Ex no: 01 Date:	LEARN TO USE COMMANDS LIKE TCPDUMP, NETSTAT, IFCONFIG, NSLOOKUP AND TRACE ROUTE. CAPTURE PING AND TRACE ROUTE PDUs USING A NETWORK PROTOCAL ANALYZER AND EXAMINE

AIM:

To learn and use basic networking commands such as tcpdump, netstat, ifconfig, nslookup, and traceroute. Also, to capture and analyze ping and traceroute PDUs using a network protocol analyzer (e.g., Wireshark).

REQUIREMENTS:

- OS: Linux (Ubuntu/Debian) or Windows with admin rights.
- Tools: Wireshark, Terminal/Command Prompt.
- Internet connectivity for traceroute/nslookup.

Part A: Basic Network Commands

1. **ifconfig (Linux) / ipconfig (Windows):**

- Displays IP address, subnet mask, MAC address, and interface details.

Linux:
ifconfig

Windows:
Cmd> ipconfig /all

2. **netstat:**

- Displays network connections, routing tables, interface statistics.

Example:
Cmd> netstat -an

3. **Nslookup:**

- Resolves domain name to IP address and vice versa.

Example:
Cmd> nslookup www.google.com

4. **traceroute / tracert**

OUTPUT:

Shows the route packets take to reach a destination host.
Linux:
Cmd> traceroute www.google.com

Windows:
Cmd> tracert www.google.com

5. **tcpdump**

Captures network traffic from terminal (Linux only).

Example:
sudo tcpdump -i eth0

Part B: Capture Ping and Traceroute Using Wireshark

Steps to Capture Packets:

STEP 1: Open **Wireshark**.
STEP 2: Select your **active interface** (e.g., Ethernet, Wi-Fi).
STEP 3: Click **Start Capture**.
STEP 4: In terminal/command prompt, run:

Ping:
ping www.google.com

Traceroute:
traceroute www.google.com # Linux
tracert www.google.com # Windows

STEP 5: Stop capture after some packets are received.

How to Analyze:

• **Filter for ICMP (ping)**:
icmp
Observe Echo request and Echo reply.

• **Filter for traceroute** (TTL Exceeded):
icmp or udp
Check Time-To-Live exceeded messages.

RESULT:

The basic networking commands were executed successfully. ICMP Echo and traceroute packets were captured and analyzed using Wireshark.

Ex no: 02 Date:	WRITE A HTTP WEB CLENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP

AIM:

To write a program in java to create a HTTP web client program to download a web page using TCP sockets.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Import the necessary Java packages, java.io.* for input/output streams and java.net.* for networking operations.

STEP 3: Create a Socket object to connect to the web server using TCP (port 80 for HTTP).

STEP 4: Prepare and send an HTTP GET request to the server for the desired web page (e.g.,"GET /index.html HTTP/1.1").

STEP 5: Use an OutputStreamWriter or PrintWriter to send the request through the socket.

STEP 6: Read the **response** from the server using BufferedReader (InputStreamReader).

STEP 7: Store the received data (HTML content or image) into a local file in the current working directory.

STEP 8: Display a message confirming successful download and show the file name.

STEP 9: Close all open resources — input stream, output stream, and socket connection.

STEP 10: Stop the program.

OUTPUT:

PROGRAM:

// Download.java

```
import java.io.*;
import java.net.URL;
public class Download
{
public static void main(String[] args) throws Exception
{
 try
{
String fileName = "digital_image_processing.jpg";
String website = "http://tutorialspoint.com/java_dip/images/"+fileName;
System.out.println("Downloading File From: " + website);
URL url = new URL(website);
InputStream inputStream = url.openStream();
OutputStream outputStream = new FileOutputStream(fileName);
byte[] buffer = new byte[2048];
int length = 0;
while ((length = inputStream.read(buffer)) != -1)
{
System.out.println("Buffer Read of length: " + length);
outputStream.write(buffer, 0, length);

}
inputStream.close();
outputStream.close();
}
catch(Exception e)
{
System.out.println("Exception: " + e.getMessage());
}
}
}
```

RESULT:

Thus created a program in java for a HTTP web client program to download a web page using TCP sockets is written and executed successfully.

Ex no: 3 a Date:	**APPLICATION USING TCP SOCKETS - ECHO CLIENT AND ECHO SERVER**

AIM:

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

ALGORITHM:

STEP 1: Start the Program
STEP 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.

STEP 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.
g)

STEP 4: Stop the program.

OUTPUT:

PROGRAM:

//Server.java

```
import java.net.*;
import java.lang.*;
import java.io.*;
public class Server
{
public static final int PORT = 4000;
public static void main( String args[])
{
ServerSocket sersock = null;
Socket sock = null;

try
{
sersock = new ServerSocket(PORT);
System.out.println("Server Started :"+sersock);
try
{
sock = sersock.accept();
System.out.println("Client Connected :"+ sock);
DataInputStream ins = new DataInputStream(sock.getInputStream());
System.out.println(ins.readLine());
PrintStream ios = new PrintStream(sock.getOutputStream());
ios.println("Hello from server");
ios.close();
sock.close();
}
catch(SocketException se)
{
System.out.println("Server Socket problem "+se.getMessage());
}
}
catch(Exception e)
{
System.out.println("Couldn't start " + e.getMessage()) ;
}
System.out.println(" Connection from : " + sock.getInetAddress());
}
}
```

//Client.java

```
import java.lang.*;
import java.io.*;
import java.net.*;
import java.net.InetAddress;
class client
{
public static void main(String args[])
{
Socket sock=null;
DataInputStream dis=null;
PrintStream ps=null;
System.out.println(" Trying to connect");
try
{
sock= new Socket(InetAddress.getLocalHost(),Server.PORT);
ps= new PrintStream(sock.getOutputStream());
ps.println(" Hi from client");
DataInputStream is = new DataInputStream(sock.getInputStream());
System.out.println(is.readLine());
}
catch(SocketException e)
{
System.out.println("SocketException " + e);
}
catch(IOException e)
{
System.out.println("IOException " + e);
}
Finally
{
try
{
}
sock.close();
catch(IOException ie)
{
System.out.println(" Close Error :" + ie.getMessage());
} } } }
```

RESULT:

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

Ex no: 3 b **Date:**	**APPLICATION USING TCP SOCKETS – CHAT**

AIM:

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

ALGORITHM:

STEP 1: Start the Program
STEP 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

STEP 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

STEP 4: Stop the program

OUTPUT:

PROGRAM:

//tcpchatserver.java

```
import java.io.*;
import java.net.*;
class tcpchatserver
{
public static void main(String args[])throws Exception
{
PrintWriter toClient;
BufferedReader fromUser, fromClient;
try
{
ServerSocket Srv = new ServerSocket(4000);
System.out.print("\nServer started\n");
Socket Clt = Srv.accept();
System.out.println("Client connected");
toClient = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(Clt.getOutputStream())), true);
fromClient = new BufferedReader(new InputStreamReader(Clt.getInputStream()));
fromUser = new BufferedReader(new InputStreamReader(System.in));
String CltMsg, SrvMsg;
while(true)
{
CltMsg= fromClient.readLine();
if(CltMsg.equals("end"))
break;
else
{
}
}
System.out.println("Server
: " +CltMsg);
System.out.print("Message to Client : ");
SrvMsg = fromUser.readLine();
toClient.println(SrvMsg);
System.out.println("\nClient Disconnected");
fromClient.close();
toClient.close();
fromUser.close();
Clt.close();
Srv.close();
}
catch (Exception E)
{
```

```
System.out.println(E.getMessage());
}
}
}
```

//tcpchatclient.java

```
import java.io.*;
import java.net.*;
class tcpchatclient
{
public static void main(String args[])throws Exception
{
Socket Clt;
PrintWriter toServer;
BufferedReader fromUser, fromServer;
try
{
Clt = new Socket(InetAddress.getLocalHost(),4000);
toServer = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(Clt.getOutputStream())), true);
fromServer = new BufferedReader(new InputStreamReader(Clt.getInputStream()));
fromUser = new BufferedReader(new InputStreamReader(System.in));
String CltMsg, SrvMsg;
System.out.println("Type \"end\" to Quit");
while (true)
{
System.out.print("Message to Server : ");
CltMsg = fromUser.readLine();
toServer.println(CltMsg);
if (CltMsg.equals("end"))
break;
SrvMsg = fromServer.readLine();
System.out.println("Client
: " + SrvMsg);
}
}
catch(Exception E)
{
System.out.println(E.getMessage());
} } }
```

RESULT:

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

Ex no: 04 Date:	SIMULATION OF DNS USING UDP SOCKETS

AIM:

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

ALGORITHM:

STEP 1: Start the Program.
STEP 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.

STEP 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.

STEP 4: Stop the program.

OUTPUT:

PROGRAM:

//dnsclient.java

```
import java.io.*;
import java.net.*;
public class dnsclient
{
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 = 8080;
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();
}
}
```

//dnsserver.java

```
import java.io.*;
import java.net.*;
public class dnsserver
{
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 = {"yahoo.com", "gmail.com","cricinfo.com","facebook.com"};
String[] ip = {"68.180.206.184", "209.85.148.19","80.168.92.140","69.63.189.16"};
System.out.println("Press Ctrl + C to Quit");
while (true)
{
DatagramSocket serversocket=new DatagramSocket(8080);
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();
}
}
}
```

RESULT:

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

OUTPUT:

Ex no: 05 Date:	USE A TOOL LIKE WIRESHARK TO CAPTURE PACKETS AND EXAMINE THE PACKETS

AIM:

To implement the code to capture packets and examine the packets using wireshark.

PROCEDURE:

STEP 1: A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.
STEP 2: Identifying and analyzing protocols.
STEP 3: Identifying source & destination of traffic.

PROGRAM:

```
import sys
from scapy.all import *
# Define the packet capturing
functiondef
packet_handler(packet):
print(packet.show())
# Capture packets on the network
interfacesniff(iface='eth0',
prn=packet_handler)
```

RESULT:

Thus the program was executed successfully using tool Wireshark to capture packet and examine the packet.

Ex no: 6 a Date:	SIMULATION OF ARP PROTOCOLS

AIM:

To write a program in java to simulate ARP protocols.

ALGORITHM:

STEP 1: Start the program
STEP 2: In Client

a. Start the program
b. Using socket connection is established between client and server.
c. Get the IP address to be converted into MAC address.
d. Send this IP address to server.
e. Server returns the MAC address to client.

STEP 3: In Server

a. Start the program
b. Accept the socket which is created by the client.
c. Server maintains the table in which IP and corresponding MAC addresses are stored.
d. Read the IP address which is send by the client.
e. Map the IP address with its MAC address and return the MAC address to client.

STEP 4: Stop the program.

OUTPUT:

PROGRAM:

//Clientarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp
{
public static void main(String args[])
{
try
{
BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
Socket clsct=new Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical address(IP):");
String str1=in.readLine();
dout.writeBytes(str1+'\n');
String str=din.readLine();
System.out.println("The Physical Address is: "+str);
clsct.close();
}
catch (Exception e)
{
System.out.println(e);
} } }
```

//Serverarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverarp
{
public static void main(String args[])
{
try
{
ServerSocket obj=new ServerSocket(139);
Socket obj1=obj.accept();
while(true)
{
DataInputStream din=new DataInputStream(obj1.getInputStream());
DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());
String str=din.readLine();
```

```
String ip[]={"165.165.80.80","165.165.79.1"};
String mac[]={"6A:08:AA:C2","8A:BC:E3:FA"};
for(int i=0;i<ip.length;i++)
{
if(str.equals(ip[i]))
{
dout.writeBytes(mac[i]+'\n');
break;
} }
obj.close();
} }
catch(Exception e)
{
System.out.println(e);
} } }
```

RESULT:

Thus a program in java simulated ARP protocols.

Ex no: 6 b Date:	SIMULATION OF RARP PROTOCOLS

AIM:

To write a program in java to simulate RARP protocols.

ALGORITHM:

STEP 1: Start the program.
STEP 2: In Client

a. Start the program.
b. Using socket connection is established between client and server.
c. Get the MAC address to be converted into IP address.
d. Send this MAC address to server.
e. Server returns the IP address to client.

STEP 3: In Server

a. Start the program
b. Accept the socket which is created by the client.
c. Server maintains the table in which IP and corresponding MAC addresses are stored.
d. Read the MAC address which is send by the client.
e. Map the MAC address with its MAC address and return the IP address to client.

STEP 4: Stop the program.

OUTPUT:

PROGRAM:

//clientrarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class clientrarp
{
public static void main(String args[]){
try {
DatagramSocket client = new DatagramSocket();
InetAddress addr = InetAddress.getByName("127.0.0.1");
byte[] sendByte = new byte[1204];
byte[] receiveByte = new byte[1024];
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter the Physical Address ");
String str = in.readLine();
sendByte = str.getBytes();
DatagramPacket sender = new DatagramPacket(sendByte,sendByte.length,addr,1309);
client.send(sender);
DatagramPacket receiver = new DatagramPacket(receiveByte,receiveByte.length);
client.receive(receiver);
String s = new String(receiver.getData());
System.out.println("The Logical Address is:" + s.trim());
client.close(); }
catch(Exception e) {
System.out.println(e);
} } }
```

//serverrarp.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class serverrarp
{
public static void main(String args[])
{
try
{
DatagramSocket server = new DatagramSocket(1309);
while(true){
byte[] sendByte = new byte[1204];
byte[] receiveByte = new byte[1204];
DatagramPacket receiver = new DatagramPacket(receiveByte,receiveByte.length);
```

```
server.receive(receiver);
String str = new String(receiver.getData());
String s = str.trim();
InetAddress addr = receiver.getAddress();
int port = receiver.getPort();
String ip[] = {"10.0.3.186"};
String mac[] = {"D4:3D:7E:12:A3:D9"};
for (int i = 0; i < ip.length; i++) {
if(s.equals(mac[i]))
{
sendByte = ip[i].getBytes();
DatagramPacket sender = new DatagramPacket(sendByte,sendByte.length,addr,port);
server.send(sender);
break;
}
}
break;
}
}catch(Exception e)
{
System.out.println(e);
}
}
}
```

RESULT:

Thus a program in java simulated RARP protocols.

Ex no: 7 a Date:	**NETWORK SIMULATION FOR TWO LANs SHARING FILE A ROUTER USING CISCO PACKET TRACER**

AIM:

To simulate two LANs connected via a router using Cisco Packet Tracer and configure IP addressing to enable communication between the two LANs.

TOOLS REQUIRED:

- Cisco Packet Tracer.
- Devices: 1 Router, 2 Switches, 4 PCs.

PROCEDURE:

STEP 1: Open Cisco Packet Tracer and create a new project.
STEP 2: Drag and drop the devices:
- Router (e.g., 2911)
- 2 Switches (2960)
- 4 PCs (2 PCs per LAN)

STEP 3: Connect the devices:
- PC1 & PC2 → Switch1
- PC3 & PC4 → Switch2
- Switch1 → Router (Fast Ethernet f0/0)
- Switch2 → Router (Fast Ethernet f0/1)

STEP 4: Assign IP addresses:
- LAN 1 (192.168.1.0/24):
    1. PC1 → 192.168.1.2 / 255.255.255.0 (GW: 192.168.1.1)
    2. PC2 → 192.168.1.3 / 255.255.255.0 (GW: 192.168.1.1)
    3. Router f0/0 → 192.168.1.1 / 255.255.255.0
- LAN 2 (192.168.2.0/24):
    1. PC3 → 192.168.2.2 / 255.255.255.0 (GW: 192.168.2.1)
    2. PC4 → 192.168.2.3 / 255.255.255.0 (GW: 192.168.2.1)
    3. Router f0/1 → 192.168.2.1 / 255.255.255.0

STEP 5: Configure Router Interfaces:

```
Router> enable
Router# configure terminal
Router(config)# interface fastEthernet 0/0
Router(config-if)# ip address 192.168.1.1 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface fastEthernet 0/1
```

OUTPUT:

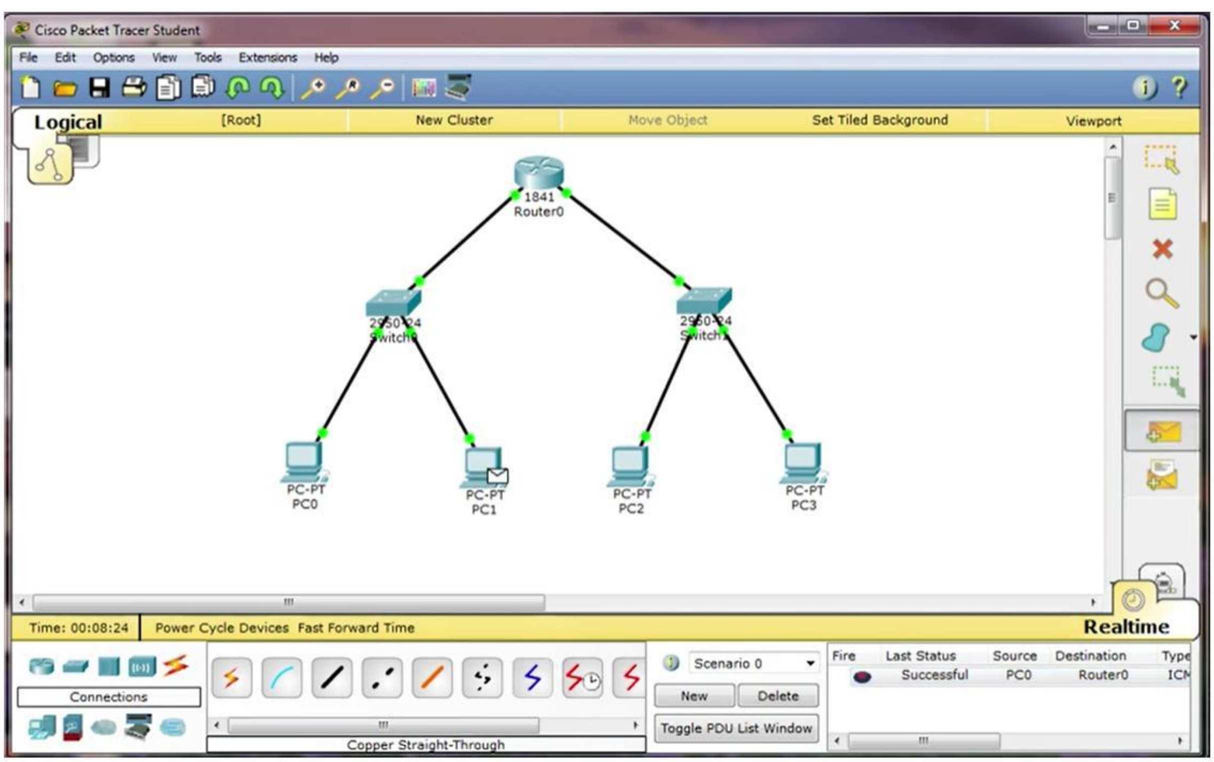
Router(config-if)# ip address 192.168.2.1 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# exit
Router# write

STEP 6: Verify Connectivity:

- From PC1 → Ping PC2 (within LAN1).
- From PC3 → Ping PC4 (within LAN2).
- From PC1 → Ping PC3 (across router).
- From PC2 → Ping PC4 (across router).

NETWORK DIAGRAM:

```
LAN 1 (192.168.1.0/24) LAN 2 (192.168.2.0/24)
----------------------------------------------------------
[PC1:192.168.1.2] [PC3:192.168.2.2]
[PC2:192.168.1.3] [PC4:192.168.2.3]
        |                |
   [Switch1]        [Switch2]
        |                |
|---(f0/0) Router (f0/1)---------|
192.168.1.1 192.168.2.1
```



RESULT:

Two LANs were successfully created and connected using a router in Cisco Packet Tracer. IP addresses were assigned, and routing was configured, enabling communication both within LANs and across LANs.

Ex no: 7 b Date:	**NETWORK SIMULATION FOR LAN WITH ROUTER TO ASSIGN IP ADDRESS USING CISCO PACKET TRACER**

AIM:

To simulate a LAN network with a router in Cisco Packet Tracer and configure the router to assign IP addresses to hosts using manual configuration and verify connectivity.

TOOLS REQUIRED:

- Cisco Packet Tracer.
- Devices: 1 Router, 1 Switch, 2 PCs, and 1 Server.

PROCEDURE:

STEP 1: Start Cisco Packet Tracer and create a new project.
STEP 2: Drag and drop the devices:

- Router (e.g., 2911)
- Switch (2960)
- 2 PCs (PC1, PC2)
- 1 Server

STEP 3: Connect the devices:

- PC1 → Switch (Copper Straight-Through cable)
- PC2 → Switch (Copper Straight-Through cable)
- Switch → Router (FastEthernet cable)
- Router → Server (FastEthernet cable)

STEP 4: Assign IP addresses manually:

- PC1 → 192.168.1.2 / 255.255.255.0 (Gateway: 192.168.1.1)
- PC2 → 192.168.1.3 / 255.255.255.0 (Gateway: 192.168.1.1)
- Server → 192.168.2.2 / 255.255.255.0 (Gateway: 192.168.2.1)

STEP 5: Configure Router interfaces:

- f0/0 → 192.168.1.1 / 255.255.255.0
- f0/1 → 192.168.2.1 / 255.255.255.06.

STEP 6: Verify connectivity:

- Use ping command from PC1 to Server.
- Use tracert to verify routing path.

COMMANDS USED:

Router CLI Configuration:

Router> enable
Router# configure terminal

OUTPUT:

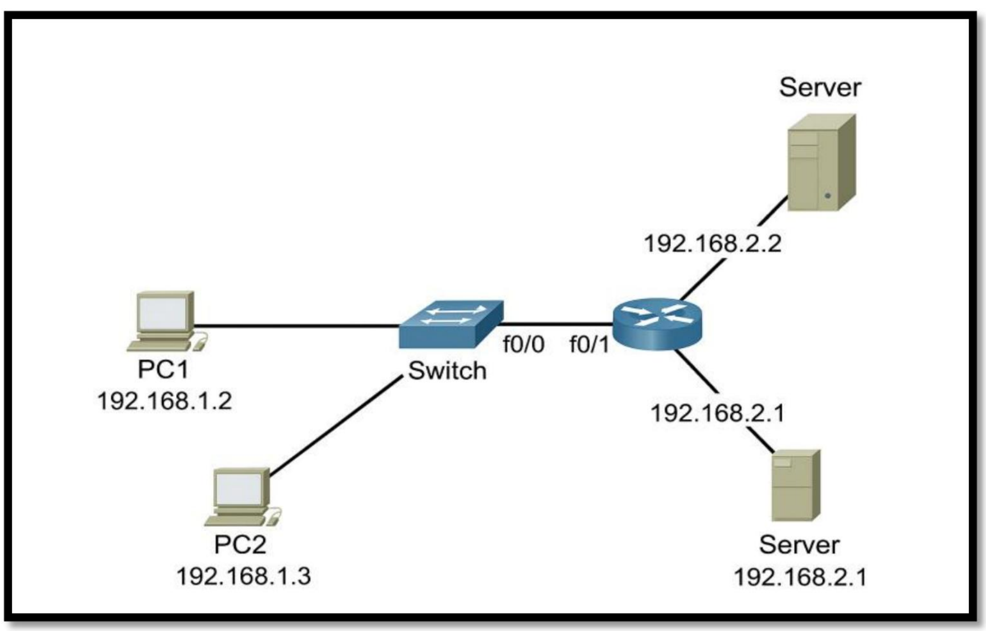
Router(config)# interface fastEthernet 0/0
Router(config-if)# ip address 192.168.1.1 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# interface fastEthernet 0/1
Router(config-if)# ip address 192.168.2.1 255.255.255.0
Router(config-if)# no shutdown
Router(config-if)# exit
Router(config)# exit
Router# write

PC Commands (in Command Prompt of Packet Tracer):

ping 192.168.1.3 // Ping between PCs
ping 192.168.2.2 // Ping Server
tracert 192.168.2.2 // Trace route to server

NETWORK DIAGRAM:

```
[PC1:192.168.1.2] [Server:192.168.2.2]
        |                    |
  [PC2:192.168.1.3]            |
        |                    |
   [Switch]  ----  [Router]  -----
       f0/0               f0/1
     192.168.1.1 192.168.2.1
```



RESULT:

A LAN network was successfully simulated in Cisco Packet Tracer with a router configured to assign and manage IP addresses for different networks. Connectivity between hosts and the server was verified using ping and tracert commands.

Ex no: 08 Date:	**STUDY OF TCP/UDP PERFORMANCE USING CISCO PACKET TRACER**

AIM:

To study and analyze the performance of TCP and UDP protocols using Cisco Packet Tracer simulation.

TOOLS REQUIRED:

- Cisco Packet Tracer.
- Devices: PCs, Switches, Router, and Servers.

PROCEDURE:

STEP 1: Open Cisco Packet Tracer and create a new workspace.
STEP 2: Place the following devices:
- 2 PCs (clients)
- 1 Server
- 1 Switch
- 1 Router (if multiple networks are required).

STEP 3: Connect the devices using Copper Straight-Through cables.
- PC1 → Switch
- PC2 → Switch
- Server → Switch
- Switch → Router (if inter-networking is required).

STEP 4: Assign IP Addresses:
- PC1 → 192.168.1.2 / 255.255.255.0 (Gateway: 192.168.1.1)
- PC2 → 192.168.1.3 / 255.255.255.0 (Gateway: 192.168.1.1)
- Server → 192.168.1.10 / 255.255.255.0 (Gateway: 192.168.1.1)
- Router Interface → 192.168.1.1 / 255.255.255.0

STEP 5: Configure Server Services:
- Enable Web Service (HTTP) → works with TCP.
- Enable TFTP Service → works with UDP.

STEP 6: On the client PCs:
- Use Web Browser to connect to the server's IP (HTTP → TCP).
- Use TFTP client tool (in Command Prompt) to transfer files (TFTP → UDP).

STEP 7: Capture packets using Simulation Mode in Packet Tracer.
- Observe TCP connection setup (3-way handshake) for HTTP.
- Observe connectionless transfer for UDP (TFTP).

STEP 8: Compare delay, packet reliability, and retransmissions for TCP vs. UDP.

OUTPUT:

COMMANDS USED (PC COMMAND PROMPT):

- ping 192.168.1.10 → Test connectivity.
- tftp -i 192.168.1.10 GET test.txt → Transfer file via UDP.
- Web Browser → http://192.168.1.10 → Access HTTP page via TCP.

NETWORK DIAGRAM:

- One Server connected to Switch
- Two PCs connected to same Switch
- Switch connected to Router (optional for multiple networks)

RESULT:

The performance of TCP and UDP protocols was successfully studied using Cisco Packet Tracer. TCP ensured reliable communication while UDP provided faster, connectionless communication.

Ex no: 9 a Date:	DISTANCE VECTOR ROUTING ALGORITHM

AIM:

To simulate and compare **Distance Vector (DV)** and **Link State (LS / Dijkstra)** routing algorithms using Java and produce routing tables (distance + next hop).

a)DISTANCE VECTOR (Bellman-Ford style)

ALGORITHM:

STEP 1: Start the program.

STEP 2: Define or read the network topology as a cost (adjacency) matrix, where

- Each element `cost[i][j]` represents the cost from node `i` to node `j`.
- Use `INF` (infinity) for no direct connection.

STEP 3: For each router, initialize its distance vector table**:**

- Distance to itself = 0
- Distance to directly connected neighbors = link cost
- Distance to others = ∞ (unknown at start)

STEP 4: For every router, send its distance vector to all directly connected neighbors.

STEP 5: On receiving the distance vector from a neighbor, update the routing table using the Bellman-Ford equation.

STEP 6: Repeat Steps 4 and 5 for all routers in the network until no further updates occur — this means convergence has been reached.

STEP 7: After convergence, display each router's routing table showing:

- Destination node
- Minimum distance (cost)
- Next hop

STEP 8: Compare and analyze the final routing tables to verify correctness.

STEP 9: Stop the program.

OUTPUT:

PROGRAM:

//DistanceVectorSimulation.java

```
import java.util.Arrays;
public class DistanceVectorSimulation {
static final int INF = 99999;
public static void main(String[] args) {
// Example cost matrix: change to test other topologies
int[][] cost = {
{0, 1, 4, 7},
{1, 0, 2, INF},
{4, 2, 0, 2},
{7, INF, 2, 0}
};
simulateDistanceVector(cost);
}
public static void simulateDistanceVector(int[][] cost) {
int n = cost.length;
int[][] dist = new int[n][n];
int[][] nextHop = new int[n][n];
// Initialization
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {

if (i == j) {
dist[i][j] = 0;
nextHop[i][j] = j;
} else if (cost[i][j] != INF) {
dist[i][j] = cost[i][j];
nextHop[i][j] = j;
} else {
dist[i][j] = INF;
nextHop[i][j] = -1;
}
}
}
System.out.println("Initial distance vectors:");
printTables(dist, nextHop);
boolean updated;
int iter = 0;
do {
iter++;
updated = false; System.out.println("\n--- Iteration " + iter + " ---");
```

```
for (int r = 0; r < n; r++) {
for (int nb = 0; nb < n; nb++) {
if (nb == r) continue;
if (cost[r][nb] == INF) continue; // not a neighbor
for (int dest = 0; dest < n; dest++) {
if (dist[nb][dest] == INF) continue;
int newCost = safeAdd(cost[r][nb], dist[nb][dest]);
if (newCost < dist[r][dest]) {
System.out.printf("Router %d: dist[%d->%d] updated %s
-> %d via %d\n",
r, r, dest,
(dist[r][dest]>=INF?"INF":dist[r][dest]), newCost, nb);
dist[r][dest] = newCost;
nextHop[r][dest] = nb;
updated = true;
} } } }
printTables(dist, nextHop);
} while (updated && iter < 100);
System.out.println("\nConverged after " + iter + " iterations. Final
tables:");
printTables(dist, nextHop);
}
private static int safeAdd(int a, int b) {
if (a >= INF || b >= INF) return INF;
long s = (long)a + (long)b;
return s >= INF ? INF : (int)s;
}
private static void printTables(int[][] dist, int[][] nextHop) {
int n = dist.length;
for (int
r = 0; r < n; r++) {
System.out.println("\nRouter " + r + " routing table:");
System.out.printf("%-12s %-10s %-8s\n", "Destination",
"Distance", "NextHop");
for (int d = 0; d < n; d++) {
String ds = (dist[r][d] >= INF) ? "INF" :
Integer.toString(dist[r][d]);
String nh = (nextHop[r][d] == -1) ? "-" :
Integer.toString(nextHop[r][d]);
System.out.printf("%-12d %-10s %-8s\n", d, ds, nh);
} } } }
```

RESULT:

Demonstrates DV exchange and convergence to shortest-path distances. Useful to show count to-infinity if you later change a link to INF and observe behavior.

Ex no: 9 b **Date:**	**LINK STATE ROUTING ALGORITHM**

AIM:

To simulate and compare **Distance Vector (DV)** and **Link State (LS / Dijkstra)** routing algorithms using Java and produce routing tables (distance + next hop).

ALGORITHM:

STEP 1: Start the program.

STEP 2: Represent the network as an adjacency (cost) matrix; each router knows all link costs (the full topology).

STEP 3: Each router constructs and maintains a link-state database by collecting Link-State Packets (LSPs) from every router (LSPs contain a router's directly connected neighbors and link costs).

STEP 4: Use flooding to distribute LSPs: when a router generates or receives a new LSP, it forwards that LSP to all neighbors (avoiding duplicates via sequence numbers/age).

STEP 5: Once the link-state database is complete and stable, choose a router as the **s**ource.

STEP 6: Run Dijkstra's algorithm at the source to compute the shortest path tree to all destinations:

STEP 7: Derive the routing table from the shortest-path tree:

- For each destination, follow predecessor[] back to the first hop from the source — that first hop is the next hop.
- Record the destination, minimum distance, and next hop in the routing table.

STEP 8: Periodically or on topology change, regenerate LSPs, flood them, update the link-state database, and recompute Dijkstra to maintain correct routes.

STEP 9: Stop the program.

OUTPUT:

PROGRAM

//LinkStateSimulation.java

```
import java.util.*;
public class LinkStateSimulation {
static final int INF = 99999;
public static void main(String[] args) {
int[][] cost = {
{0, 1, 4, 7},
{1, 0, 2, INF},
{4, 2, 0, 2},
{7, INF, 2, 0}
};
simulateLinkState(cost);
}
public static void simulateLinkState(int[][] cost) {
int n = cost.length;
for (int src = 0; src < n; src++) {
System.out.println("\nRunning Dijkstra from source router " +
src);
int[] dist = new int[n];
int[] prev = new int[n];
boolean[] visited = new boolean[n];
Arrays.fill(dist, INF);
Arrays.fill(prev, -1);

dist[src] = 0;
for (int i = 0; i < n; i++) {
int u = -1;
int min = INF;
for (int v = 0; v < n; v++) {
if (!visited[v] && dist[v] < min) {
min = dist[v];
u = v;
}
}
if (u == -1) break;
visited[u] = true;
for (int v = 0; v < n; v++) {
if (cost[u][v] >= INF) continue;
if (dist[u] + cost[u][v] < dist[v]) {
dist[v] = dist[u] + cost[u][v];
prev[v] = u;
}
}
```

```
}
System.out.printf("%-12s %-10s %-10s\n", "Destination",
"Distance", "NextHop");
for (int d = 0; d < n; d++) {
String ds = (dist[d] >= INF) ? "INF" :
Integer.toString(dist[d]);
String nh = "-";
if (d != src && dist[d] < INF) {
// find next hop by backtracking prev[]
int cur = d;
int prevNode = prev[cur];
while (prevNode != -1 && prevNode != src) {
cur = prevNode;
prevNode = prev[cur];
}
nh = (prevNode == -1 && cur == d) ? Integer.toString(d) :
Integer.toString(cur);
} else if (d == src) {
nh = Integer.toString(src);
}
System.out.printf("%-12d %-10s %-10s\n", d, ds, nh);
} } } }
```

RESULT:

Link-State (Dijkstra) computes global shortest paths quickly and produces consistent routing tables from each router’s perspective.

Ex no: 10 Date:	SIMULATION OF AN ERROR DETECTION CODE USING JAVA

AIM:

To implement and simulate Cyclic Redundancy Check (CRC) encoding and checking (error detection) in Java.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Let data be the original bit string to be transmitted and generator be the divisor polynomial bit string (e.g., 1101).

STEP 3: Determine the degree of the generator polynomial (m = length of generator). Append (m – 1) zeros to the end of the data bits to prepare for division.

STEP 4: Perform binary division (mod-2) of the appended data by the generator polynomial using XOR operation instead of subtraction. Obtain the remainder after division.

STEP 5: Form the transmitted codeword by appending the remainder to the original data bits.

STEP 6: At the receiver side, divide the received codeword by the same generator polynomial using mod-2 division.

STEP 7: If the remainder obtained is all zeros, it means no error detected during transmission. Otherwise, if the remainder is non-zero, it indicates an error has occurred.

STEP 8: Stop the program.

OUTPUT:

PROGRAM:

//CRC.java

```
import java.util.Scanner;
public class CRC {
// XOR for strings (skip first bit because division step expects it)
private static String xor(String a, String b) {
StringBuilder sb = new StringBuilder();
for (int i = 1; i < b.length(); i++) {
sb.append(a.charAt(i) == b.charAt(i) ? '0' : '1');
}
return sb.toString();
}
private static String mod2Div(String dividend, String divisor) {
int pick = divisor.length();
String tmp = dividend.substring(0, pick);
while (pick < dividend.length()) {
if (tmp.charAt(0) == '1') {
tmp = xor(divisor, tmp) + dividend.charAt(pick);
} else {
// use zero string of length pick
String zeros = String.format("%" + pick + "s", "").replace('
', '0');
tmp = xor(zeros, tmp) + dividend.charAt(pick);
}
pick++;
}
// Last step
if (tmp.charAt(0) == '1') {
tmp = xor(divisor, tmp);
} else {
String zeros = String.format("%" + pick + "s", "").replace(' ',
'0');
tmp = xor(zeros, tmp);
}
return tmp;
}
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.print("Enter Data bits (e.g., 101100): ");
String data = sc.nextLine().trim();
System.out.print("Enter Generator (e.g., 1101): ");
String generator = sc.nextLine().trim();
int m = generator.length();
String appended = data + "0".repeat(m - 1);
```

```
String remainder = mod2Div(appended, generator);
String codeword = data + remainder;
System.out.println("CRC Remainder: " + remainder);
System.out.println("Transmitted Codeword: " + codeword);
// Receiver check
System.out.print("Enter Received Codeword (simulate errors or paste
transmitted): ");
String received = sc.nextLine().trim();
String rem = mod2Div(received, generator);
boolean error = rem.contains("1");
if (error) {
System.out.println("Error detected in received data. Remainder: "
+ rem);
} else {
System.out.println("No error detected. Remainder: " + rem);
}
sc.close();
} }
```

RESULT:

CRC encoder and checker work: remainder appended to data yields codeword; receiver detects errors when remainder ≠ 0.