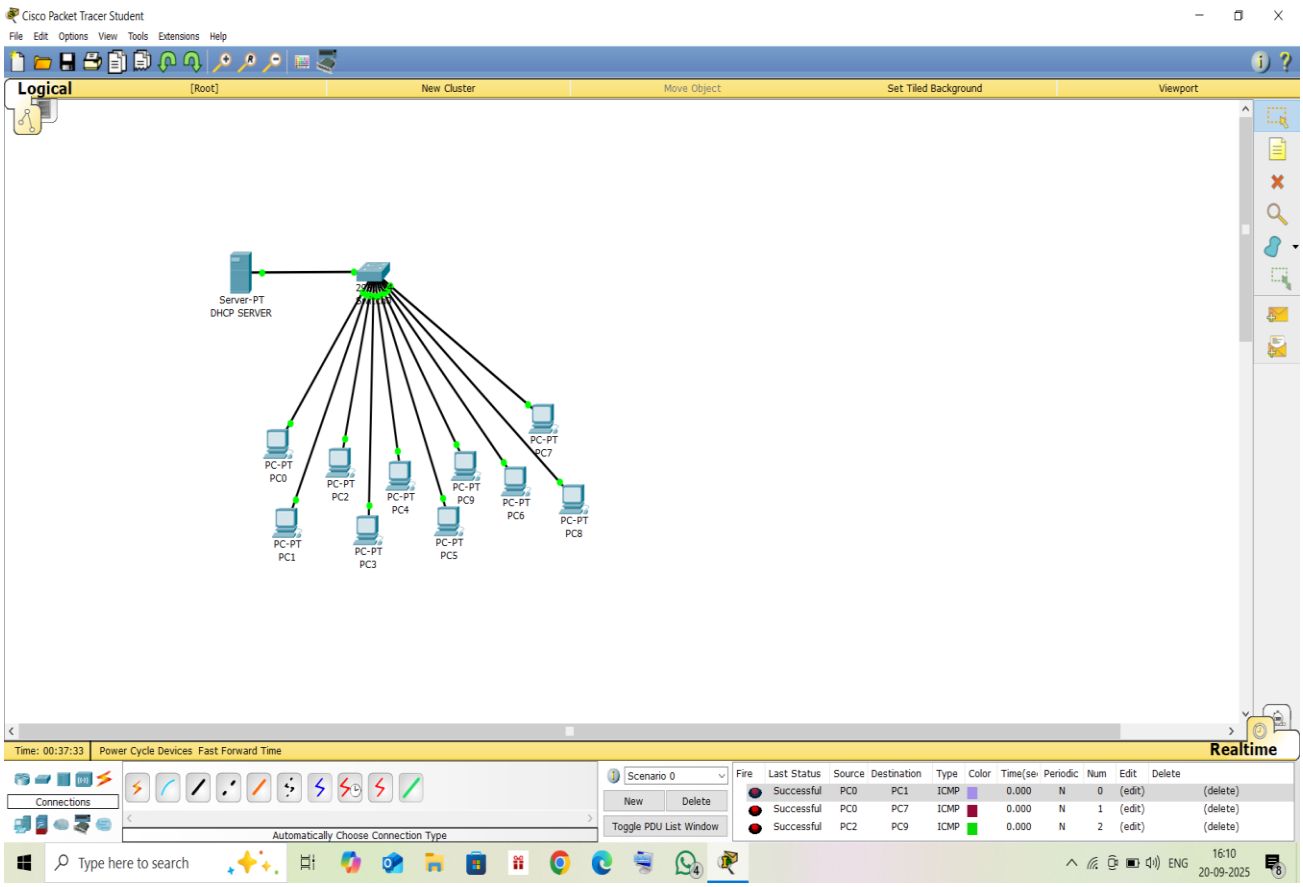
Algorithm:

1. Start.
2. Create the network topology:
    - Place 1 Switch, 1 Server, and 10 PCs in the workspace.
    - Connect all devices using Copper Straight-Through cables.
3. Configure the Server as a DHCP Server:
    - Click on the Server → Desktop → IP Configuration.
    - Assign a static IP address to the server (e.g., IP: `10.10.10.1`, Subnet Mask: `255.0.0.0`).
    - Go to Services → DHCP and turn DHCP ON.
    - Under Pool Name, enter a name (e.g., `LAN_Pool`).
    - Set the following:
        - Start IP Address: `10.10.10.2`
        - Subnet Mask: `255.0.0.0`
        - Maximum Number of Users: `10`
    - Click Add to save the configuration.
4. Configure PCs to Obtain IP Automatically:
    - For each PC, go to Desktop → IP Configuration.
    - Select DHCP instead of static.
    - The PC will automatically receive an IP address from the DHCP server.
5. Verify IP Assignment:
    - Check each PC's IP Configuration to ensure it received a unique IP from the DHCP server.
6. Test Connectivity:
    - Use Command Prompt in any PC.
    - Type `ping 10.10.10.1` to check connectivity with the server.
    - Type `ping` followed by another PC's IP to check inter-PC communication.
7. If all pings are successful, DHCP configuration is complete.
8. Stop.

Design:



Output:



RESULT:

Thus a server is successfully configured as DHCP server using Cisco Packet Tracer.

EXPT NO:8B **DNS CONFIGURATION IN A SIMULATED ENVIRONMENT**

DATE:

Aim:

To configure a server as a DNS server using Cisco Packet Tracer.

DNS:

The **Domain Name System (DNS)** is a network service that translates human-readable domain names, like *www.google.com*, into corresponding IP addresses that computers use to identify each other on a network. It acts as the internet's directory, allowing users to access websites without needing to remember numerical IP addresses. When a user enters a domain name in a web browser, the DNS server resolves it to its IP address, enabling communication between the client and the correct server. This process simplifies navigation on the internet and ensures efficient, user-friendly access to online resources.

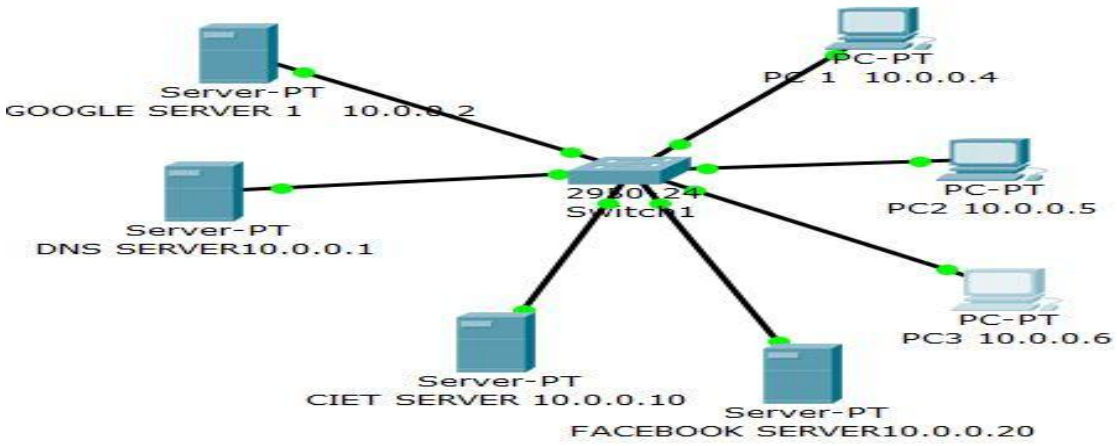
Algorithm:

1. Start.
2. Create the Network Topology:
    - Place one switch, four servers (name it as DNS, Google, CIET, Face book), and three PCs in the workspace.
    - Connect all devices to the switch using Copper Straight-Through cables.
3. Assign IP Addresses:
    - DNS Server: `10.0.0.1`
    - Google Server: `10.0.0.2`
    - CIET Server: `10.0.0.10`
    - Face book Server: `10.0.0.20`
    - PC1: `10.0.0.4`
    - PC2: `10.0.0.5`
    - PC3: `10.0.0.6`
    - Subnet Mask: `255.0.0.0` for all devices
    - Default Gateway: `10.0.0.1`
4. Configure DNS Server:
    - Go to Services → DNS on the DNS server.
    - Turn DNS ON.
    - Add domain names and their respective IPs, for example:
        - Name: google.com → Address: `10.0.0.2`
        - Name: ciet.com → Address: `10.0.0.10`
        - Name: facebook.com → Address: `10.0.0.20`
        - Change the html header for each page
5. Configure PCs to Use the DNS Server:
    - Go to each PC → Desktop → IP Configuration.
    - Assign the respective IP address and Subnet Mask.
    - In DNS Server field, enter `10.0.0.1`.
6. Verify Connectivity:
    - Open Command Prompt on each PC.

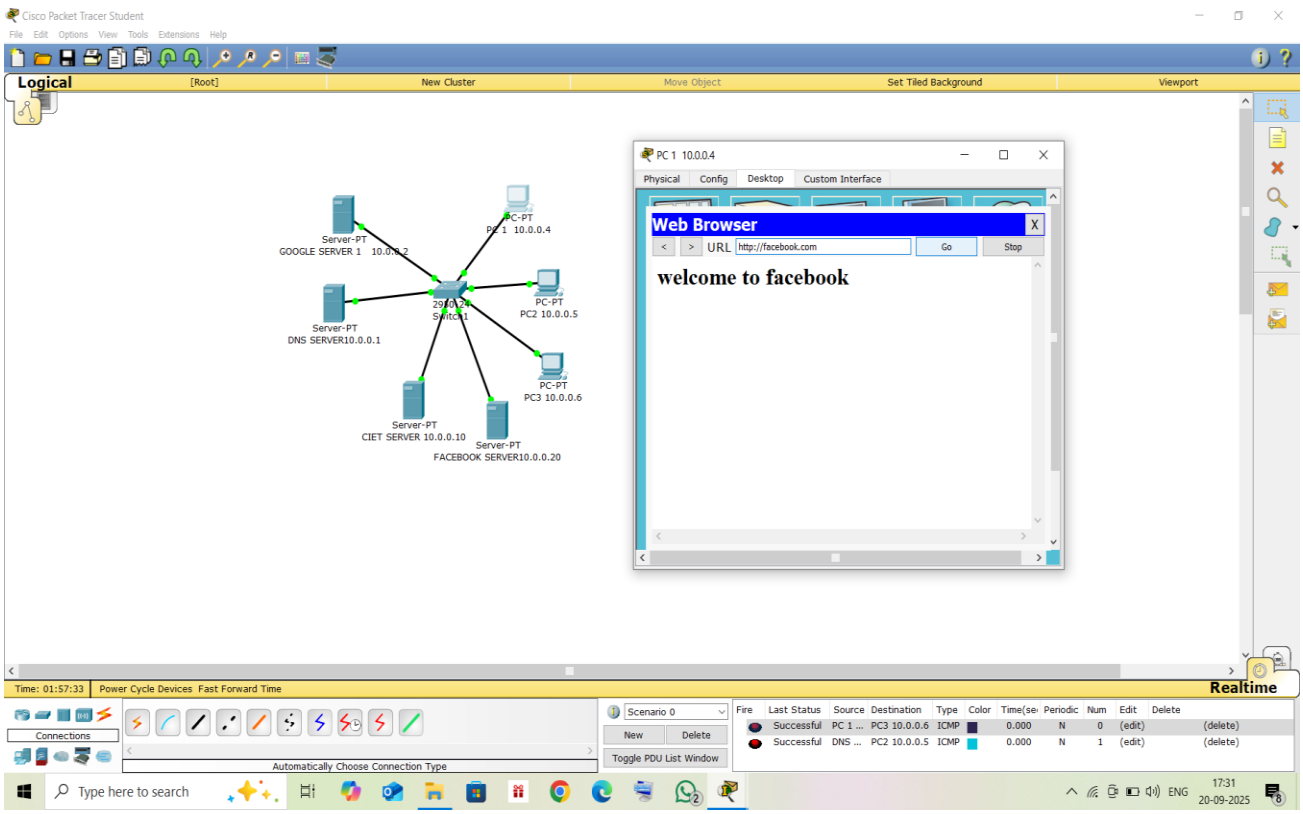
- Test using:
  - `ping 10.0.0.1` (to check connectivity with DNS server)
  - `ping 10.0.0.2`, `10.0.0.10`, `10.0.0.20` (to check server connections)
  - `ping google.com`, `ping ciet.com`, `ping facebook.com` (to test DNS resolution)

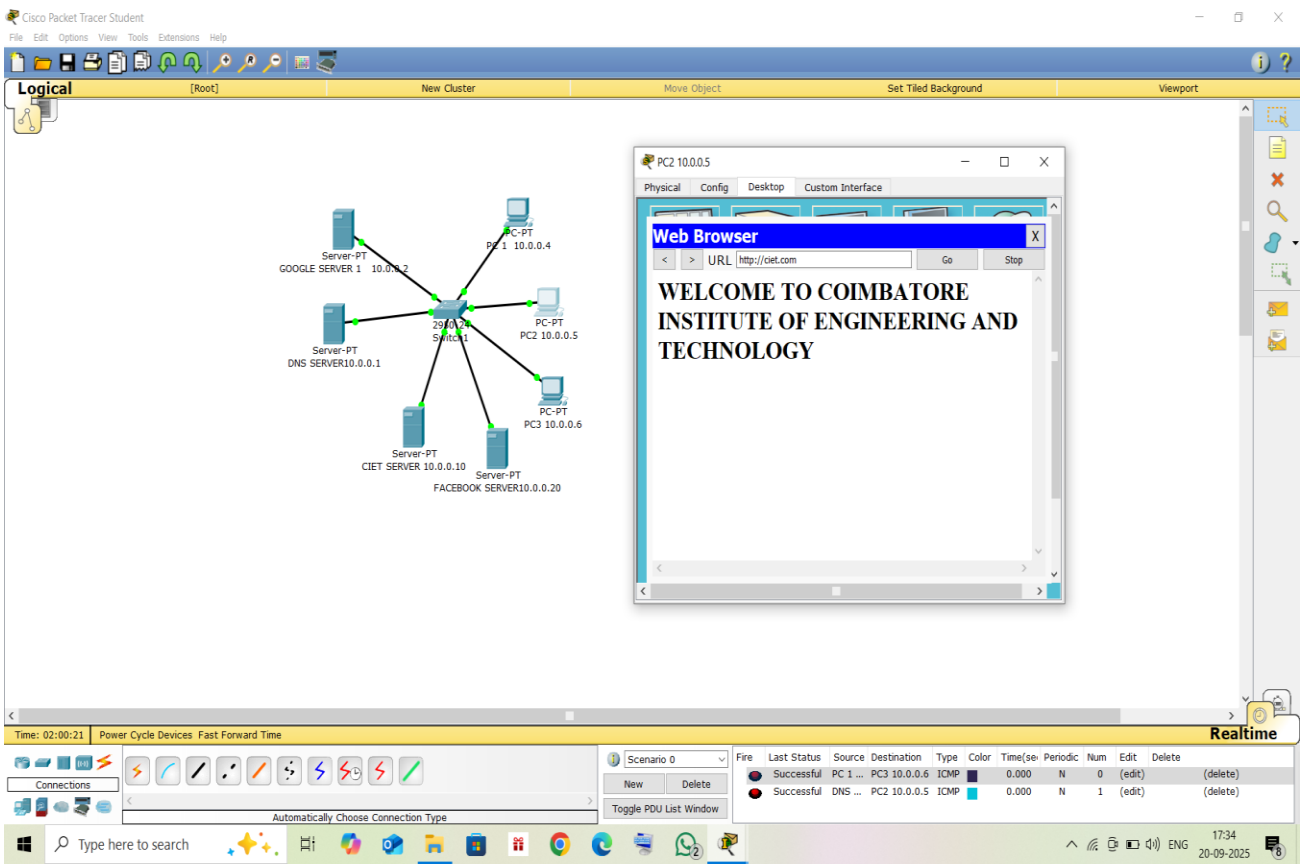
7. If all pings are successful, DNS configuration is correct.
8. Stop.

Design:



Output:





RESULT:

Thus a server is successfully configured as DNS server using Cisco Packet Tracer.

EXP.NO:9A **INTERCONNECTING TWO LANS USING A SINGLE ROUTER IN CISCO PACKET TRACER**

DATE:

Aim:

To interconnect two LANs using a single Router in Cisco Packet Tracer.

Algorithm:

1. Start
2. Create Network Devices
   - Place one Router (1841) in the workspace.
   - Place two Switches (2960) for each LAN.
   - Place 5 PCs for each LAN (total 10 PCs).
   - Connect Devices
   - Connect Router → Switch0 (LAN 1) using a copper straight-through cable.
   - Connect Router → Switch1 (LAN 2) using a copper straight-through cable.
   - Connect each PC to its respective Switch using copper straight-through cables.
3. Assign IP Addresses
   LAN 1 (Network: 10.10.10.0/24):
   a. PC1 → 10.10.10.1
   b. PC2 → 10.10.10.2
   c. PC3 → 10.10.10.3
   d. PC4 → 10.10.10.4
   e. PC5 → 10.10.10.5
   LAN 2 (Network: 20.20.20.0/24):
   f. PC6 → 20.20.20.1
   g. PC7 → 20.20.20.2
   h. PC8 → 20.20.20.3
   i. PC9 → 20.20.20.4
   j. PC10 → 20.20.20.5
4. Configure Router Interfaces
   Click on Router0 → CLI tab.
   Enter the following commands:

```
Router> enable
Router# configure terminal
Router(config)# interface fastEthernet0/0
Router(config-if)# ip address 10.10.10.0 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface fastEthernet0/1
Router(config-if)# ip address 20.20.20.0 255.255.255.0
Router(config-if)# no shutdown
```

Router(config-if)# exit
Router(config)# end
Router# write

5. Assign Default Gateways
   For LAN 1 PCs:
   Default gateway = 10.10.10.0
   For LAN 2 PCs:
   Default gateway = 20.20.20.0
6. Verify Connectivity
7. Stop

Design:

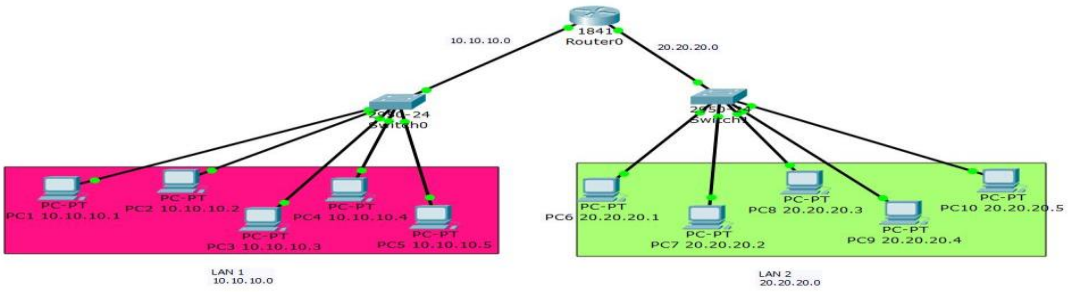


Fig: Design of the network

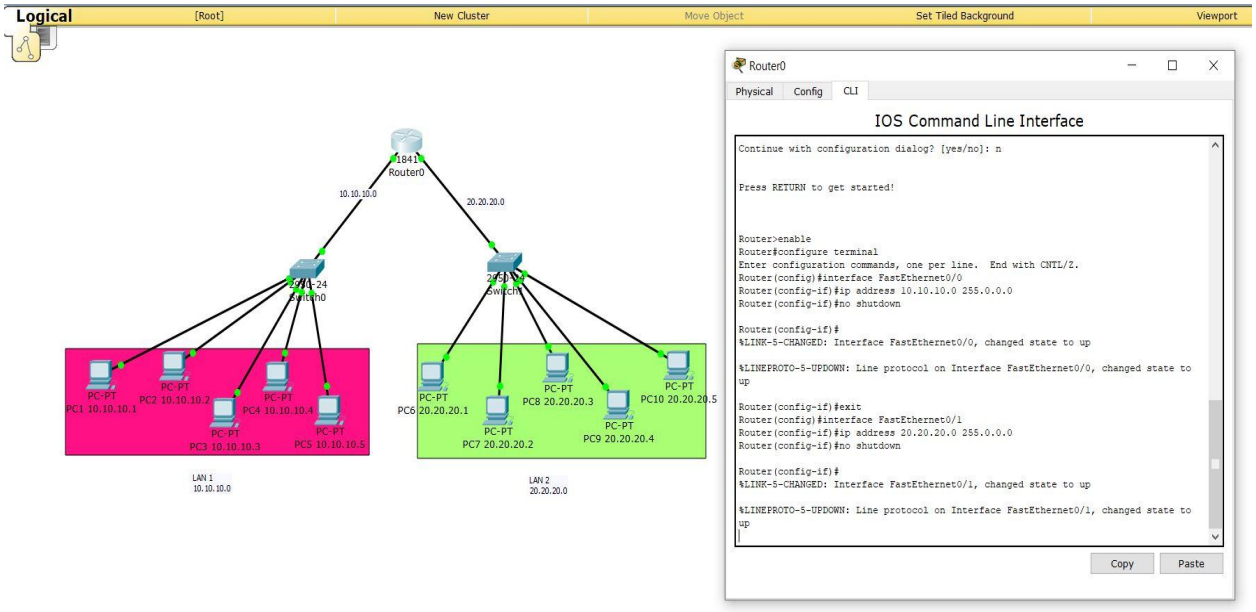


Fig: Configuration of Router

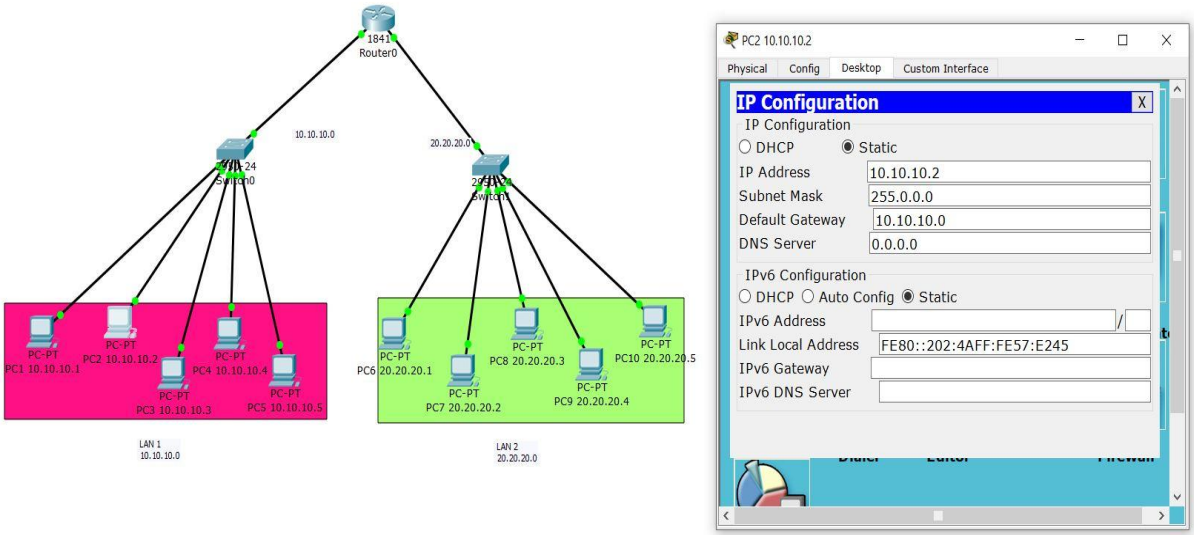


Fig: Configuration of Pc in LAN1

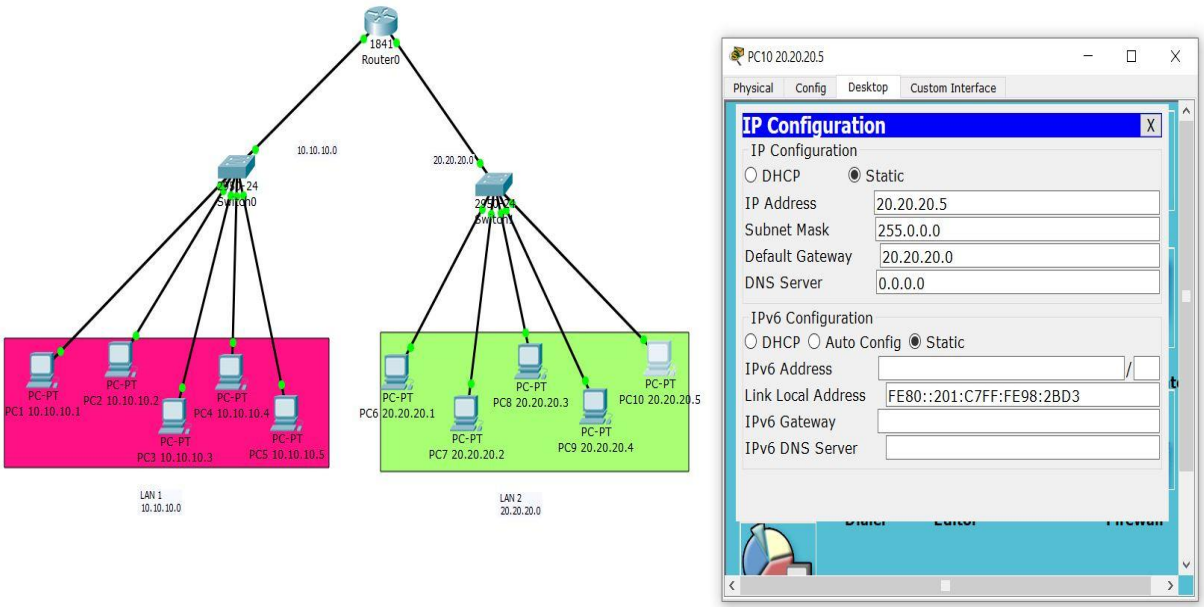
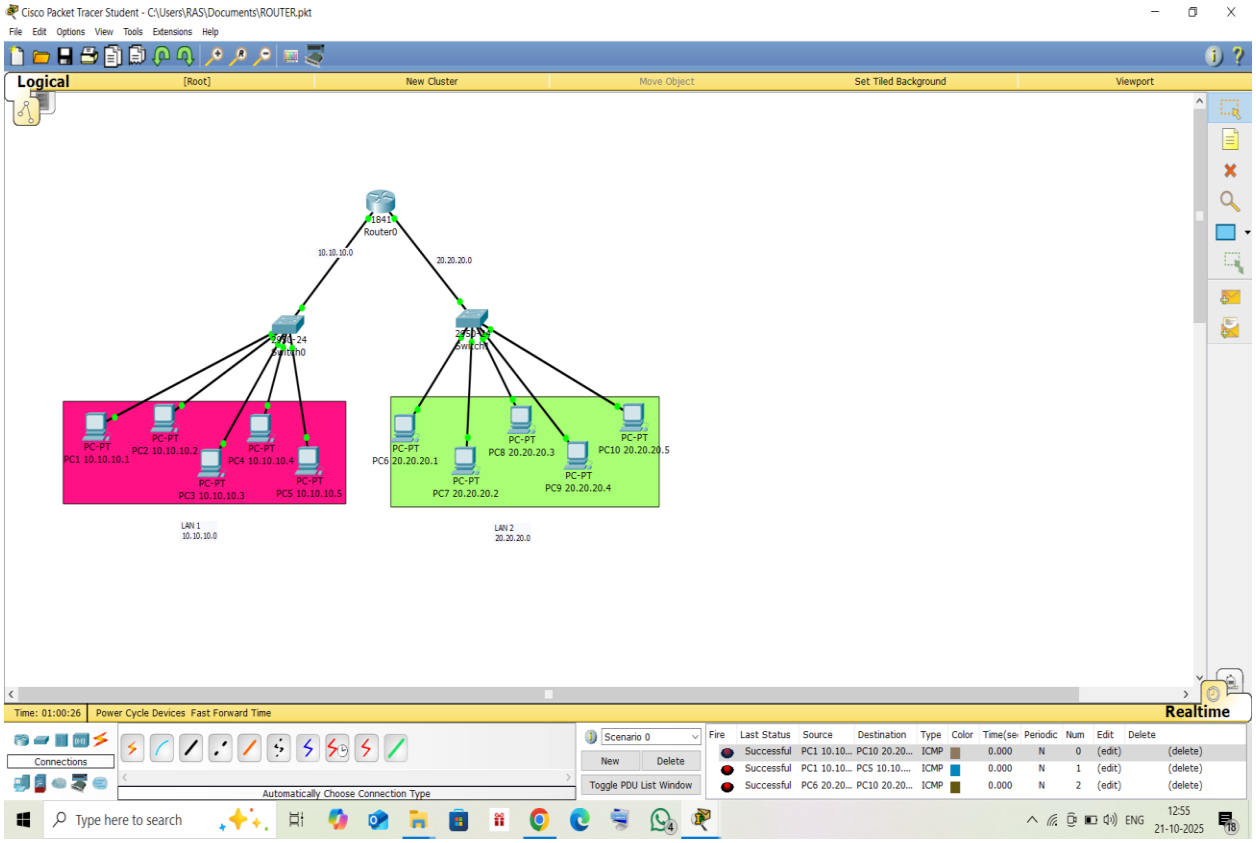


Fig: Configuration of Pc in LAN2

Output:



RESULT:

Thus we successfully interconnect two LANs using a single Router in Cisco Packet Tracer.

EXP.NO:9 B **INTERCONNECTING TWO LANS USING TWO ROUTERS IN CISCO PACKET TRACER**

DATE:

Aim:

To interconnect two LANs using two Routers in Cisco Packet Tracer.

Algorithm:

1. Start
2. Create Network Devices
   - Place two routers (Router1 and Router2).
   - Place two switches (Switch0 for LAN 1, Switch1 for LAN 2).
   - Place five PCs in each LAN.
3. Connect Devices
   - Connect Router1 → Switch0 using a copper straight-through cable.
   - Connect Router2 → Switch1 using a copper straight-through cable.
   - Connect each PC to its respective switch.
   - Connect Router1 → Router2 using a serial or crossover cable (or FastEthernet ports).
4. Assign IP Addresses
   - LAN 1 Network (10.10.10.0/24):
     - PCs: 10.10.10.1 to 10.10.10.5
     - Router1 interface (toward LAN1): 10.10.10.254
   - LAN 2 Network (20.20.20.0/24):
     - PCs: 20.20.20.1 to 20.20.20.5
     - Router2 interface (toward LAN2): 20.20.20.254
   - Router-to-Router Network (30.30.30.0/24):
     - Router1 interface (toward Router2): 30.30.30.1
     - Router2 interface (toward Router1): 30.30.30.2
5. Configure Router1
   - Open Router1 → CLI and enter:
   - `Router> enable`
   - `Router# configure terminal`
   - `Router(config)# interface fastEthernet0/0`
   - `Router(config-if)# ip address 10.10.10.0 255.255.255.0`
   - `Router(config-if)# no shutdown`
   - `Router(config-if)# exit`
   - `Router(config)# interface fastEthernet0/1`
   - `Router(config-if)# ip address 30.30.30.1 255.255.255.0`
   - `Router(config-if)# no shutdown`
   - `Router(config-if)# exit`
   - Configure Static Route to reach LAN 2:
   - `Router(config)# ip route 20.20.20.0 255.255.255.0 30.30.30.2`
   - `Router(config)# end`
   - `Router# write`

6. Configure Router2
    - Open Router2 → CLI and enter:
    - `Router> enable`
    - `Router# configure terminal`
    - `Router(config)# interface fastEthernet0/0`
    - `Router(config-if)# ip address 20.20.20.0 255.255.255.0`
    - `Router(config-if)# no shutdown`
    - `Router(config-if)# exit`
    - `Router(config)# interface fastEthernet0/1`
    - `Router(config-if)# ip address 30.30.30.2 255.255.255.0`
    - `Router(config-if)# no shutdown`
    - `Router(config-if)# exit`
    - Configure Static Route to reach LAN 1:
    - `Router(config)# ip route 10.10.10.0 255.255.255.0 30.30.30.1`
    - `Router(config)# end`
    - `Router# write`
7. Assign Default Gateways
    - LAN 1 PCs: Default gateway → 10.10.10.0
    - LAN 2 PCs: Default gateway → 20.20.20.0
8. Verify Connectivity
    - Open Command Prompt on PC1 (LAN 1) → Ping PC6 (LAN 2):
    - `ping 20.20.20.1`
    - If replies are received, static routing is successfully configured.
9. Stop

Design:

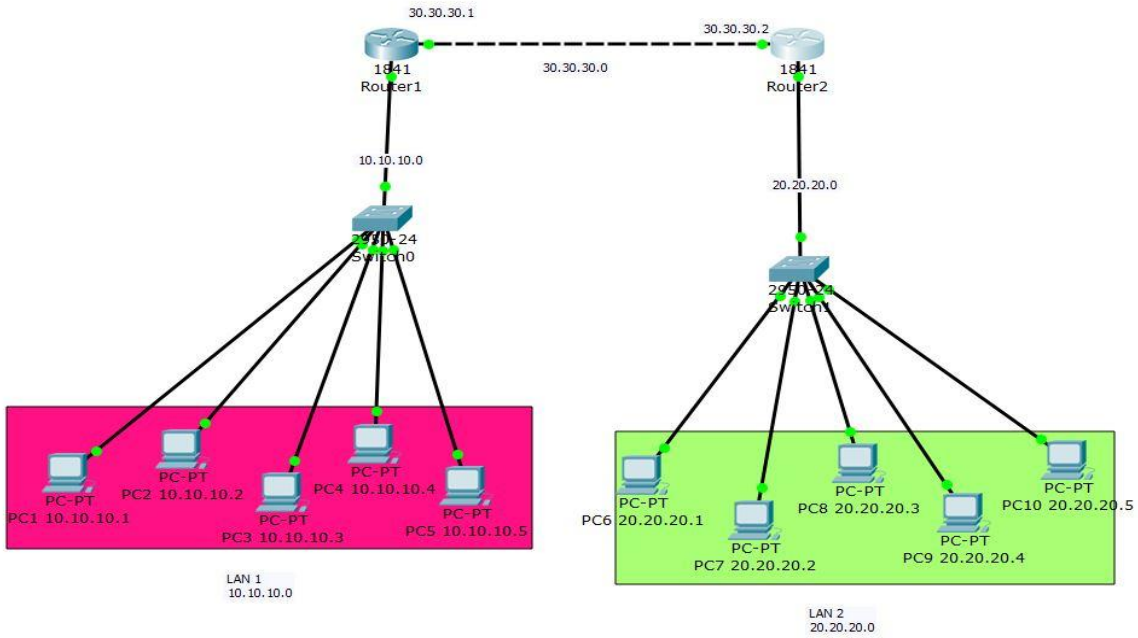


Fig: Network design

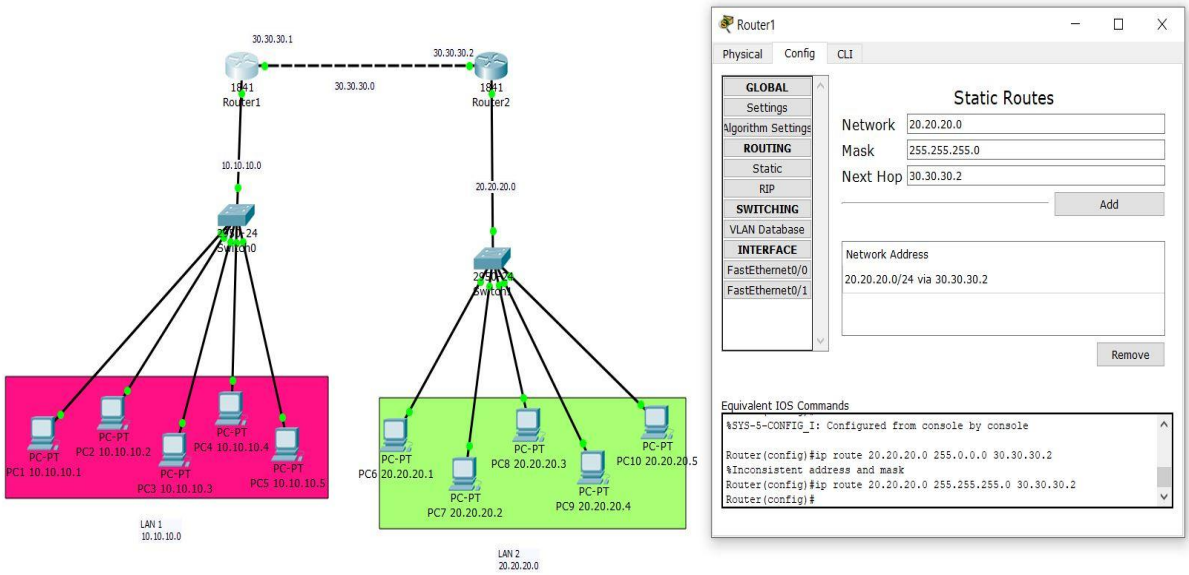


Fig: Configuration of Router1

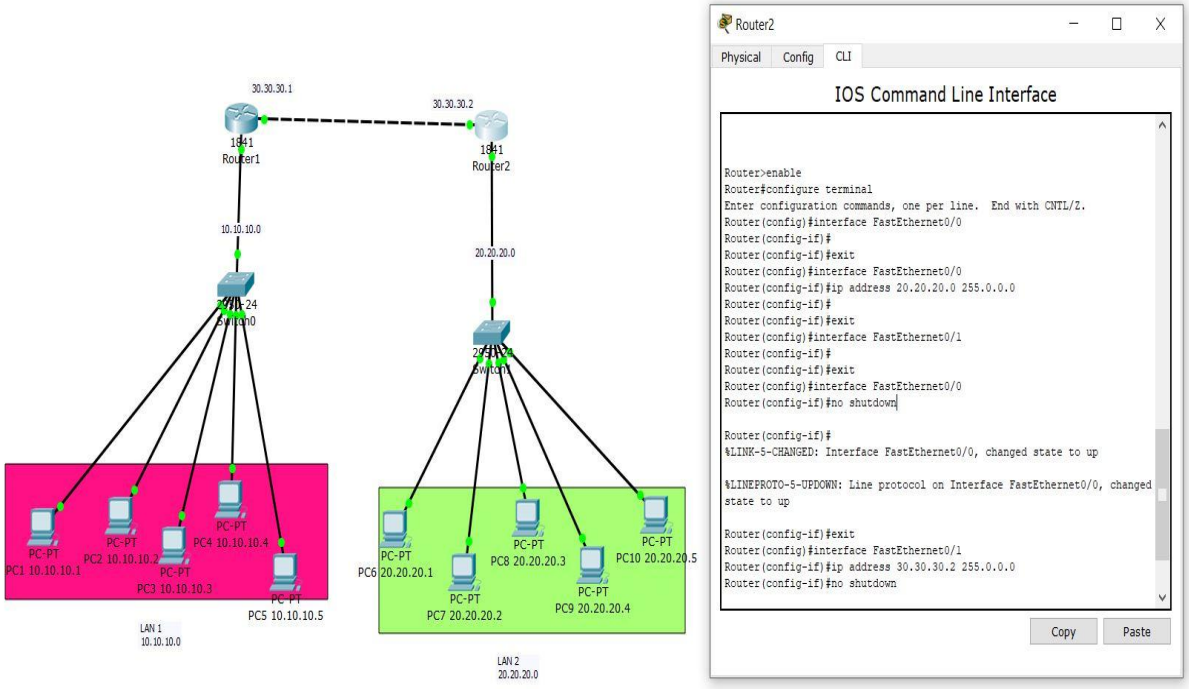
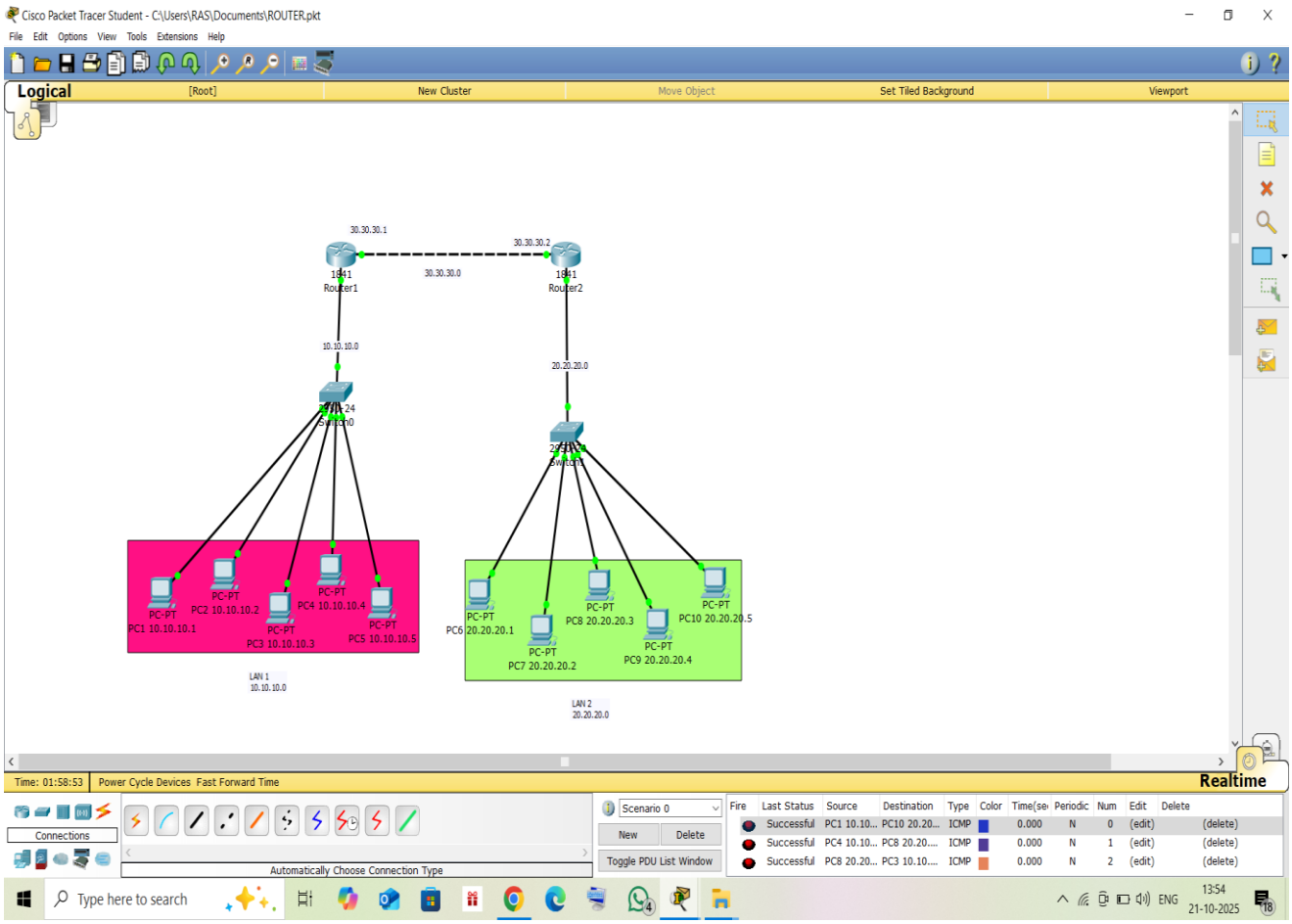


Fig: Configuration of Router2 in CLI

Output:



RESULT:

Thus two LANs with different IP networks are successfully connected through two routers using static routing, enabling inter-network communication.

EXP.NO:9 C **INTERCONNECTING FOUR LANS USING TWO ROUTERS IN CISCO PACKET TRACER**

DATE:

Aim:

Algorithm:

1. Start
2. Create Network Devices
    - Place two routers (Router1 and Router2).
    - Place four switches (Switch0 for LAN 1, Switch2 for LAN 2, Switch3 for LAN 3 Switch1 for LAN 4).
    - Place five PCs in each LAN.
3. Connect Devices
    - Connect Router1 → Switch0 using a copper straight-through cable.
    - Connect Router1 → Switch2 using a copper straight-through cable.
    - Connect Router2 → Switch1 using a copper straight-through cable.
    - Connect Router2 → Switch3 using a copper straight-through cable.
    - Connect each PC to its respective switch.
    - Connect Router1 → Router2 using a serial connection.
4. Assign IP addresses and subnet masks to each PC according to their LAN:
    - **LAN1 (10.10.10.0/8):**
        - PCs: 10.10.10.1 – 10.10.10.5
        - Subnet Mask: 255.0.0.0
        - Default Gateway: 10.10.10.254
    - **LAN2 (30.30.30.0/8):**
        - PCs: 30.30.30.1 – 30.30.30.5
        - Subnet Mask: 255.0.0.0
        - Default Gateway: 30.30.30.254
    - **LAN3 (40.40.40.0/8):**
        - PCs: 40.40.40.1 – 40.40.40.5
        - Subnet Mask: 255.0.0.0
        - Default Gateway: 40.40.40.254
    - **LAN4 (20.20.20.0/8):**
        - PCs: 20.20.20.1 – 20.20.20.5
        - Subnet Mask: 255.0.0.0
        - Default Gateway: 20.20.20.254

5. Configure Router1 Interfaces

    Enter configuration mode:

```
Router> enable
Router# configure terminal
```

Assign IPs to interfaces:

```
Router(config)# interface fastethernet0/0
Router(config-if)# ip address 10.10.10.254 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# interface fastethernet0/1
Router(config-if)# ip address 30.30.30.254 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# interface serial0/0/0
Router(config-if)# ip address 50.50.50.1 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
```

6. Configure Router2 Interfaces

Enter configuration mode:

```
Router> enable
Router# configure terminal
```

Assign IPs to interfaces:

```
Router(config)# interface fastethernet0/0
Router(config-if)# ip address 20.20.20.254 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# interface fastethernet0/1
Router(config-if)# ip address 40.40.40.254 255.0.0.0
Router(config-if)# no shutdown
Router(config-if)# exit

Router(config)# interface serial0/0/0
Router(config-if)# ip address 50.50.50.2 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
```

7. Configure Static Routing

On Router1, configure routes to networks behind Router2:

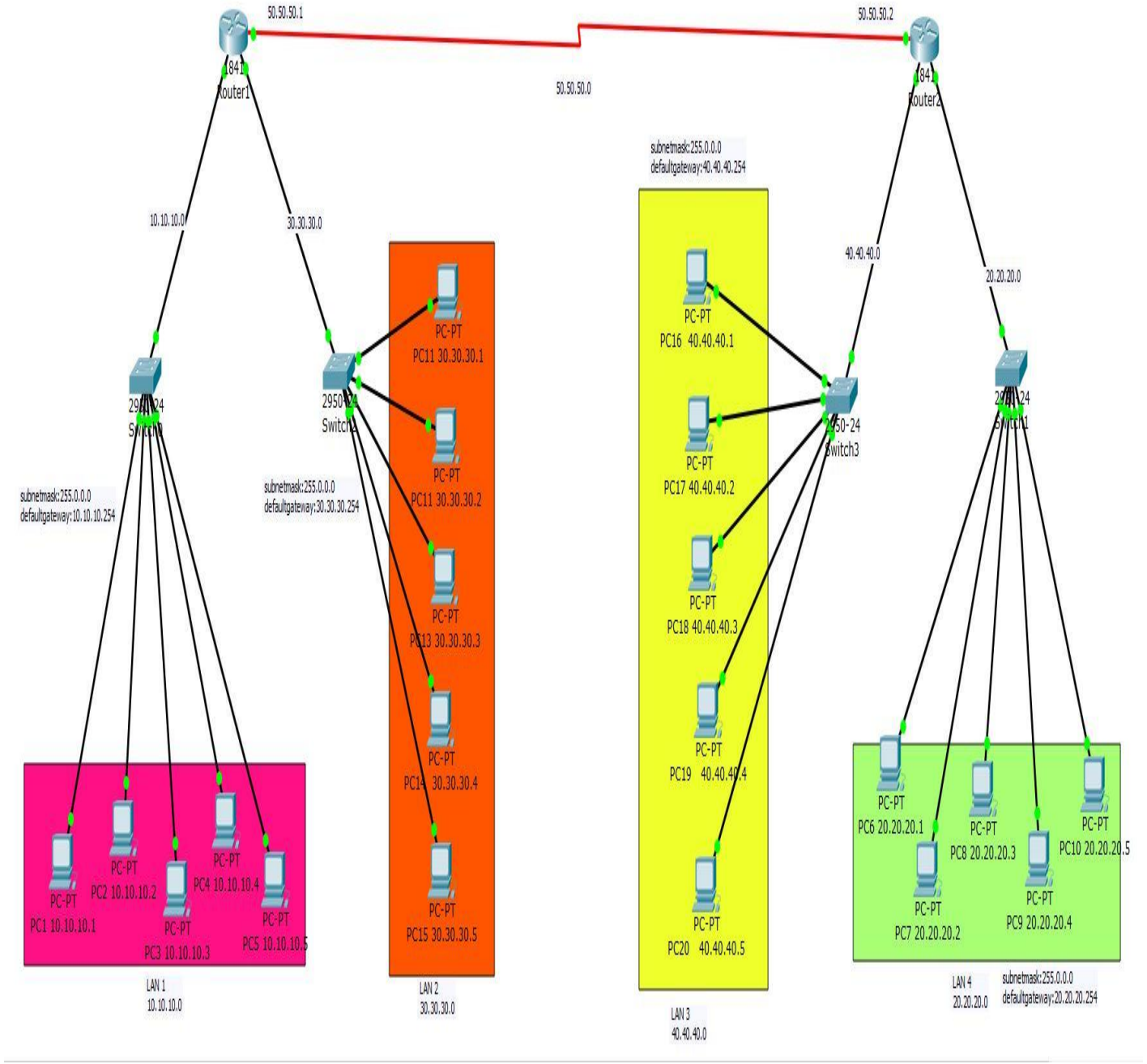
```
Router1(config)# ip route 20.20.20.0 255.0.0.0 50.50.50.2
Router1(config)# ip route 40.40.40.0 255.0.0.0 50.50.50.2
```

On Router2, configure routes to networks behind Router1:

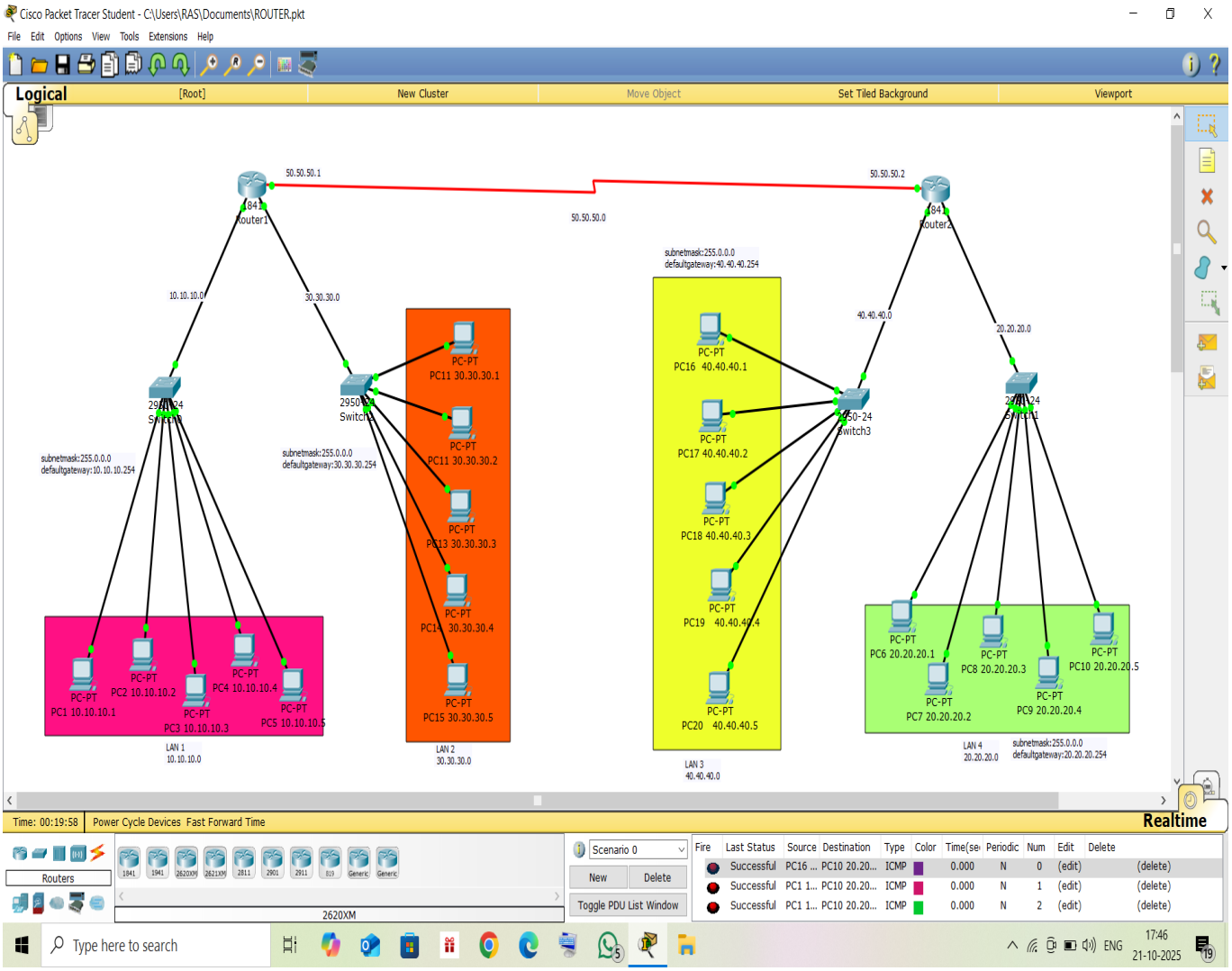
```
Router2(config)# ip route 10.10.10.0 255.0.0.0 50.50.50.1
Router2(config)# ip route 30.30.30.0 255.0.0.0 50.50.50.1
```

8. Verify Connections
9. Stop

Design:



Output:



RESULT:

Thus we successfully interconnect four LANs using two Routers in Cisco Packet Tracer.

EXP.NO:10 A **STUDY OF WIRE SHARK TOOL**

DATE:

Aim:

To study Wire Shark Tool in detail.

Introduction:

Wireshark is a software tool used to monitor the network traffic through a network interface. It is the most widely used network monitoring tool today. Wireshark is loved equally by system administrators, network engineers, network enthusiasts, network security professionals and black hat hackers. The extent of its popularity is such, that experience with Wireshark is considered as a valuable/essential trait in a computer networking-related professional.

Reasons why Wireshark is so popular:

1. It has a great GUI as well as a conventional CLI (T Shark).
2. It offers network monitoring on almost all types of network standards (Ethernet, WLAN, Bluetooth etc)
3. It is open-source with a large community of backers and developers.
4. All the necessary components for monitoring, analyzing and documenting the network traffic are present. It is free to use.

History of Wireshark

Wireshark was started with the intention of developing a tool for closely analyzing network packets. It was started by Gerald Combez in 1997. Its initial name was Ethereal. It was initially released in July 1998 as version 0.2.0. Due to the support it got from the developer community, it grew rapidly and was released as version 1.0 in 2008, almost two years after it was renamed to Wireshark.

Wireshark installation:

Windows:

- You can do a proper installation or run Wireshark as a portable app on your windows system. To download the installation executable or the portable app go to Wireshark Downloads.
- Run the executable and follow on-screen instructions to complete the installation.

Here are some reasons people use Wireshark:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- QA engineers use it to verify network applications
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals

Features

The following are some of the many features Wireshark provides:

- Available for *UNIX* and *Windows*.
- *Capture* live packet data from a network interface.
- *Open* files containing packet data captured with tcpdump/WinDump, Wireshark, and many other packet capture programs.
- *Import* packets from text files containing hex dumps of packet data.
- Display packets with *very detailed protocol information*.
- *Save* packet data captured.
- *Export* some or all packets in a number of capture file formats.
- *Filter packets* on many criteria.
- *Search* for packets on many criteria.
- *Colorize* packet display based on filters.
- Create various *statistics*.

RESULT:

Thus the study of Wire Shark Tool was done successfully.

EXP.NO:10 B **WIRE SHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS**

DATE:

Aim:

Study of working with captured packets and analyzing by scanning and filtering of packets.

Working with Packets

You will eventually encounter situations involving a very large number of packets. As the number of these packets grows into the thousands and even millions, you will need to be able to navigate through packets more efficiently. For this purpose, Wireshark allows you to find and mark packets that match certain criteria. You can also print packets for easy reference.

Finding Packets

To find packets that match particular criteria, open the Find Packet dialog, shown in Figure: 1 by pressing CTRL-F.

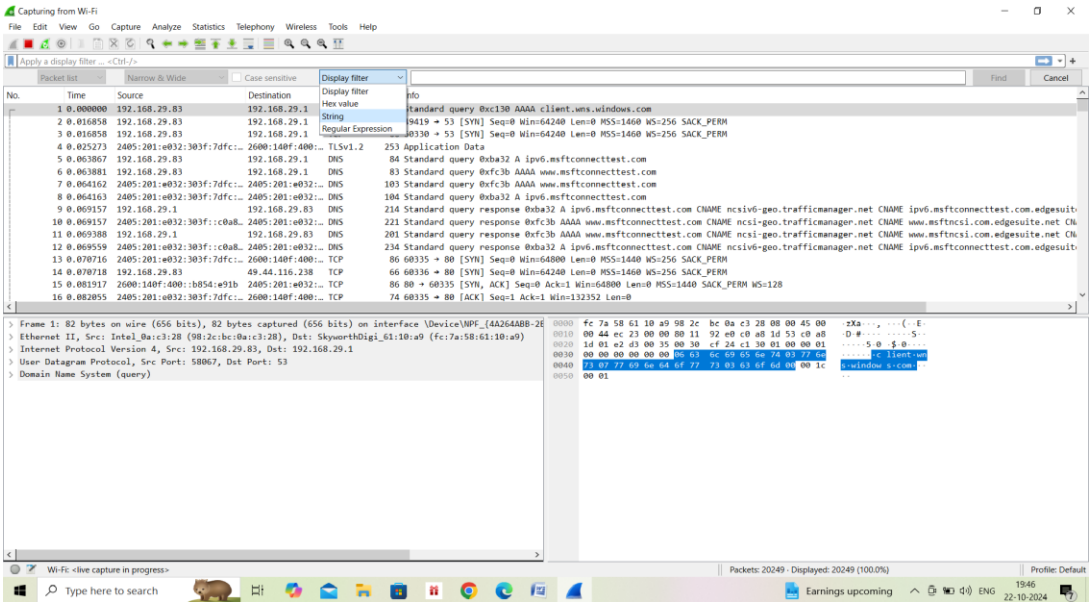


Figure 1

This dialog offers three options for finding packets:

- The Display filter option allows you to enter an expression based filter that will find only those packets that satisfy that expression.
- The Hex value option searches for packets with a

hexadecimal (with bytes separated by colons) value you specify.

Table-1: Search Types for Finding Packets

Search Type	Examples

Display filter	`not ip`
	`ip addr==192.168.0.1`
	`arp`

Hex value	`00:ff`
	`ff:ff`
	`00:AB:B1:f0`

String	`Workstation1`
	`UserB`
	`domain`

Table-1 shows examples of these search types.

Other options include the ability to select the window in which you want to search, the character set to use, and the search direction. You can extend the capability of your string searches by specifying the pane the search is per-formed in, setting the character set used, and making the search case sensitive. Once you've made your selections, enter your search criteria in the text box, and click Find to find the first packet that meets your criteria. To find the next matching packet, press CTRL-N; find the previous matching packet by pressing CTRL-B.

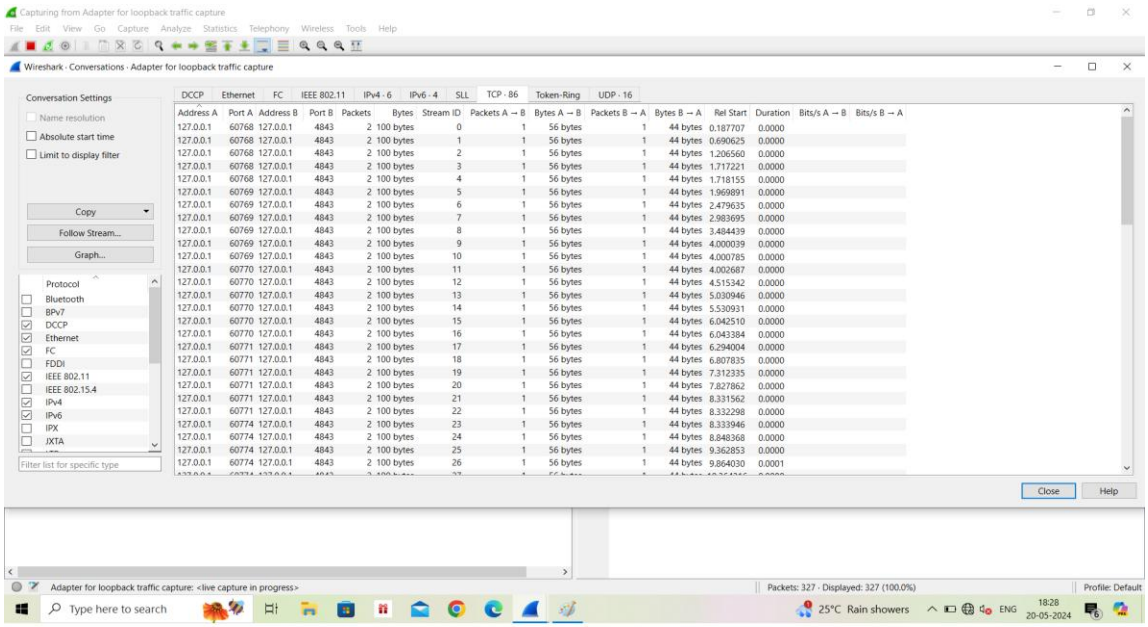


Figure 2

Setting Capture Options

Wireshark offers quite a few more capture options in the Capture Options dialog, shown in Figure-2. To open this dialog, choose Capture - Interfaces and click the Options button next to the interface on which you want to capture packets.

The Capture Options dialog has more bells and whistles than you can shake a stick at, all designed to give you more flexibility while capturing packets. It's divided into Capture,

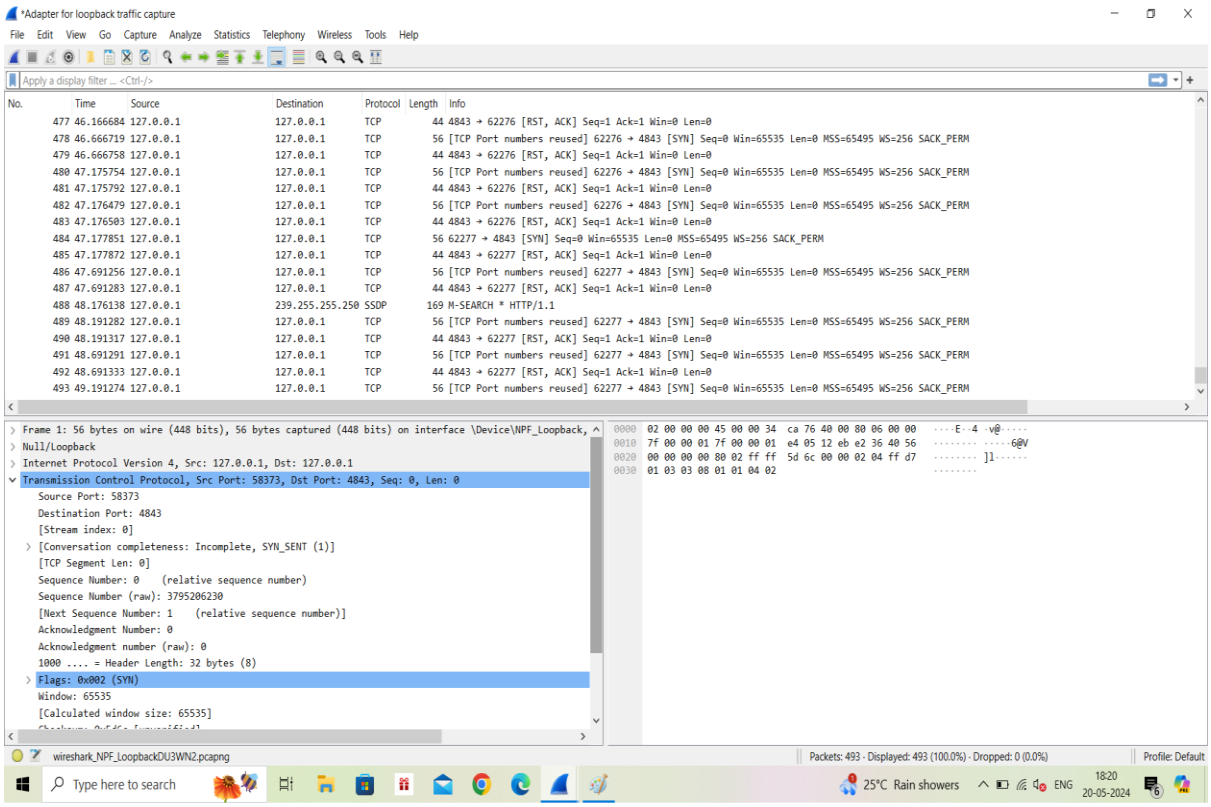
Capture Files, Stop Capture, Display Options, and Name Resolution sections, out of which only capture setting discussed here.

Capture Settings

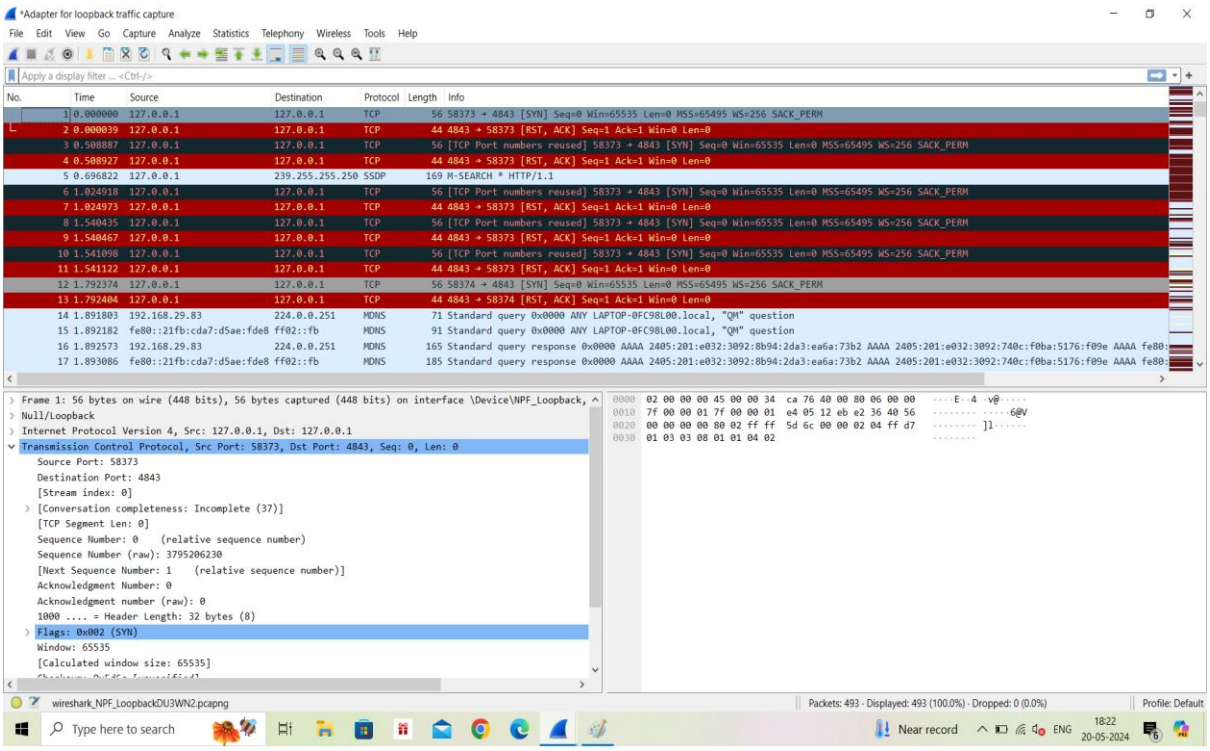
The Interface drop-down list in the Capture section is where you can select the network interface to configure. The left drop-down list allows you to specify whether the interface is local or remote, and the right drop-down list shows all available capture interfaces. The IP address of the interface you have selected is displayed directly below this drop-down list.

The three checkboxes on the left side of the dialog box allow you to enable or disable promiscuous mode (always enabled by default), capture packets in the currently experimental pcap-ng format, and limit the size of each capture packet by bytes.

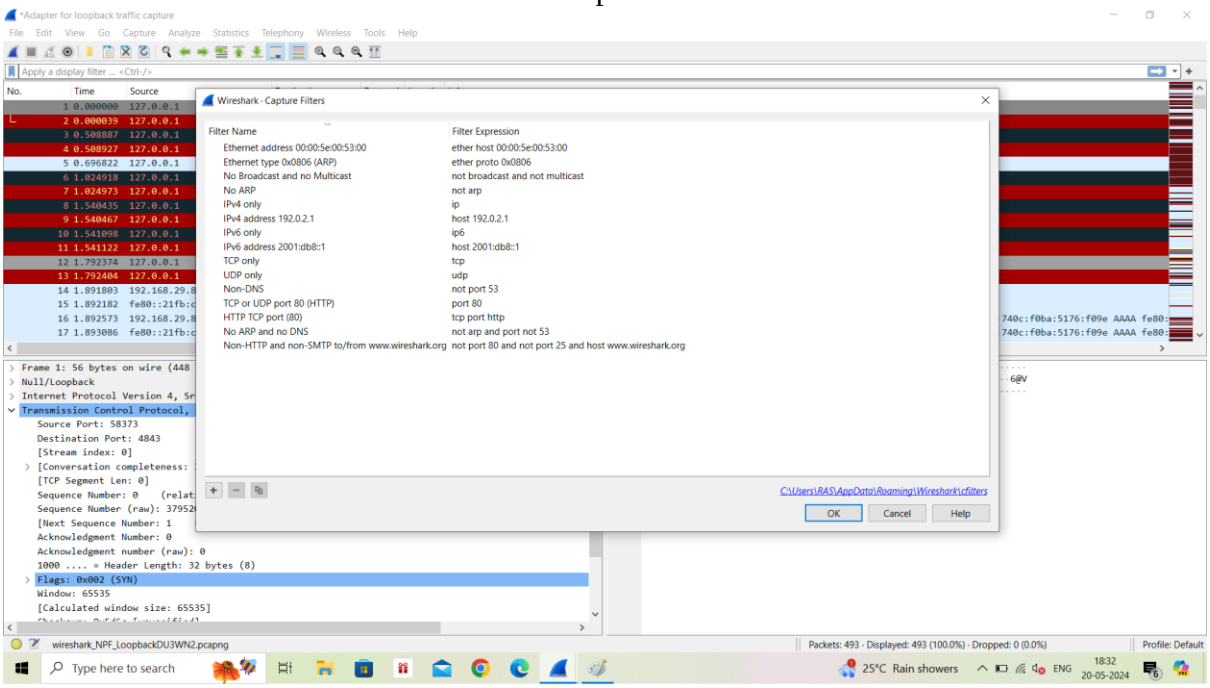
The buttons on the right side of the Capture section let you access wire-less and remote settings (as applicable). Beneath those is the buffer size option, which is available only on systems running Microsoft Windows. You can specify the amount of capture packet data that is stored in the kernel buffer before it is written to disk. (This is a value you won't normally modify unless you begin noticing that you are dropping a lot of packets.)



Capturing of packets in wireshark software



Colour differentiation in the packets in wireshark software



Packet filtering options in wireshark software

PACKET ANALYZING

It focuses on using packet analysis for network troubleshooting; a considerable amount of real-world packet analysis is done for security purposes. This could be the job of an intrusion analyst reviewing network traffic from potential intruders, or of a forensic investigator attempting to ascertain the extent of a malware infection on a compromised host.

Reconnaissance

The first step that an attacker takes is to perform in depth research on the target system. This step, commonly referred to as Foot printing, is often accomplished using various publicly available resources, such as the target company's website or Google. Once this research is completed, the attacker will typically begin scanning the IP address (or DNS name) of its target for open ports or running services. Result of scanning is that it tells the attacker on which ports the target is listening. Returning to our bank robber analogy, consider what would happen if the robber showed up at the bank with absolutely no knowledge of the building's physical layout. He would have no idea of how to gain access to the building, because he wouldn't know the weak points in its physical security.

SYN Scan

The type of scanning often done first against a system is a *TCP SYN scan*, also known as a *Stealth scan* or a *half-open scan*. A SYN scan is the most common type for several reasons:

- It is very fast and reliable.
- It is accurate on all platforms, regardless of TCP stack implementation.
- It is less noisy than other scanning techniques.

The TCP SYN scan relies on the three-way handshake process to deter mine which ports are open on a target host. The attacker sends a TCP SYN packet to a range of ports on the victim, as if trying to establish a channel for normal communication on the ports. Once this packet is received by the victim, one of a few things may happen, as shown in Figure.

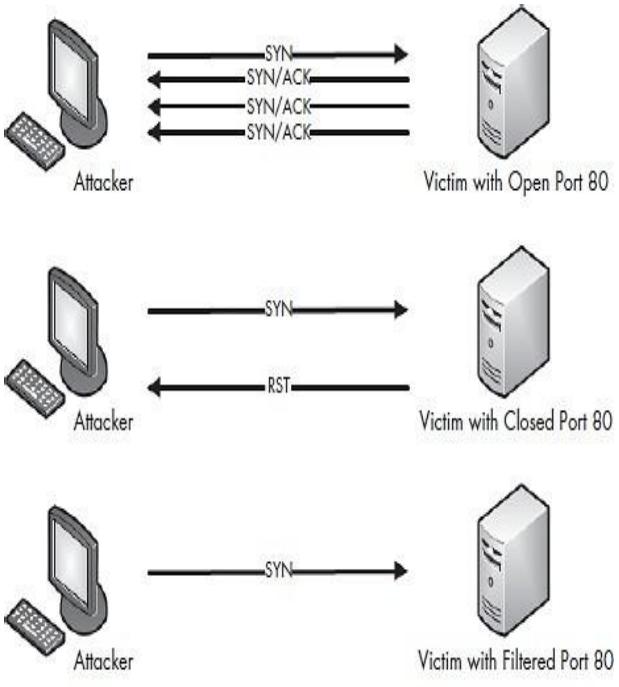
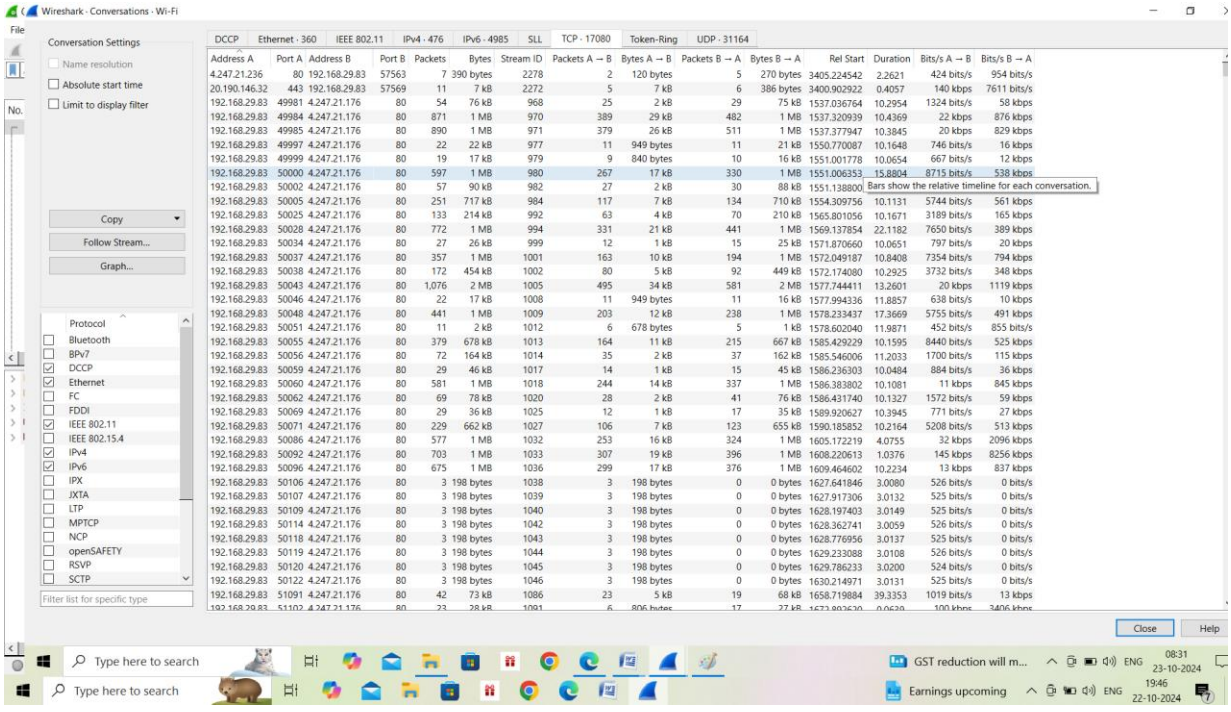


Figure-1: Possible results of a TCP SYN scan

If a service on the victim's machine is listening on a port that receives the SYN packet, it will reply to the attacker with a TCP SYN/ACK packet, the second part of the TCP handshake. Then the attacker knows that port is open and a service is listening on it. Under normal circumstances, a final TCP ACK would be sent in order to complete the connection handshake, but in this case, the attacker doesn't want that to happen, since he will not be communicating with the host further at this point. So, the attacker doesn't attempt to complete the TCP handshake.

If no service is listening on a scanned port, the attacker will not receive a SYN/ACK. Depending on the configuration of the victim's operating system the attacker could receive an RST packet in return, indicating that the port is closed. Alternatively, the attacker may receive no response at all. That could mean that the port is filtered by an intermediate device, such as a firewall or the host itself. On the other hand, it could just be that the response was lost in transit. This result typically indicates that the port is closed, but it's ultimately inconclusive.

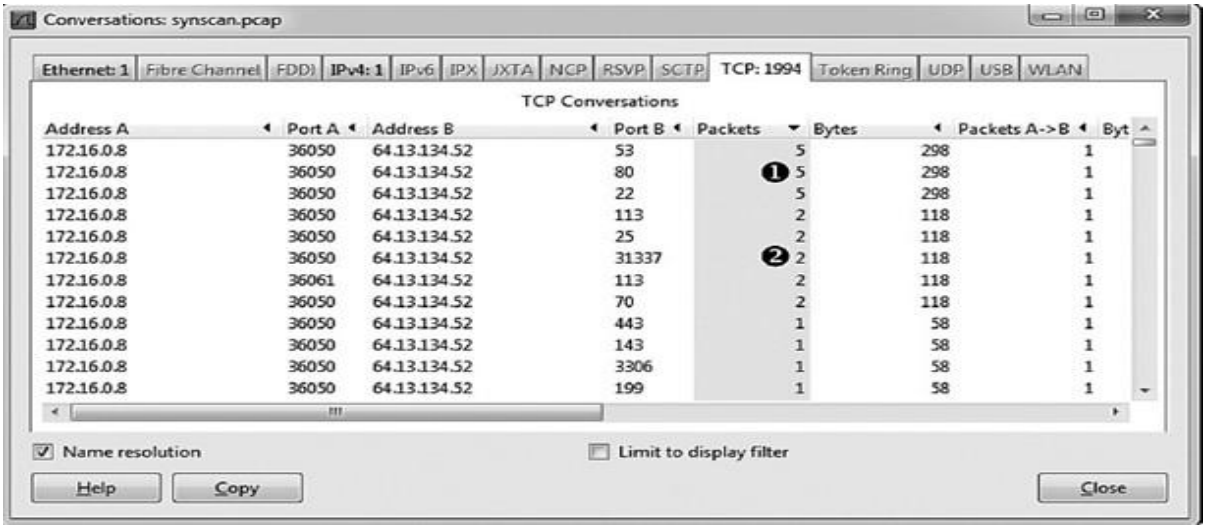
The scanning is occurring very quickly, so scrolling through the capture file is not the best way to find the response associated with each initial SYN packet. Several more packets might be sent before a response to the original packet is received. Fortunately, we can create filters to help us find the right traffic



Variety of TCP communication

Identifying Open and Closed Ports

After understanding the different types of response, a SYN scan can elicit, the next logical thought is to find a fast method of identifying which ports are open or closed. The answer lies within the conversations window once again.



In this window, you can sort the TCP conversations by packet number, with the highest values at the top by clicking the Packets column twice, as shown in Figure. Three scanned ports include five packets in each of their conversations [1]. We know that ports 53, 80, and 22 are open, because these five packets represent the initial SYN, the corresponding SYN/ACK, and the retransmitted SYN/ACKs from the victim.

For five ports, only two packets were involved in the communication [2]. The first is the initial SYN, and the second is the RST from the victim. This indicates that ports 113, 25, 31337, 113, and 70 are closed.

The remaining entries in the Conversations window include only one packet, meaning that the victim host never responded to the initial SYN. These remaining ports are most likely closed, but we're not sure.

Result:

Thus the working of captured packets has been analysed with filtering and scanning successfully.