EX NO: 1
Date:

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Captureping and traceroute PDUs using a network protocol analyzer and examine.

AIM:

To learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute

1. Tcpdump

The tcpdump utility allows you to capture packets that flow within your network to assist in network troubleshooting. The following are several examples of using tcpdump with different options. Traffic is captured based on a specified filter.

Options	**Description**
-D	Print a list of network interfaces.
-I	Specify an interface on which to capture.
-c	Specify the number of packets to receive.
-v, -vv, -vvv	Increase the level of details (verbosity)
-w	Write captured data to a file.
-r	Read captured data from a file.

Many other options and arguments can be used with tcpdump. The following are some specific examples of the power of the tcpdump utility.

1. Display traffic between 2 hosts

To display all traffic between two hosts (represented by variables host1 and host2): # tcpdump host host1 and host2

2. Display traffic from a source or destination host only

To display traffic from only a source (src) or destination (dst) host: # tcpdump src host

```
# tcpdump dst host
```

3. Display traffic for a specific protocol

Provide the protocol as an argument to display only traffic for a specific protocol, for example tcp, udp, icmp, arp:

```
# tcpdump protocol
```

For example to display traffic only for the tcp traffic:

```
# tcpdump tcp
```

4. Filtering based on source or destination port

To filter based on a source or destination port:

```
# tcpdump src port ftp
# tcpdump dst port http
```

2. Netstat

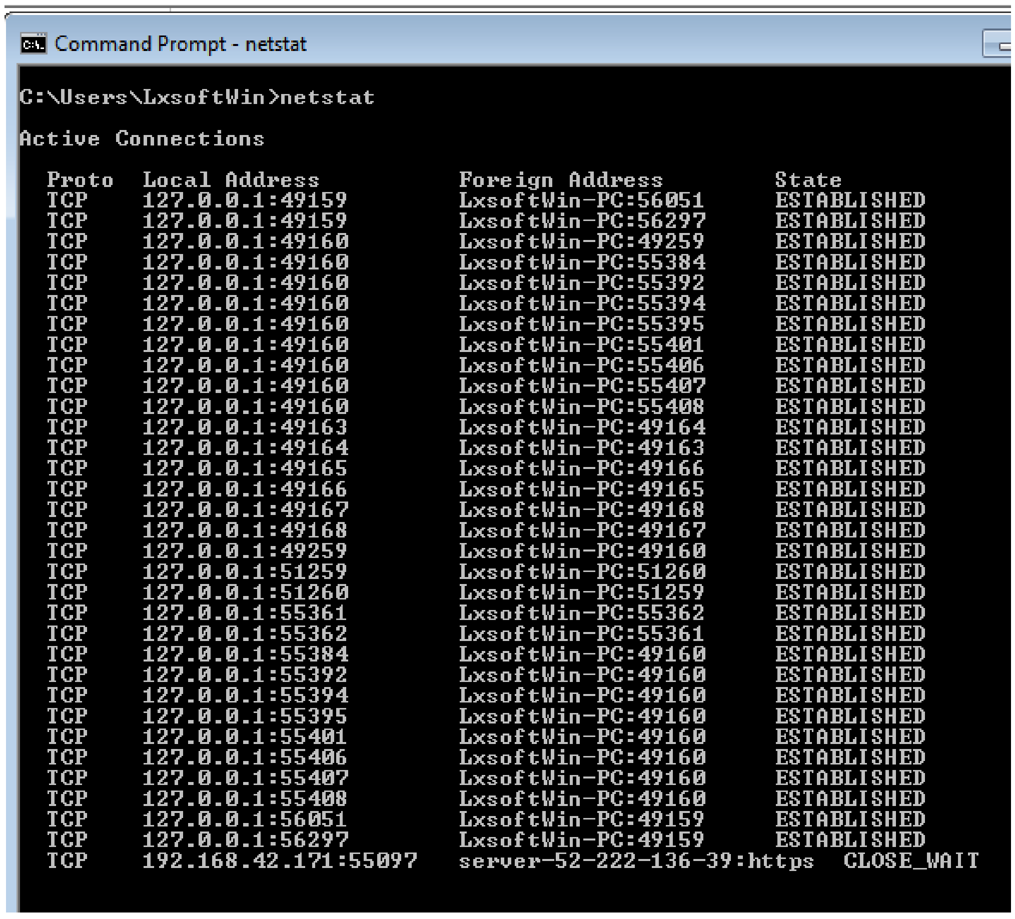
Netstat is a common command line TCP/IP networking available in most versions of Windows, Linux, UNIX and other operating systems. Netstat provides information and statistics about protocols in use and current TCP/IP network connections. The Windows help screen (analogous to a Linux or UNIX for netstat 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-known executables host multiple independent components, and in these cases the sequence of components involved in creating the connection or listening port is displayed. In this case the executable name is in [] at the bottom, on top is the component 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	Shows connections for the protocol specified by proto; proto may be any of: 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 -p option may be used to specify a subset 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.



```
C:\Users\LxsoftWin>netstat

Active Connections

  Proto  Local Address          Foreign Address        State
  TCP    127.0.0.1:49159        LxsoftWin-PC:56051     ESTABLISHED
  TCP    127.0.0.1:49159        LxsoftWin-PC:56297     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:49259     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55384     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55392     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55394     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55395     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55401     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55406     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55407     ESTABLISHED
  TCP    127.0.0.1:49160        LxsoftWin-PC:55408     ESTABLISHED
  TCP    127.0.0.1:49163        LxsoftWin-PC:49164     ESTABLISHED
  TCP    127.0.0.1:49164        LxsoftWin-PC:49163     ESTABLISHED
  TCP    127.0.0.1:49165        LxsoftWin-PC:49166     ESTABLISHED
  TCP    127.0.0.1:49166        LxsoftWin-PC:49165     ESTABLISHED
  TCP    127.0.0.1:49167        LxsoftWin-PC:49168     ESTABLISHED
  TCP    127.0.0.1:49168        LxsoftWin-PC:49167     ESTABLISHED
  TCP    127.0.0.1:49259        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:51259        LxsoftWin-PC:51260     ESTABLISHED
  TCP    127.0.0.1:51260        LxsoftWin-PC:51259     ESTABLISHED
  TCP    127.0.0.1:55361        LxsoftWin-PC:55362     ESTABLISHED
  TCP    127.0.0.1:55362        LxsoftWin-PC:55361     ESTABLISHED
  TCP    127.0.0.1:55384        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55392        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55394        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55395        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55401        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55406        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55407        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:55408        LxsoftWin-PC:49160     ESTABLISHED
  TCP    127.0.0.1:56051        LxsoftWin-PC:49159     ESTABLISHED
  TCP    127.0.0.1:56297        LxsoftWin-PC:49159     ESTABLISHED
  TCP    192.168.42.171:55097   server-52-222-136-39:https  CLOSE_WAIT
```

Syntax

```
netstat [-a] [-e] [-n] [-o] [-p Protocol] [-r] [-s] [Interval]
```

Parameters

-a	Used without displays active TCP connections. parametersDisplays all active TCP connections and the TCP and UDP ports on which the computer is listening.
-e	Displays Ethernet statistics, such as the number of bytes and packets sent and received. This parameter can be combined with -s.
-n	Displays active TCP connections, however, addresses and port numbers are expressed numerically and no attempt is made to determine names
.	
-o	Displays active TCP connections and includes the process ID (PID) for each connection. You can find the application based on the PID on the Processes tab in Windows Task Manager. This parameter can be combined with -a, -n, and -p.
-P	Shows connections for the protocol specified by Protocol. In this case, the Protocol can be tcp, udp, tcpv6, or udpv6. If this parameter is used with -s to display statistics by protocol, Protocol can be tcp, udp, icmp, ip, tcpv6, udpv6, icmpv6, or ipv6.
-s	Displays statistics by protocol. By default, statistics are shown for the TCP, UDP, ICMP, and IP protocols. If the IPv6 protocol for Windows XP is installed, statistics are shown for the TCP over IPv6, UDP over IPv6, ICMPv6, and IPv6 protocols. The -p parameter can be used to specify a set of protocols.
-r	Displays the contents of the IP routing table. This is equivalent to the route print command.Redisplays the selected information every Interval seconds. Press CTRL+C to Interval stop the redisplay. If this parameter is omitted, netstat prints the selected information only once.
/?	- Displays help at the command prompt.

3. Ifconfig

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. I**pconfig** replaced the older winipcfg utility.

Using ipconfig

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 adapters.

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 the full TCP/IP configuration for all adapters. Without this. parameter, ipconfig displays only the IP address, subnet mask, and default gateway values for each adapter. Adapters can represent physical interfaces, such as installed network adapters, or logical interfaces, such as dial-up connections.
/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 ID is not specified, the current class ID is removed.

Examples:

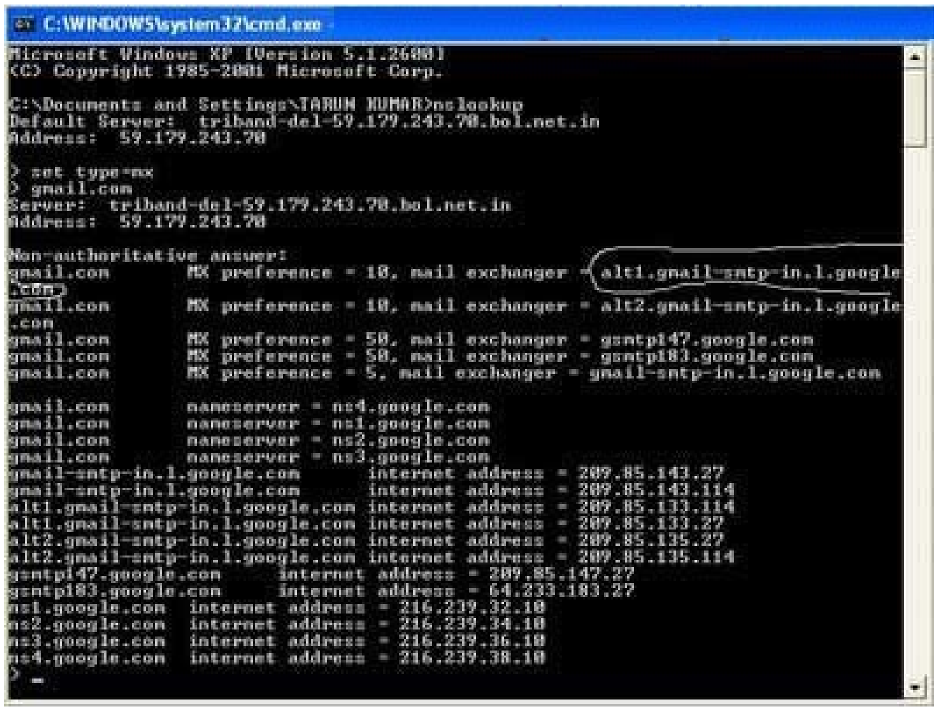
Ipconfig To display the basic TCP/IP configuration for all adapters

ipconfig /all To display the full TCP/IP configuration for all adapters

ipconfig /renew "Local Area Connection"	To renew a DHCP-assigned IP address configuration for only the Local Area Connection adapter
ipconfig /flushdns	To flush the DNS resolver cache when troubleshooting DNS name resolution problems
ipconfig /showclassid Local	To display the DHCP class ID for all adapters with names that start with Local
ipconfig /setclassid "Local Area Connection" TEST	To set the DHCP class ID for the Local Area Connection adapter to TEST

4.<u>Nslookup</u>

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.



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\TARUN KUMAR>nslookup
Default Server:  triband-del-59.179.243.70.bol.net.in
Address:  59.179.243.70

> set type=mx
> gmail.com
Server:  triband-del-59.179.243.70.bol.net.in
Address:  59.179.243.70

Non-authoritative answer:
gmail.com       MX preference = 10, mail exchanger = alt1.gmail-smtp-in.l.google
.com
gmail.com       MX preference = 10, mail exchanger = alt2.gmail-smtp-in.l.google
.com
gmail.com       MX preference = 50, mail exchanger = gsmtp147.google.com
gmail.com       MX preference = 50, mail exchanger = gsmtp183.google.com
gmail.com       MX preference = 5, mail exchanger = gmail-smtp-in.l.google.com

gmail.com       nameserver = ns4.google.com
gmail.com       nameserver = ns1.google.com
gmail.com       nameserver = ns2.google.com
gmail.com       nameserver = ns3.google.com
gmail-smtp-in.l.google.com      internet address = 209.85.143.27
gmail-smtp-in.l.google.com      internet address = 209.85.143.114
alt1.gmail-smtp-in.l.google.com internet address = 209.85.133.114
alt1.gmail-smtp-in.l.google.com internet address = 209.85.133.27
alt2.gmail-smtp-in.l.google.com internet address = 209.85.135.27
alt2.gmail-smtp-in.l.google.com internet address = 209.85.135.114
gsmtp147.google.com     internet address = 209.85.147.27
gsmtp183.google.com     internet address = 64.233.183.27
ns1.google.com  internet address = 216.239.32.10
ns2.google.com  internet address = 216.239.34.10
ns3.google.com  internet address = 216.239.36.10
ns4.google.com  internet address = 216.239.38.10
> _
```

5.traceroute

Traceroute is a network diagnostic tool used to track the pathway taken by a packet on an IP network from source to destination. Traceroute also records the time taken for each hop the packet makes during its route to the destination.

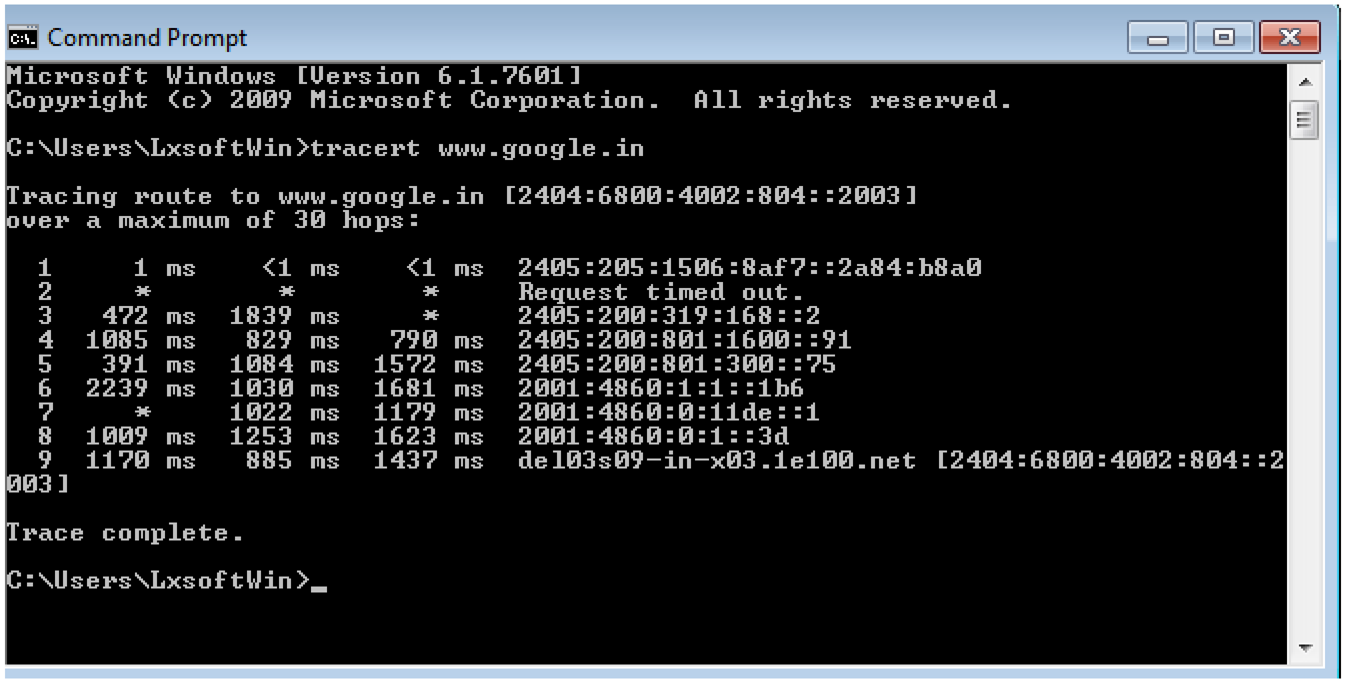
Traceroute uses Internet Control Message Protocol (ICMP) echo packets with variable time to live (TTL) values. The response time of each hop is calculated. To guarantee accuracy, each hop is queried multiple times (usually three times) to better measure the response of that particular hop.

tracert www.google.com

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.

```
Tracing route to www.l.google.com [209.85.225.104]
over a maximum of 30 hops:
1 <1 ms <1 ms <1 ms 10.1.0.1
2 35 ms 19 ms 29 ms 98.245.140.1
3 11 ms 27 ms 9 ms te-0-3.dnv.comcast.net [68.85.105.201]
...
13 81 ms 76 ms 75 ms 209.85.241.37
14 84 ms 91 ms 87 ms 209.85.248.102
15 76 ms 112 ms 76 ms iy-f104.1e100.net [209.85.225.104]
Trace complete.
```

tracert -j 10.12.0.1 10.29.3.1 10.1.44.1 www.google.com



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\LxsoftWin>tracert www.google.in

Tracing route to www.google.in [2404:6800:4002:804::2003]
over a maximum of 30 hops:

  1     1 ms    <1 ms    <1 ms  2405:205:1506:8af7::2a84:b8a0
  2     *        *        *     Request timed out.
  3   472 ms  1839 ms     *     2405:200:319:168::2
  4  1085 ms   829 ms   790 ms  2405:200:801:1600::91
  5   391 ms  1084 ms  1572 ms  2405:200:801:300::75
  6  2239 ms  1030 ms  1681 ms  2001:4860:1:1::1b6
  7     *     1022 ms  1179 ms  2001:4860:0:11de::1
  8  1009 ms  1253 ms  1623 ms  2001:4860:0:1::3d
  9  1170 ms   885 ms  1437 ms  del03s09-in-x03.1e100.net [2404:6800:4002:804::2
003]

Trace complete.

C:\Users\LxsoftWin>_
```

RESULT:

The networking commands like commands like tcpdump, netstat, ifconfig, nslookup and traceroute was executed successfully.

Ex No: 2 Write a HTTP web client program to download a web page using sockets.

Date:

AIM:

To write a java program for socket for HTTP for web page upload and download .

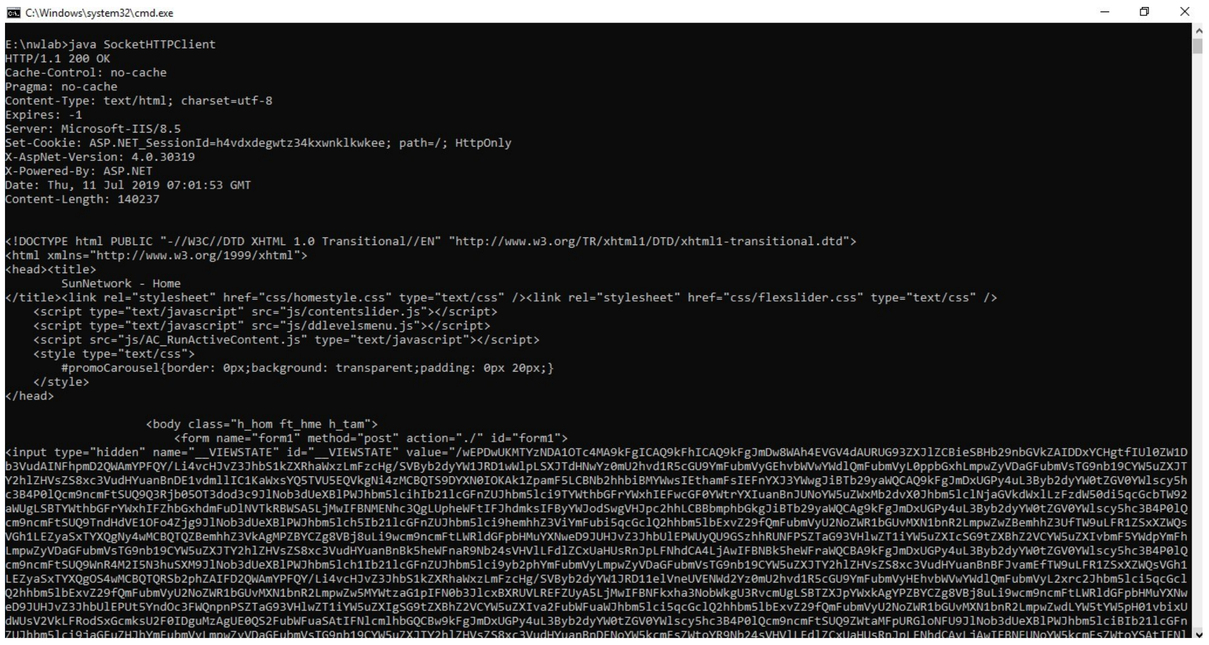
ALGORITHM

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

PROGRAM:

```
import java.io.*;
import java.net.*;
public class SocketHTTPClient
{

public static void main(String[] args)

{

String hostName = "www.sunnetwork.in";

int portNumber = 80;

try

{

Socket socket = new Socket(hostName, portNumber);
PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
BufferedReader in =new BufferedReader(new InputStreamReader(socket.getInputStream()));
System.out.println("GET / HTTP/1.1\nHost: www.sunnetwork.in\n\n");
String inputLine;
while ((inputLine = in.readLine()) != null)
{
System.out.println(inputLine);
}
}
catch (UnknownHostException e)
{
System.err.println("Don't know about host " + hostName);
System.exit(1);
}
catch (IOException e)
{
System.err.println("Couldn't get I/O for the connection to " + hostName);
System.exit(1);
}
}
}
```

OUTPUT



C:\Windows\system32\cmd.exe

```
E:\nwlab>java SocketHTTPClient
HTTP/1.1 200 OK
Cache-Control: no-cache
Pragma: no-cache
Content-Type: text/html; charset=utf-8
Expires: -1
Server: Microsoft-IIS/8.5
Set-Cookie: ASP.NET_SessionId=h4vdxdegwtz34kxwnklkwkee; path=/; HttpOnly
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Thu, 11 Jul 2019 07:01:53 GMT
Content-Length: 140237


<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>
        SunNetwork - Home
</title><link rel="stylesheet" href="css/homestyle.css" type="text/css" /><link rel="stylesheet" href="css/flexslider.css" type="text/css" />
    <script type="text/javascript" src="js/contentslider.js"></script>
    <script type="text/javascript" src="js/ddlevelsmenu.js"></script>
    <script src="js/AC_RunActiveContent.js" type="text/javascript"></script>
    <style type="text/css">
        #promoCarousel{border: 0px;background: transparent;padding: 0px 20px;}
    </style>
</head>

                    <body class="h_hom ft_hme h_tam">
                        <form name="form1" method="post" action="./" id="form1">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTYzNDA1OTc4MA9kFgICAQ9kFhICAQ9kFgJmDw8WAh4EVGV4dAURUG93ZXJlZCBieSBHb29nbGVkZAIDDxYCHgtfIUl0ZW1D
b3VudAINFhpmD2QWAmYPFQY/Li4vcHJvZ3JhbS1kZXRhaWxzLmFzcHg/SVByb2dyYW1JRD1wWlpLSXJTdHNwYz0mU2hvd1R5cGU9YmFubmVyGEhvbWVwYWdlQmFubmVyL0ppbGxhLmpwZyVDaGFubmVsTG9nb19CYW5uZXJT
Y2hlZHVsZS8xc3VudHYuanBnDE1vdmllIC1KaWxsYQ5TVU5EQVkgNi4zMCBQTS9DYXN0IOKAk1ZpamF5LCBNb2hhbiBMYWwsIEthamFsIEFnYXJ3YWwgJiBTb29yaWQCAQ9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5h
c3B4P0lQcm9ncmFtSUQ9Q3Rjb05OT3dod3c9JlNob3dUeXBlPWJhbm5lcihIb21lcGFnZUJhbm5lci9TYWthbGFrYWxhIEFwcGF0YWtrYXIuanBnJUNoYW5uZWxMb2dvX0Jhbm5lclNjaGVkdWxlLzFzdW50di5qcGcbTW92
aWUgLSBTYWthbGFrYWxhIFZhbGxhdmFuDlNVTkRBWSA5LjMwIFBNMENhc3QgLUpheWFtIFJhdmksIFByYWJodSwgVHJpc2hhLCBBbmphbGkgJiBTb29yaWQCAg9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQ
cm9ncmFtSUQ9TndHdVE1OFo4Zjg9JlNob3dUeXBlPWJhbm5lch5Ib21lcGFnZUJhbm5lci9hemhhZ3ViYmFubi5qcGclQ2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZwZBemhhZ3UfTW9uLFR1ZSxXZWQs
VGh1LEZyaSxTYXQgNy4wMCBQTQZBemhhZ3VkAgMPZBYCZg8VBj8uLi9wcm9ncmFtLWRldGFpbHMuYXNweD9JUHJvZ3JhbUlEPWUyQU9GSzhhRUNFPSZTaG93VHlwZT1iYW5uZXIcSG9tZXBhZ2VCYW5uZXIvbmF5YWdpYmFh
LmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnBk5heWFnaR9Nb24sVHVlLFdlZCxUaHUsRnJpLFNhdCA4LjAwIFBNBk5heWFraWQCBA9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQ
cm9ncmFtSUQ9WnR4M2I5N3huSXM9JlNob3dUeXBlPWJhbm5lch1Ib21lcGFnZUJhbm5lci9yb2phYmFubmVyLmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnBFJvamEfTW9uLFR1ZSxXZWQsVGh1
LEZyaSxTYXQgOS4wMCBQTQRSb2phZAIFD2QWAmYPFQY/Li4vcHJvZ3JhbS1kZXRhaWxzLmFzcHg/SVByb2dyYW1JRD11elVneUVENWd2Yz0mU2hvd1R5cGU9YmFubmVyHEhvbWVwYWdlQmFubmVyL2xrc2Jhbm5lci5qcGcl
Q2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZw5MYWtzaG1pIFN0b3JlcxBXRUVLREFZUyA5LjMwIFBNFkxha3NobWkgU3RvcmUgLSBTZXJpYWxkAgYPZBYCZg8VBj8uLi9wcm9ncmFtLWRldGFpbHMuYXNw
eD9JUHJvZ3JhbUlEPUt5YndOc3FWQnpnPSZTaG93VHlwZT1iYW5uZXIgSG9tZXBhZ2VCYW5uZXIva2FubWFuaWJhbm5lci5qcGclQ2hhbm5lbExvZ29fQmFubmVyU2NoZWR1bGUvMXN1bnR2LmpwZwdLYW5tYW5pH01vbixU
dWUsV2VkLFRodSxGcmksU2F0IDguMzAgUE0QS2FubWFuaSAtIFNlcmlhbGQCBw9kFgJmDxUGPy4uL3Byb2dyYW0tZGV0YWlscy5hc3B4P0lQcm9ncmFtSUQ9ZWtaMFpURGloNFU9JlNob3dUeXBlPWJhbm5lciBIb21lcGFn
ZUJhbm5lci9jaGFuZHJhYmFubmVyLmpwZyVDaGFubmVsTG9nb19CYW5uZXJTY2hlZHVsZS8xc3VudHYuanBnDENoYW5kcmFsZWtoYR9Nb24sVHVlLFdlZCxUaHUsRnJpLFNhdCAyLjAwIFBNFUNoYW5kcmFsZWtoYSAtIFNl
```

RESULT

Thus the program for creating sockets for HTTP web page to download was implemented.

Ex No: 3(a) **(a) Applications using TCP sockets like Echo client and Echo server**

Date:

AIM

To write a java program for applications using TCP sockets like Echo client and Echo server

ALGORITHM

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

PROGRAM :

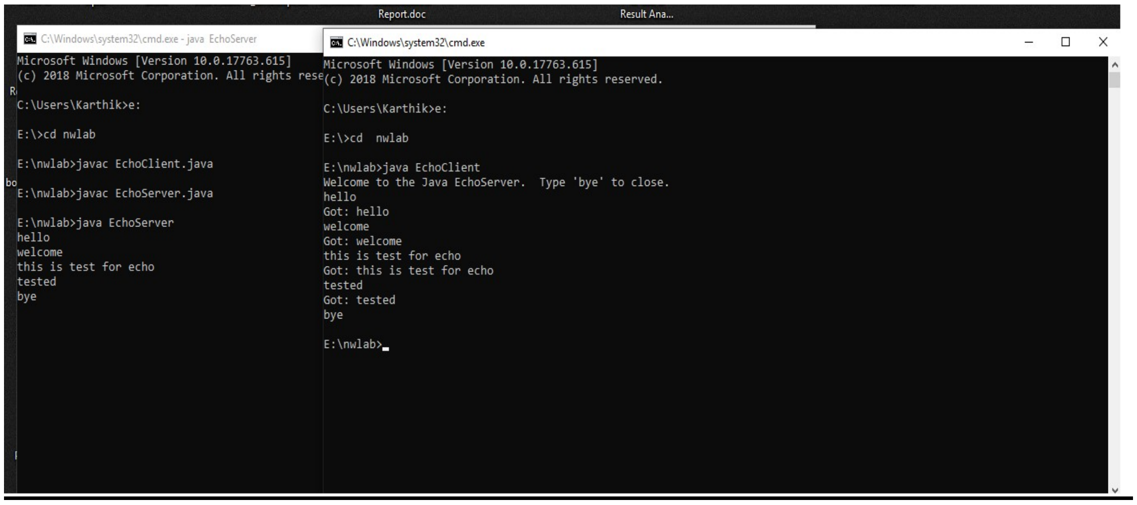
EchoServer.Java:

```
import java.io.*;
import java.net.*;
public class EchoServer
{
public EchoServer(int portnum)
{
try
{
 server = new ServerSocket(portnum);
}
catch (Exception err)
{
System.out.println(err);
}
}
public void serve()
{
try
{
while (true)
{
Socket client = server.accept();
BufferedReader r = new BufferedReader(new InputStreamReader(client.getInputStream()));
PrintWriter w = new PrintWriter(client.getOutputStream(),true);
System.out.println("("Welcome to the Java EchoServer. Type 'bye'close.");
String line;
do
{
line = r.readLine();
if ( line != null )
w.println("Got: "+ line);
System.out.println (line);
}
while ( !line.trim().equals("bye") );
client.close();
}
}
catch (Exception err)
{
System.err.println(err);
}
}
public static void main(String[] args)
{
EchoServer s = new EchoServer(9999);
s.serve();
}
private ServerSocket server;
}
```

EchoClient.java :

```
import java.io.*;
import java.net.*;
public class EchoClient
{
public static void main(String[] args)
{
 try
{
 Socket s = new Socket("127.0.0.1", 9999);
BufferedReader r = new BufferedReader(new InputStreamReader(s.getInputStream()));
PrintWriter w = new PrintWriter(s.getOutputStream(), true);
BufferedReader con = new BufferedReader(new InputStreamReader(System.in));
String line;
do
{
line = r.readLine();
if ( line != null )
System.out.println(line);
line = con.readLine();
w.println(line);
}
while ( !line.trim().equals("bye") );
}
catch (Exception err)
{
System.err.println(err);
}
}
}
```

OUTPUT:



```
Microsoft Windows [Version 10.0.17763.615]
(c) 2018 Microsoft Corporation. All rights rese

C:\Users\Karthik>e:

E:\>cd nwlab

E:\nwlab>javac EchoClient.java

E:\nwlab>javac EchoServer.java

E:\nwlab>java EchoServer
hello
welcome
this is test for echo
tested
bye
```

```
Microsoft Windows [Version 10.0.17763.615]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Karthik>e:

E:\>cd  nwlab

E:\nwlab>java EchoClient
Welcome to the Java EchoServer.  Type 'bye' to close.
hello
Got: hello
welcome
Got: welcome
this is test for echo
Got: this is test for echo
tested
Got: tested
bye

E:\nwlab>
```

RESULT:

Thus the program echo client and echo server was implemented successfully.

Ex No: 3(b) **(b) Chat**

Date:

AIM:

To implement a CHAT application using TCP Socket.

Algorithm:

1. It uses TCP socket communication .We have a server as well as a client.
2. Both can be run in the same machine or different machines. If both are running in the machine, the address to be given at the client side is local host address.
3. If both are running in different machines, then in the client side we need to specify the ip address of machine in which server application is running.

Program

Chatserver.java:

```
import java.net.*;
import java.io.*;
public class chatserver
{
        public static void main(String args[]) throws Exception
        {
                ServerSocket ss=new ServerSocket(2000);
                Socket sk=ss.accept();
                BufferedReader cin=new BufferedReader(new InputStreamReader(sk.getInputStream()));
                PrintStream cout=new PrintStream(sk.getOutputStream());
                BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
                String s;
                while ( true )
                {
                        s=cin.readLine();
                        if (s.equalsIgnoreCase("END"))
                        {

                                cout.println("BYE");
                                break;

                         }
                        System. out.print("Client : "+s+"\n");

                        System.out.print("Server : ");

                         s=stdin.readLine();

                        cout.println(s);

                }

                ss.close();
                sk.close();

                cin.close();

                cout.close();


                stdin.close();

        }

}
```

Chatclient.java

```
import java.net.*;

import java.io.*;

public class chatclient

{
        public static void main(String args[]) throws Exception
        {

                Socket sk=new Socket("127.0.0.1",2000);

                BufferedReader sin=new BufferedReader(new InputStreamReader(sk.getInputStream()));
                PrintStream sout=new PrintStream(sk.getOutputStream());
                BufferedReader stdin=new BufferedReader(new InputStreamReader(System.in));
                String s;

                while ( true )

                {
                        System.out.print("Client : ");
                        s=stdin.readLine();
                        sout.println(s);
                        s=sin.readLine();
                        System.out.print("Server : "+s+"\n");
                        if ( s.equalsIgnoreCase("BYE") )
                          break;
                 }
                 sk.close();
                 sin.close();
                 sout.close();
                 stdin.close();
             }
         }
```

OUTPUT:

Server:

```
E:\nwlab>javac chatserver.java
E:\nwlab>java chatserver
Client : hi
Server : hi
```

Client:

```
E:\nwlab>javac chatclient.java
E:\nwlab>java chatclient
Client : hi
Server : hi Client
```

RESULT:

CHAT application using TCP Socket is implemented successfully.

Ex No: 3c **(C) File Transfer**
Date:

AIM:

To write a java program for file transfer using TCP Sockets.

ALGORITHM

SERVER

Step1: Import java packages and create class file server.

Step2: Create a new server socket and bind it to the port.

Step3: Accept the client connection

Step4: Get the file name and stored into the BufferedReader.

Step5: Create a new object class file and realine.

Step6: If file is exists then FileReader read the content until EOF is reached.

Step7: Stop the program.

CLIENT

Step1: Import java packages and create class file server.

Step2: Create a new server socket and bind it to the port.

Step3: Now connection is established.

Step4: The object of a BufferReader class is used for storing data content which has been retrieved from socket object.

Step5 The connection is closed.

Step6: Stop the program

Program

File Server

```
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.OutputStream;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class FileServer
{
   public static void main(String[] args) throws Exception
        {
             //Initialize Sockets
             ServerSocket ssock = new ServerSocket(5000);
             Socket socket = ssock.accept();
           //The InetAddress specification
                InetAddress IA = InetAddress.getByName("localhost");
        //Specify the file
             File file = new File("e:\\Bookmarks.html");
             FileInputStream fis = new FileInputStream(file);
                BufferedInputStream bis = new BufferedInputStream(fis);
                //Get socket's output stream
                OutputStream os = socket.getOutputStream();
             //Read File Contents into contents array
             byte[] contents;
               long fileLength = file.length();
               long current = 0;
                long start = System.nanoTime();
                 while(current!=fileLength){
                         int size = 10000;
                         if(fileLength - current >= size)
                         current += size;
                         else{
                                     size = (int)(fileLength - current);
                                     current = fileLength;
                           }
                                 contents = new byte[size];
                         bis.read(contents, 0, size);
                os.write(contents);
                System.out.print("Sending file ... "+(current*100)/fileLength+"% complete!");
                }
           os.flush();
        //File transfer done. Close the socket connection! socket.close();
        ssock.close();
        System.out.println("File sent succesfully!");
        }
        }
```

File Client

```
import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;
public class FileClient {
    public static void main(String[] args) throws Exception{
        //Initialize socket
            Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
            byte[] contents = new byte[10000];
        //Initialize the FileOutputStream to the output file's full path.
                FileOutputStream fos = new FileOutputStream("e:\\Bookmarks1.html");
                BufferedOutputStream bos = new BufferedOutputStream(fos);
                InputStream is = socket.getInputStream();
         //No of bytes read in one read() call
            int bytesRead = 0;
            while((bytesRead=is.read(contents))!=-1)
                bos.write(contents, 0, bytesRead);
            bos.flush();
            socket.close();
            System.out.println("File saved successfully!");
    }
}
```

OUTPUT

SERVER

```
E:\nwlab>java FileServer
 Sending file ... 9% complete!
 Sending file ... 19% complete!
 Sending file ... 28% complete!
 Sending file ... 38% complete!
Sending file ... 47% complete!
Sending file ... 57% complete!
Sending file ... 66% complete!
Sending file ... 76% complete!
Sending file ... 86% complete!
Sending file ... 95% complete!
Sending file ... 100% complete! File sent
successfully!

E:\nwlab>client E:\nwlab>java
FileClient File saved successfully!

E:\nwlab>
```

RESULT

Thus the java program file transfer application using TCP Sockets was executed

Ex No: 4 Date:	Simulation of DNS using UDP sockets

AIM

To write a java program for DNS application program Algorithm

ALGORITHM

1. Start the program.
2. Get the frame size from the user
3. To create the frame based on the user request.
4. To send frames to server from the client side.
5. If your frames reach the server it will send ACK signal to client otherwise it will send NACK signal to client.
6. Stop the program

PROGRAM

Udpdnsserver.java

```
  import java.io.*;
  import java.net.*;
 public class udpdnsserver
{
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(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();
}
}
}
```

Udpdnsclient.java

```
  import java.io.*;
  import java.net.*;
 public class udpdnsclient
 {
 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

Server

```
E:\nwlab> javac udpdnsserver.java
E:\nwlab>  java udpdnsserver
Press Ctrl + C to Quit
Request for host yahoo.com
Request for host cricinfo.com
Request for host youtube.com
```

Client

```
E:\nwlab> udpdnsclient.java
E:\nwlab>   java udpdnsclient
Enter the hostname: yahoo.com
IP Address: 68.180.206.184
E:\nwlab>   java udpdnsclient
Enter the hostname: cricinfo.com
IP Address: 80.168.92.140
E:\nwlab>   java udpdnsclient
Enter the hostname: youtube.com
IP Address: Host Not Found
```

RESULT

Thus the   DNS application program was executed successfully.

Ex No: 5(a) Date:	**SIMULATING ARP / RARP PROTOCOLS** **(a) SIMULATING ARP PROTOCOLS**

AIM:

To write a java program for simulating ARP protocols using TCP

ALGORITHM:

Client

1. Start the program
2. Using socket connection is established between client and server.
3. Get the IP address to be converted into MAC address.
4. Send this IP address to server.
5. Server returns the MAC address to client.

Server

1.Start the program

2.Accept the socket which is created by the client.

3.Server maintains the table in which IP and corresponding MAC addresses are stored.

4.Read the IP address which is send by the client.

5.Map the IP address with its MAC address and return the MAC address to client.

Program for Address Resolution Protocol (ARP) using TCP

Client:

```
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);
        }
        }
}
```

Server:

```
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);
        }
        }
}
```

OUTPUT:

```
E:\networks>java Serverarp E:\
networks>java Clientarp
Enter the Logical address (IP):
165.165.80.80
The Physical Address is: 6A:08:AA:C2
```

RESULT:

Thus the ARP protocol using TCP Sockets program was executed.

Ex No:5(b) **Date:**	**(b) REVERSE ADDRESS RESOLUTION PROTOCOL (RARP) USING UDP**

AIM:

To write a java program for simulating RARP protocols using UDP

ALGORITHM

Client

1. Start the program
2. Using datagram sockets UDP function is established.
3. Get the MAC address to be converted into IP address.
4. Send this MAC address to server.
5. Server returns the IP address to client.

Server

1.Start the program.

2.Server maintains the table in which IP and corresponding MAC addresses are stored.

3.Read the MAC address which is send by the client.

4.Map the IP address with its MAC address and return the IP address to client.

PROGRAM

CLIENT

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientrarp12
{
        public static void main(String args[])
        {
        try
        {
                DatagramSocket client=new DatagramSocket();
                InetAddress addr=InetAddress.getByName("127.0.0.1");
                byte[] sendbyte=new byte[1024];
                byte[] receivebyte=new byte[1024];
                BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
                System.out.println("Enter the Physical address (MAC):");
                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(IP): "+s.trim());
                client.close();
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
        }
}
```

SERVER

```
import java.io.*;
import java.net.*;
import java.util.*;
class Serverrarp12
{
        public static void main(String args[])
        {
        try
        {
                DatagramSocket server=new DatagramSocket(1309);
                while(true)
                {
                        byte[] sendbyte=new byte[1024];
                         byte[] receivebyte=new byte[1024];
                        DatagramPacket receiver=new
                        DatagramPacket(receivebyte,receivebyte.length);
                        server.receive(receiver);
                        String str=new String(receiver.getData());
                        String s=str.trim();
                        //System.out.println(s);
                        InetAddress addr=receiver.getAddress();
                        int port=receiver.getPort();
                        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(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);
        }
        }
}
```

OUTPUT

```
I:\ex>java Serverrarp12 I:\ex>java
Clientrarp12
Enter the Physical address (MAC):
6A:08:AA:C2
The Logical Address is(IP): 165.165.80.80
```

RESULT:

Thus the RARP program using UDP was executed.

Ex No:6(a)

Date:

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

(a) STUDY OF NETWORK SIMULATOR (NS)

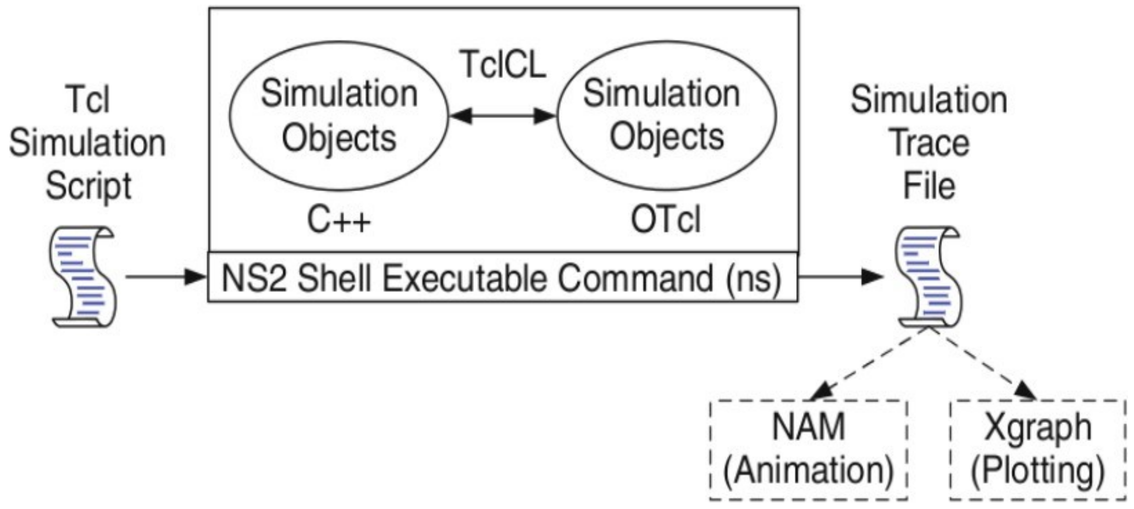
AIM

To study about NS2 simulator in detail.

THEORY:

Network Simulator (Version 2), widely known as NS2, is simply an event driven simulation tool that has proved useful in studying the dynamic nature of communication networks.
Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989.

Basic Architecture of NS2:



The above figure shows the basic architecture of NS2. NS2 provides users with an executable command ns which takes on input argument, the name of a Tcl simulation scripting file. Users are feeding the name of a Tcl simulation script (which sets up a simulation) as an input argument of an NS2 executable command ns.

In most cases, a simulation trace file is created, and is used to plot graph and/or to create animation. NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend).

The C++ and the OTcl are linked together using TclCL. Mapped to a C++ object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle (e.g., n as a Node handle) is just a string (e.g.,_o10) in the OTcl domain, and does not contain any functionality. instead, the functionality (e.g., receiving a packet) is defined in the mapped C++ object (e.g., of class Connector). In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may defines its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively

Tcl scripting:

- Tcl is a general purpose scripting language. [Interpreter]
- Tcl runs on most of the platforms such as Unix, Windows, and Mac.
- The strength of Tcl is its simplicity.
- It is not necessary to declare a data type for variable prior to the usage

Basics of TCL

Syntax: command arg1 arg2 arg3

Hello World!

```
        puts stdout{Hello, World!} Hello,
World!
```

Variables

Command Substitution

```
set a 5 set len [string length foobar]
set b $a set len [expr [string length foobar] + 9]
```

Simple Arithmetic

```
expr 7.2 / 4
```

Procedures

```
proc Diag {a b} {
set c [expr sqrt($a * $a + $b * $b)] return $c }
puts ―Diagonal of a 3, 4 right triangle is [Diag 3 4]‖ Output:
Diagonal of a 3, 4 right triangle is 5.0
```

Loops

while{$i < $n} {
. . .
}
	for {set i 0} {$i < $n} {incr i}
	{
	. . .
	}

NS Simulator Preliminaries.

1. Initialization and termination aspects of the ns simulator.
2. Definition of network nodes, links, queues and topology.
3. Definition of agents and of applications.
4. The nam visualization tool.
5. Tracing and random variables.

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

```
set ns [new Simulator]
```

Which is thus the first line in the tcl script? This line declares a new variable as using the set command, you can call this variable as you wish, In general people declares it as ns because it is an instance of the Simulator class, so an object the code[new Simulator] is indeed the installation of the class Simulator using the reserved word new.

In order to have output files with data on the simulation (trace files) or files used for visualization (nam files), we need to create the files using —"open" command:

#Open the Trace file

```
set tracefile1 [open out.tr w]

$ns trace-all $tracefile1

 #Open the NAM trace file

set namfile [open out.nam w]

$ns namtrace-all $namfile
```

The above creates a dta trace file called —out.tr‖ and a nam visualization trace file called out.nam‖.Within the tcl script,these files are not called explicitly by their names, but instead by pointers that are declared above and called —tracefile1 and —nam file respectively. Remark that they begins with a # symbol.The second line open the file —out.tr‖ to be used for writing, declared with the letter —w‖.The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go.

The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command $ns flush-trace. In our case, this will be the file pointed at by the pointer —$namfile ,i.e the file —out.tr‖.

The termination of the program is done using a —finish‖ procedure.

#Define a 'finish' procedure

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace Close
$tracefile1 Close $namfile

Exec nam out.nam & Exit 0
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure. The simulator method **—flush-trace"** will dump the traces on the respective files. The tcl command

—close" closes the trace files defined before and **exec** executes the nam program for visualization.
The command **exit** will ends the application and return the number 0 as status to the system.
Zero is the default for a clean exit. Other values can be used to say that is a exit because something fails.

At the end of ns program we should call the procedure —finish‖ and specify at what time the termination should occur. For example,

$ns at 125.0 "finish"

will be used to call **—finish‖** at time 125sec.Indeed,the **at** method of the simulator allows us to schedule events explicitly.
The simulation can then begin using the command

$ns run

Definition of a network of links and nodes

The way to define a node is

set n0 [$ns node]

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write $n0.
Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

$ns duplex-link $n0 $n2 10Mb 10ms DropTail

Which means that $n0 and $n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction. To define a directional link instead of a bi-directional one, we should replace "duplexlink" by "simplex- link".

In NS, an output queue of a node is implemented as a part of each link whose input is that node. The definition of the link then includes the way to handle overflow at that queue. In our case, if the buffer capacity of the output queue is exceeded then the last packet to arrive is dropped. Many alternative options exist, such as the RED (Random Early Discard) mechanism, the FQ (Fair Queuing), the DRR (Deficit Round Robin), the stochastic Fair Queuing (SFQ) and the CBQ (which including a priority and a round-robin scheduler).

In ns, an output queue of a node is implemented as a part of each link whose input is that node. We should also define the buffer capacity of the queue related to each link. An example would be:

```
#set Queue Size of link (n0-n2) to 20
$ns queue-limit $n0 $n2 20
```

Agents and Applications

We need to define routing (sources, destinations) the agents (protocols) the application that use them

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received. There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

```
set tcp [new Agent/TCP]
```

The command **$ns attach-agent $n0 $tcp** defines the source node of the tcp connection.

The command

```
set sink [new Agent /TCPSink]
```

Defines the behavior of the destination node of TCP and assigns to it a pointer called sink

#Setup a UDP connection

```
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp set null [new
Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
#setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
```

```
$cbr attach-agent $udp

$cbr set packetsize_ 100

$cbr set rate_ 0.01Mb

$cbr set random_ false
```

Above shows the definition of a CBR application using a UDP agent.

The command**$ns attach-agent $n4 $sink** defines the destination node.
The command **$ns connect $tcp $sink** finally makes the TCP connection between the source and destination nodes.

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes.This can be changed to another value, say 552bytes, using the command $tcp set packetSize_ 552. When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command $tcp set fid_ 1 that assigns to the TCP connection a flow identification of "1".We shall later give the flow identification of "2" to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP. Instead of defining the rate in the command $cbr set rate_ 0.01Mb, one can define the time interval between transmission of packets using the command.

```
$cbr set interval_ 0.005
```

The packet size can be set to some value using

```
$cbr set packetSize_ <packet size>
```

Scheduling Events

NS is a discrete event based simulation. The tcp script defines when event should occur. The initializing command set ns [new Simulator] creates an event scheduler, and events are then scheduled using the format:

$ns at <time> <event>

The scheduler is started when running ns that is through the command $ns run. The beginning and end of the FTP and CBR application can be done through the following command

```
$ns at 0.1 “$cbr start”
$ns at 1.0 “ $ftp start”
$ns at 124.0 “$ftp stop”
$ns at 124.5 “$cbr stop”
```

RESULT:

Thus the Network Simulator 2 is studied in detail.

EX.No:6(b)
Date:

CONGESTION CONTROL ALGORITHMS USING NS

AIM:

To simulate a link failure and observe the congestion control algorithm using NS2.

ALGORITHM:

1. Create a simulation object
2. Set routing protocol to routing
3. Trace packets and all links onto NAM trace and to trace file
4. Create right nodes
5. Describe their layout topology as octagon
6. Add a sink agent to node
7. Connect source and sink.

PROGRAM:

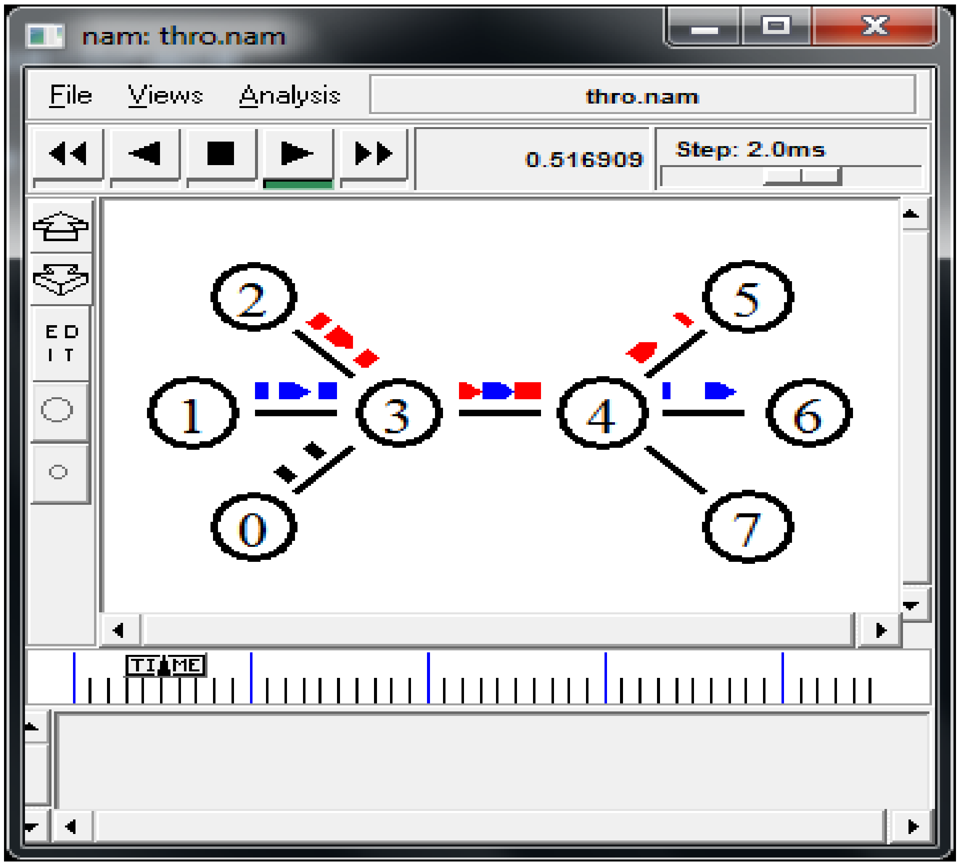
```
set ns [new Simulator]

set nr [open thro_red.tr w]

$ns trace-all $nr

set nf [open thro.nam w]

$ns      namtrace-all

     $nf proc  finish

     { }

  {

     global ns nr nf

     $ns flush-

     trace close

     $nf close $nr
```

```
    exec nam thro.nam &
    exit 0 }
set   n0   [$ns
node]  set  n1
[$ns  node]  set
n2  [$ns  node]
set   n3   [$ns
node]  set  n4
[$ns  node]  set
n5  [$ns  node]
set   n6   [$ns
node]  set  n7
[$ns node]
$ns duplex-link $n0 $n3 1Mb 10ms RED
$ns duplex-link $n1 $n3 1Mb 10ms RED
$ns duplex-link $n2 $n3 1Mb 10ms RED
$ns duplex-link $n3 $n4 1Mb 10ms RED
$ns duplex-link $n4 $n5 1Mb 10ms RED
$ns duplex-link $n4 $n6 1Mb 10ms RED
$ns duplex-link $n4 $n7 1Mb 10ms RED
$ns duplex-link-op $n0 $n3 orient right-up
$ns duplex-link-op $n3 $n4 orient middle
$ns duplex-link-op $n2 $n3 orient right-down
$ns duplex-link-op $n4 $n5 orient right-up
$ns duplex-link-op $n4 $n7 orient right-down
$ns duplex-link-op $n1 $n3 orient right
```

```
$ns duplex-link-op $n6 $n4 orient

 left set udp0 [new Agent/UDP]

$ns attach-agent $n2 $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500

$cbr0 set interval_ 0.005

$cbr0 attach-agent $udp0

set null0 [new Agent/Null]

$ns attach-agent $n5 $null0

$ns connect $udp0 $null0

set udp1 [new Agent/UDP]

$ns attach-agent $n1 $udp1

set cbr1 [new Application/Traffic/CBR]

$cbr1 set packetSize_ 500

$cbr1 set interval_ 0.005

$cbr1 attach-agent $udp1

set null0 [new Agent/Null]

$ns attach-agent $n6 $null0

$ns connect $udp1 $null0

set udp2 [new Agent/UDP]

$ns attach-agent $n0 $udp2

set cbr2 [new Application/Traffic/CBR]

$cbr2 set packet size_ 500

$cbr2 set interval_ 0.005

$cbr2 attach-agent $udp2

set null0 [new Agent/Null]

$ns attach-agent $n7 $null0
```

```
$ns connect $udp2 $null0

$udp0 set fid_ 1
$udp1 set fid_ 2

$udp2 set fid_ 3

$ns color 1 Red

$ns color 2 Green

$ns color 2 Blue

 $ns at 0.1 "$cbr0 start"

$ns at 0.2 "$cbr1 start"

$ns at 0.5 "$cbr2 start"

$ns at 4.0 "$cbr2 stop"

$ns at 4.2 "$cbr1 stop"

$ns at 4.5 "$cbr0 stop"

$ns at 5.0 "finish"

$ns run
```

OUTPUT:



RESULT:

Thus the congestion control algorithm is simulated successfully by using NS2.

Ex.No:7(a)

Date:

STUDY OF TCP/UDP PERFORMANCE USING SIMULATION TOOL

(a) STUDY OF TCP PERFORMANCE

Aim:

To study about TCP Performance

The **Transmission Control Protocol** (**TCP**) is one of the core protocols of the Internet Protocol Suite. TCP is one of the two original components of the suite, complementing the Internet Protocol (IP), and therefore the entire suite is commonly referred to as *TCP/IP*. TCP provides the service of exchanging data directly between two network hosts, whereas IP handles addressing and routing message across one or more networks. In particular, TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol that major Internet applications rely on, applications such as the World Wide Web, e-mail, and file transfer. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP) which provides a datagram service that emphasizes reduced latency over reliability.

Network function

TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details.

IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a *header* followed by a *body*. The header describes the packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost, duplicated, or delivered out of order. TCP detects these problems, requests retransmission of lost packets, rearranges out-of-order packets, and even helps minimize network congestion to reduce the occurrence of the other problems. Once the TCP receiver has finally reassembled a perfect copy of the data originally transmitted, it passes that datagram to the application program. Thus, TCP abstracts the application's communication from the underlying networking details.

TCP is utilized extensively by many of the Internet's most popular applications, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and some streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore, TCP sometimes incurs relatively long delays (in the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service that guarantees delivery of a data stream sent from one host to another without duplication or losing data. Since packet transfer is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends, and waits for acknowledgment before sending the next packet. The sender also keeps a timer from when the packet was sent, and retransmits a packet if the timer expires. The timer is needed in case a packet gets lost or corrupted.

TCP consists of a set of rules: for the protocol, that are used with the Internet Protocol, and for the IP, to send data "in a form of message units" between computers over the Internet. At the same time that IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data transmission, called *segments* that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a Web server, the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. Even though every packet has the same destination address, they can be routed on different paths through the network. When the client program on the destination computer receives them, the TCP layer (Transport Layer) reassembles the individual segments and ensures they are correctly ordered and error free as it streams them to an application.

Protocol operation

A Simplified TCP State Diagram. See TCP EFSM diagram for a more detailed state diagram including the states inside the ESTABLISHED state.

TCP protocol operations may be divided into three phases. Connections must be properly established in a multi-step handshake process (*connection establishment*) before entering the *data transfer* phase. After data transmission is completed, the *connection termination* closes established virtual circuits and releases all allocated resources.

A TCP connection is managed by an operating system through a programming interface that represents the local end-point for communications, the *Internet socket*. During the lifetime of a TCP connection it undergoes a series of state changes:

1. LISTEN: In case of a server, waiting for a connection request from any remote client.
2. SYN-SENT : waiting for the remote peer to send back a TCP segment with the SYN and ACK flags set. (usually set by TCP clients)
3. SYN-RECEIVED: waiting for the remote peer to send back an acknowledgment after having sent back a connection acknowledgment to the remote peer. (usually set by TCP servers)
4. ESTABLISHED: the port is ready to receive/send data from/to the remote peer.
5. FIN-WAIT-1
6. FIN-WAIT-2
7. CLOSE-WAIT
8. CLOSING
9. LAST-ACK
10. TIME-WAIT: represents waiting for enough time to pass to be sure the remote peer received the acknowledgment of its connection termination request. According to RFC 793 a connection can stay in TIME-WAIT for a maximum of four minutes.

11. CLOSED **Connection establishment**

To establish a connection, TCP uses a three-way handshake. Before a client attempts to connect with a server, the server must first bind to a port to open it up for connections: this is called a passive open. Once the passive open is established, a client may initiate an active open. To establish a connection, the three-way (or 3-step) handshake occurs:

1. **SYN**: The active open is performed by the client sending a SYN to the server. It sets the segment's sequence number to a random value A.
2. **SYN-ACK**: In response, the server replies with a SYN-ACK. The acknowledgment number is set to one more than the received sequence number (A + 1), and the sequence number that the server chooses for the packet is another random number, B.

1. **ACK**: Finally, the client sends an ACK back to the server. The sequence number is set to the received acknowledgement value i.e. A + 1, and the acknowledgement number is set to one more than the received sequence number i.e. B + 1.

At this point, both the client and server have received an acknowledgment of the connection.

Data transfer

There are a few key features that set TCP apart from User Datagram Protocol:

- Ordered data transfer - the destination host rearranges according to sequence number
- Retransmission of lost packets - any cumulative stream not acknowledged is retransmitted
- Error-free data transfer (The checksum in UDP is optional)
- Flow control - limits the rate a sender transfers data to guarantee reliable delivery. The receiver continually hints the sender on how much data can be received (controlled by the sliding window). When the receiving host's buffer fills, the next acknowledgment contains a 0 in the window size, to stop transfer and allow the data in the buffer to be processed.
- Congestion control.

Reliable transmission

TCP uses a *sequence number* to identify each byte of data. The sequence number identifies the order of the bytes sent from each computer so that the data can be reconstructed in order, regardless of any fragmentation, disordering, or packet loss that may occur during transmission. For every payload byte transmitted the sequence number must be incremented. In the first two steps of the 3-way handshake, both computers exchange an initial sequence number (ISN). This number can be arbitrary, and should in fact be unpredictable to defend against TCP Sequence Prediction Attacks.

TCP primarily uses a *cumulative acknowledgment* scheme, where the receiver sends an acknowledgment signifying that the receiver has received all data preceding the acknowledged sequence number. Essentially, the first byte in a segment's data field is assigned a sequence number, which is inserted in the sequence number field, and the receiver sends an acknowledgment specifying the sequence number of the next byte they expect to receive. In addition to cumulative acknowledgments, TCP receivers can also send selective acknowledgments to provide further information (*see selective acknowledgments*).

Maximum segment size

The Maximum segment size (MSS) is the largest amount of data, specified in bytes, that TCP is willing to send in a single segment. For best performance, the MSS should be set small enough to avoid IP fragmentation, which can lead to excessive retransmissions if there is packet loss. To try to accomplish this, typically the MSS is negotiated using the MSS option when the TCP connection is established, in which case it is determined by the maximum transmission unit (MTU) size of the data link layer of the networks to which the sender and receiver are directly attached. Furthermore, TCP senders can use Path MTU discovery to infer the minimum MTU along the network path between the sender and receiver, and use this to dynamically adjust the MSS to avoid IP fragmentation within the network.

TCP Timestamps

TCP timestamps, defined in RFC 1323, help TCP compute the round-trip time between the sender and receiver. Timestamp options include a 4-byte timestamp value, where the sender inserts its current value of its timestamp clock, and a 4-byte echo reply timestamp value, where the receiver generally inserts the most recent timestamp value that it has received. The sender uses the echo reply timestamp in an acknowledgment to compute the total elapsed time since the acknowledged segment was sent.

TCP timestamps are also used to help in the case where TCP sequence numbers encounter their 232 bound and "wrap around" the sequence number space. This scheme is known as *Protect Against Wrapped Sequence* numbers, or *PAWS* (see RFC 1323 for details). Furthermore, the Eifel detection algorithm, defined in RFC 3522, which detects unnecessary loss recovery

Connection termination

The connection termination phase uses, at most, a four-way handshake, with each side of the connection terminating independently. When an endpoint wishes to stop its half of the connection, it transmits a FIN packet, which the other end acknowledges with an ACK. Therefore, a typical tear-down requires a pair of FIN and ACK segments from each TCP endpoint.

A connection can be "half-open", in which case one side has terminated its end, but the other has not. The side that has terminated can no longer send any data into the connection, but the other side can. The terminating side should continue reading the data until the other side terminates as well.

It is also possible to terminate the connection by a 3-way handshake, when host A sends a FIN and host B replies with a FIN & ACK (merely combines 2 steps into one) and host A replies with an ACK. This is perhaps the most common method.

It is possible for both hosts to send FINs simultaneously then both just have to ACK. This could possibly be considered a 2-way handshake since the FIN/ACK sequence is done in parallel for both directions.

Some host TCP stacks may implement a "half-duplex" close sequence, as Linux or HP-UX do. If such a host actively closes a connection but still has not read all the incoming data the stack already received from the link, this host sends a RST instead of a FIN (Section 4.2.2.13 in RFC 1122). This allows a TCP application to be sure the remote application has read all the data the former sent—waiting the FIN from the remote side, when it actively closes the connection. However, the remote TCP stack cannot distinguish between a *Connection Aborting RST* and this *Data Loss RST*. Both cause the remote stack to throw away all the data it received, but that the application still didn't read.[*clarification needed*]

Some application protocols may violate the OSI model layers, using the TCP open/close handshaking for the application protocol open/close handshaking - these may find the RST problem on active close. As an example:

s = connect(remote);

```
send(s, data);
close(s);
```

For a usual program flow like above, a TCP/IP stack like that described above does not guarantee that all the data arrives to the other application *unless* the programmer is sure that the remote side will not send anything.

TCP ports

Main article: TCP and UDP port

TCP uses the notion of port numbers to identify sending and receiving application end-points on a host, or *Internet sockets*. Each side of a TCP connection has an associated 16-bit unsigned port number (0-65535) reserved by the sending or receiving application. Arriving TCP data packets are identified as belonging to a specific TCP connection by its sockets, that is, the combination of source host address, source port, destination host address, and destination port. This means that a server computer can provide several clients with several services simultaneously, as long as a client takes care of initiating any simultaneous connections to one destination port from different source ports.

Port numbers are categorized into three basic categories: well-known, registered, and dynamic/private. The well-known ports are assigned by the Internet Assigned Numbers Authority (IANA) and are typically used by system-level or root processes. Well-known applications running as servers and passively listening for connections typically use these ports. Some examples include: FTP (21), SSH (22), TELNET (23), SMTP (25) and HTTP (80). Registered ports are typically used by end user applications as ephemeral source ports when contacting servers, but they can also identify named services that have been registered by a third

RESULT:

Thus the performance of TCP is studied successfully.

Ex.No: 7(b) **Date:**	**(b) STUDY OF UDP PERFORMANCE**

Aim:

To study about UDP Performance

The **User Datagram Protocol** (**UDP**) is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as *datagrams*, to other hosts on an Internet Protocol (IP) network without requiring prior communications to set up special transmission channels or data paths. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model without implicit hand-shaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers answering small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games.

Service ports

Main article: TCP and UDP port

UDP applications use datagram sockets to establish host-to-host communications. An application binds a socket to its endpoint of data transmission, which is a combination of an IP address and a service port. A port is a software structure that is identified by the port number, a 16 bit integer value, allowing for port numbers between 0 and 65535. Port 0 is reserved, but is a permissible source port value if the sending process does not expect messages in response.

The Internet Assigned Numbers Authority has divided port numbers into three ranges. Port numbers 0 through 1023 are used for common, well-known services. On Unix-like operating systems, using one of these ports requires superuser operating permission. Port numbers 1024 through 49151 are the registered ports used for IANA-registered services. Ports 49152 through 65535 are dynamic ports that are not officially for any specific service, and can be used for any purpose. They are also used as ephemeral ports, from which software running on the host may randomly choose a port in order to define itself. In effect, they are used as temporary ports primarily by clients when communicating with servers.

Packet structure

UDP is a minimal message-oriented Transport Layer protocol that is documented in IETF RFC 768.

UDP provides no guarantees to the upper layer protocol for message delivery and the UDP protocol layer retains no state of UDP messages once sent. For this reason, UDP is sometimes referred to as *Unreliable* Datagram Protocol.[*citation needed*]

UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload.[3] If transmission reliability is desired, it must be implemented in the user's application.

bits	**0 – 15**	**16 – 31**
0	Source Port Number	Destination Port Number
32	Length	Checksum
64		
	Data	

The UDP header consists of 4 fields, all of which are 2 bytes (16 bits). The use of two of those is optional in IPv4 (pink background in table). In IPv6 only the source port is optional (see below).

Source port number

This field identifies the sender's port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero. If the source host is the client, the port number is likely to be an ephemeral port number. If the source host is the server, the port number is likely to be a well-known port number.

Destination port number

This field identifies the receiver's port and is required. Similar to source port number, if the client is the destination host then the port number will likely be an ephemeral port number and if the destination host is the server then the port number will likely be a well-known port number.

Length

A field that specifies the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes since that's the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes (65,535 − 8 byte UDP header − 20 byte IP header).

Checksum

The checksum field is used for error-checking of the header *and* data. If no checksum is generated by the transmitter, the field uses the value all-zeros. This field is not optional for IPv6.

Checksum computation

The method used to compute the checksum is defined in RFC 768:

Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

In other words, all 16-bit words are summed using one's complement arithmetic. The sum is then one's complemented to yield the value of the UDP checksum field.

If the checksum calculation results in the value zero (all 16 bits 0) it should be sent as the one's complement (all 1's).

The difference between IPv4 and IPv6 is in the data used to compute the checksum.

IPv4 PSEUDO-HEADER

When UDP runs over IPv4, the checksum is computed using a PSEUDO-HEADER that contains some of the same information from the real IPv4 header. The PSEUDO-HEADER is not the real IPv4 header used to send an IP packet. The following table defines the PSEUDO-HEADER used only for the checksum calculation.

bits	**0 – 7**	**8 – 15**	**16 – 23**	**24 – 31**
0	Source address			
32	Destination address			
64	Zeros	Protocol	UDP length	
96	Source Port		Destination Port	
128	Length		Checksum	
160	Data			

When computing the checksum, again a PSEUDO-HEADER is used that mimics the real IPv6 header:

The source address is the one in the IPv6 header. The destination address is the final destination; if the IPv6 packet doesn't contain a Routing header, that will be the destination address in the IPv6 header; otherwise, at the originating node, it will be the address in the last element of the Routing header, and, at the receiving node, it will be the destination address in the IPv6 header. The value of the Next Header field is the protocol value for UDP: 17. The UDP length field is the length of the UDP header and data.

Reliability and congestion control solutions

Lacking reliability, UDP applications must generally be willing to accept some loss, errors or duplication. Some applications such as TFTP may add rudimentary reliability mechanisms into the application layer as needed. Most often, UDP applications do not require reliability mechanisms and may even be hindered by them. Streaming media, real-time multiplayer games and voice over IP (VoIP) are examples of applications that often use UDP. If an application requires a high degree of reliability, a protocol such as the Transmission Control Protocol or erasure codes may be used instead.

Lacking any congestion avoidance and control mechanisms, network-based mechanisms are required to minimize potential congestion collapse effects of uncontrolled, high rate UDP traffic loads. In other words, since UDP senders cannot detect congestion, network-based elements such as routers using packet queuing and dropping techniques will often be the only tool available to slow down excessive UDP traffic. The Datagram Congestion Control Protocol (DCCP) is being designed as a partial solution to this potential problem by adding end host TCP-friendly congestion control behavior to high-rate UDP streams such as streaming media.

Comparison of UDP and TCP

Main article: Transport Layer

Transmission Control Protocol is a connection-oriented protocol, which means that it requires handshaking to set up end-to-end communications. Once a connection is set up user data may be sent bi-directionally over the connection.

- *Reliable* – TCP manages message acknowledgment, retransmission and timeout. Multiple attempts to deliver the message are made. If it gets lost along the way, the server will re-request the lost part. In TCP, there's either no missing data, or, in case of multiple timeouts, the connection is dropped.
- *Ordered* – if two messages are sent over a connection in sequence, the first message will reach the receiving application first. When data segments arrive in the wrong order, TCP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.
- *Heavyweight* – TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control.
- *Streaming* – Data is read as a byte stream, no distinguishing indications are transmitted to signal message (segment) boundaries.

UDP is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

- *Unreliable* – When a message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission or timeout.
- *Not ordered* – If two messages are sent to the same recipient, the order in which they arrive cannot be predicted.
- *Lightweight* – There is no ordering of messages, no tracking connections, etc. It is a small transport layer designed on top of IP.
- *Datagrams* – Packets are sent individually and are checked for integrity only if they arrive. Packets have definite boundaries which are honored upon receipt, meaning a read operation at the receiver socket will yield an entire message as it was originally sent.

RESULT:

Thus the performance of UDP is studied successfully.

Ex.No:8(a)
Date:

SIMULATION OF DISTANCE VECTOR/ LINK STATE ROUTING ALGORITHM

(a) SIMULATION OF DISTANCE VECTOR ROUTING ALGORITHM

AIM:

Develop a program for distance vector routing algorithm to find suitable path for transmission.

ALGORITHM:

1. Start
2. By convention, the distance of the node to itself is assigned to zero and when a node is unreachable the distance is accepted as 999.
3. Accept the input distance matrix from the user (*costmat [][]*) that represents the distance between each node in the network.
4. Store the distance between nodes in a suitable varible.
5. Calculate the minimum distance between two nodes by iterating.
    - If the distance between two nodes is larger than the calculated alternate available path, replace the existing distance with the calculated distance.
6. Print the shortest path calculated.
7. Stop.

PROGRAM:

```
#include<stdio.h>
struct node
{
   unsigned dist[20];
   unsigned from[20];
}rt[10];
int main()
{
   int costmat[20][20];
   int nodes,i,j,k,count=0;
   printf("\nEnter the number of nodes : ");
   scanf("%d",&nodes); //Enter the nodes
   printf("\nEnter the cost matrix :\n");
   for(i=0;i<nodes;i++)
   {
      for(j=0;j<nodes;j++)
      {
         scanf("%d",&costmat[i][j]);
         costmat[i][i]=0;
         rt[i].dist[j]=costmat[i][j]; //initialise the distance equal to cost matrix
         rt[i].from[j]=j;
      }
   }
      do
      {
         count=0;

         for(i=0;i<nodes;i++) //We choose arbitary vertex k and we calculate the direct distance from the
node i to k using the cost matrix

         //and add the distance from k to node j

         for(j=0;j<nodes;j++)

         for(k=0;k<nodes;k++)

            if(rt[i].dist[j]>costmat[i][k]+rt[k].dist[j])

            {

              //We calculate the minimum distance

               rt[i].dist[j]=rt[i].dist[k]+rt[k].dist[j];

               rt[i].from[j]=k;

               count++;

            }

      }while(count!=0);

      for(i=0;i<nodes;i++)

      {

         printf("\n\n For router %d\n",i+1);

         for(j=0;j<nodes;j++)

         {

            printf("\t\nnode %d via %d Distance %d ",j+1,rt[i].from[j]+1,rt[i].dist[j]);

         }

      }

   printf("\n\n");

   getch();

}
```

OUTPUT:

```
Enter the number of nodes: 3

Enter the cost matrix:

0 2 7

2 0 1

7 1 0

For router 1

node 1 via 1 Distance 0

node 2 via 2 Distance 2

node 3 via 3 Distance 3

For router 2

node 1 via 1 Distance 2

node 2 via 2 Distance 0

node 3 via 3 Distance 1

For router 3

node 1 via 1 Distance 3

node 2 via 2 Distance 1

node 3 via 3 Distance 0
```

RESULT:

Thus the Distance Vector Routing algorithm was simulated by using C programming.

Ex. No: 8(b) Date:	(b) SIMULATION OF LINK STATE ROUTING ALGORITHM

AIM:

Develop a program for link state routing algorithm using Dijkstra algorithm to find shortest path for transmission

Link State Routing protocol

In link state routing, each router shares its knowledge of its neighborhood with every other router in theinternet work.

(i) Knowledge about Neighborhood: Instead of sending its entire routing table a router sends info about its neighborhood only.

(ii) To all Routers: each router sends this information to every other router on the internet work not just to its neighbor .It does so by a process called flooding.

(iii)Information sharing when there is a change: Each router sends out information about the neighbors when there is change.

ALGORITHM:

Step-1: The node is taken and chosen as a root node of the tree, this creates the tree with a single node, and now set the total cost of each node to some value based on the information in Link State Database

Step-2: Now the node selects one node, among all the nodes not in the tree like structure, which is nearest to the root, and adds this to the tree. The shape of the tree gets changed.

Step-3: After this node is added to the tree, the cost of all the nodes not in the tree needs to be updated because the paths may have been changed.

Step-4: The node repeats the Step 2. and Step 3. until all the nodes are added in the tree

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#define LIMIT 10
#define INFINITY 10000
int m,n,k,i,j,dist[LIMIT][LIMIT],sn,dn,min=0;
struct node
{
int hello[LIMIT],from,adj[LIMIT];
}node[LIMIT];
main()
{
clrscr();
printf("Enter how many nodes do u want::");
scanf("%d",&n);
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
dist[i][j]=-1;
for(i=1;i<=n;i++)
{
printf("\nEnter distance for %d node:",i);
for(j=1;j<=n;j++)
if(dist[i][j]==-1)
if(i==j)
{
dist[i][j]=0;
printf(" 0");
}
else
{
scanf("%d",&m);
dist[i][j]=dist[j][i]=m;
}
else
printf(" %d",dist[i][j]);
}
printf("\nDistance matrix is:");
for(i=1;i<=n;i++)
{
printf("\n");
node[i].from=0;
for(j=1;j<=n;j++)
```

```
printf("\t%d",dist[i][j]);
}
printf("\nSENDING HELLO PACKETS");
for(i=1;i<=n;i++)
{
k=1;
for(j=1;j<=n;j++)
{
if(dist[i][j]>0)
node[i].from++;
if(dist[j][i]>0)
node[i].hello[k++]=j;
}
}
for(i=1;i<=n;i++)
{
printf("\nHello packets to node %d:",i);
for(j=1;j<=node[i].from;j++)
printf("\n%d",node[i].hello[j]);
}
printf("\nSENDING ECHO PACKETS");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
if(dist[i][j]>0)
printf("\nfrom %d to %d, the distance is:%d",i,j,dist[i][j]);
}
printf("\n CONSTRUCTION LINKSTATE PACKETS");
for(i=1;i<=n;i++)
{
printf("\nLINK STATE PACKET FOR %d",i);
printf("\n|-------------|");
printf("\n| %d |",i);
printf("\n|-------------|");
printf("\n| seq |");
printf("\n|-------------|");
printf("\n| Age |");
printf("\n|-------------|");
for(j=1;j<=n;j++)
if(dist[i][j]>0)
{
printf("\n| %d | %d |",j,dist[i][j]);
printf("\n|-------------|");
}

}
printf("\nDISTRIBUTING THE LINK STATE PACKETS");
for(i=1;i<=n;i++)
{
printf("\nNeighbours for %d node are:",i);
for(j=1;j<=n;j++)
{
if(dist[i][j]>0)
printf("%d",j);
}
for(j=1;j<=node[i].from;j++)
{
printf("\n the distance from %d to ",i);
printf("%d is %d",node[i].hello[j],dist[i][node[i].hello[j]]);
}
}
printf("\nCOMPUTING THE SHORTEST PATH");
printf("\nEnter source and destination:");
scanf("%d%d",&sn,&dn);
min=spath(sn,dn,min);
printf("\n minimum distance:%d",min);
getch();
}
int spath(int s,int t,int min)
{
struct state
{
int pred;
int len;
enum{permanent,tentative}label;
}state[LIMIT];
int i=1,k;
struct state *p;
for(p=&state[i];p<=&state[n];p++)
{
p->pred=0;
p->len=INFINITY;
p->label=tentative;
}
state[t].len=0;
state[t].label=permanent;
```

```
k=t;
do
{
for(i=1;i<=n;i++)
if(dist[k][i]!=0 && state[i].label==tentative)
{
if(state[k].len+dist[k][i]
{
state[i].pred=k;
state[i].len=state[k].len+dist[k][i];
}
}
k=0;
min=INFINITY;
for(i=1;i<=n;i++)
if(state[i].label==tentative && state[i].len
{
min=state[i].len;
k=i;
}
state[k].label=permanent;
} while(k!=s);
return(min);
}
```

OUTPUT:

```
Enter how many nodes do u want::4

Enter distance for 1 node: 0 2 3 4

Enter distance for 2 node: 2 0 7 8

Enter distance for 3 node: 3 7 0 6

Enter distance for 4 node: 4 8 6 0
Distance matrix is:
0 2 3 4
2 0 7 8
3 7 0 6
```

```
4 8 6 0


SENDING HELLO PACKETS
Hello packets to node 1:
2 3 4
Hello packets to node 2:
1 3 4
Hello packets to node 3:
1 2 4
Hello packets to node 4:
1 2 3
SENDING ECHO PACKETS
from 1 to 2, the distance is:2
from 1 to 3, the distance is:3
from 1 to 4, the distance is:4
from 2 to 1, the distance is:2
from 2 to 3, the distance is:7
from 2 to 4, the distance is:8
from 3 to 1, the distance is:3
from 3 to 2, the distance is:7
from 3 to 4, the distance is:6
from 4 to 1, the distance is:4
from 4 to 2, the distance is:8
from 4 to 3, the distance is:6

CONSTRUCTION LINKSTATE PACKETS
LINK STATE PACKET FOR 1
|-------------|
| 1 |
|-------------|
| seq |
|-------------|
| Age |
|-------------|
| 2 | 2 |
|-------------|
| 3 | 3 |
|-------------|
| 4 | 4 |
LINK STATE PACKET FOR 2
|-------------|
| seq |
|-------------|
| Age |
|-------------|
| 1 | 2 |
|-------------|
| 3 | 7 |
|-------------|
| 4 | 8 |
|-------------|
LINK STATE PACKET FOR 3
|-------------|
| 3 |
|-------------|
| seq |
|-------------|
| Age |
|-------------|
| 1 | 3 |
|-------------|
| 2 | 7 |
|-------------|
| 4 | 6 |
LINK STATE PACKET FOR 4
|-------------|
| 4 |
|-------------|
| seq |
|-------------|
| Age |
|-------------|
| 1 | 4 |
|-------------|
| 2 | 8 |
|-------------|
| 3 | 6 |
|---------------| | 2 | 8 |
|-------------|
| 3 | 6 |
|-------------|
```

DISTRIBUTING THE LINK STATE PACKETS

```
Neighbours for 1 node are:234
the distance from 1 to 2 is 2
the distance from 1 to 3 is 3
the distance from 1 to 4 is 4
Neighbours for 2 node are:134
the distance from 2 to 1 is 2
the distance from 2 to 3 is 7
the distance from 2 to 4 is 8
Neighbours for 3 node are:124
the distance from 3 to 1 is 3
the distance from 3 to 2 is 7
the distance from 3 to 4 is 6
Neighbours for 4 node are:123
the distance from 4 to 1 is 4
the distance from 4 to 2 is 8
the distance from 4 to 3 is 6
COMPUTING THE SHORTEST PATH
Enter source and destination:2 3
minimum distance:5
```

RESULT:

Thus the program for link state routing algorithm using Dijkstra algorithm was implemented by C program successfully.

Ex.No: 9 **Date:**	**PERFORMANCE EVALUATION OF ROUTING PROTOCOLS USING SIMULATION TOOL**

Aim

Perform evaluation of a different routing algorithms to select the network path with its optimum and economical during data transfer.

i. Link State routing

Routing is the process of selecting best paths in a network. In the past, the term routing was also used to mean forwarding network traffic among networks. However this latter function is much better described as simply forwarding. Routing is performed for many kinds of networks, including the telephone network (circuit switching), electronic data networks (such as the Internet), and transportation networks. This article is concerned primarily with routing in electronic data networks using packet switching technology.

In packet switching networks, routing directs packet forwarding (the transit of logically addressed network packets from their source toward their ultimate destination) through intermediate nodes. Intermediate nodes are typically network hardware devices such as routers, bridges, gateways, firewalls, or switches. General-purpose computers can also forward packets and perform routing, though they are not specialized hardware and may suffer from limited performance. The routing process usually directs forwarding on the basis of routing tables which maintain a record of the routes to various network destinations. Thus, constructing routing tables, which are held in the router's memory, is very important for efficient routing. Most routing algorithms use only one network path at a time. Multipath routing techniques enable the use of multiple alternative paths.

In case of overlapping/equal routes, the following elements are considered in order to decide which routes get installed into the routing table (sorted by priority):

1. *Prefix-Length*: where longer subnet masks are preferred (independent of whether it is within a routing protocol or over different routing protocol)
2. *Metric*: where a lower metric/cost is preferred (only valid within one and the same routing protocol)
3. *Administrative distance*: where a lower distance is preferred (only valid between different routing protocols)

Routing, in a more narrow sense of the term, is often contrasted with bridging in its assumption that network addresses are structured and that similar addresses imply proximity within the network. Structured addresses allow a single routing table entry to represent the route to a group of devices. In large networks, structured addressing (routing, in the narrow sense) outperforms unstructured addressing (bridging). Routing has become the dominant form of addressing on the Internet. Bridging is still widely used within localized environments.

ii.Flooding

Flooding s a simple routing algorithm in which every incoming packet is sent through every outgoing link except the one it arrived on.Flooding is used in bridging and in systems such as Usenet and peer-to-peer file sharing and as part of some routing protocols, including OSPF, DVMRP, and those used in ad-hoc wireless networks.There are generally two types of flooding available, Uncontrolled Flooding and Controlled Flooding.Uncontrolled Flooding is the fatal law of flooding. All nodes have neighbours and route packets indefinitely. More than two neighbours creates a broadcast storm.

Controlled Flooding has its own two algorithms to make it reliable, SNCF (Sequence Number Controlled Flooding) and RPF (Reverse Path Flooding). In SNCF, the node attaches its own address and sequence number to the packet, since every node has a memory of addresses and sequence numbers. If it receives a packet in memory, it drops it immediately while in RPF, the node will only send the packet forward. If it is received from the next node, it sends it back to the sender.

Algorithm

There are several variants of flooding algorithm. Most work roughly as follows:

4. Each node acts as both a transmitter and a receiver.
5. Each node tries to forward every message to every one of its neighbours except the source node.

This results in every message eventually being delivered to all reachable parts of the network.

Algorithms may need to be more complex than this, since, in some case, precautions have to be taken to avoid wasted duplicate deliveries and infinite loops, and to allow messages to eventually expire from the system. A variant of flooding called *selective flooding* partially addresses these issues by only sending packets to routers in the same direction. In selective flooding the routers don't send every incoming packet on every line but only on those lines which are going approximately in the right direction.

Advantages

- Since flooding naturally utilizes every path through the network, it will also use the shortest path.
- This algorithm is very simple to implement.

Disadvantages

- Flooding can be costly in terms of wasted bandwidth. While a message may only have one destination it has to be sent to every host. In the case of a ping flood or a denial of service attack, it can be harmful to the reliability of a computer network.

- Messages can become duplicated in the network further increasing the load on the networks bandwidth as well as requiring an increase in processing complexity to disregard duplicate messages.
- Duplicate packets may circulate forever, unless certain precautions are taken:
- Use a hop count or a time to live count and include it with each packet. This value should take into account the number of nodes that a packet may have to pass through on the way to its destination.
- Have each node keep track of every packet seen and only forward each packet once
- Enforce a network topology without loops

iii . Distance vector

In computer communication theory relating to packet-switched networks, a **distance- vector routing protocol** is one of the two major classes of routing protocols, the other major class being the link-state protocol. Distance-vector routing protocols use the Bellman–Ford algorithm, Ford–Fulkerson algorithm, or DUAL FSM (in the case of Cisco Systems's protocols) to calculate paths.

A distance-vector routing protocol requires that a router informs its neighbors of topology changes periodically. Compared to link-state protocols, which require a router to inform all the nodes in a network of topology changes, distance-vector routing protocols have less computational complexity and message overhead.

The term *distance vector* refers to the fact that the protocol manipulates *vectors* (arrays) of distances to other nodes in the network. The vector distance algorithm was the original ARPANET routing algorithm and was also used in the internet under the name of RIP (Routing Information Protocol).

Examples of distance-vector routing protocols include RIPv1 and RIPv2 and IGRP.

Method

Routers using distance-vector protocol do not have knowledge of the entire path to a destination. Instead they use two methods:

1. Direction in which router or exit interface a packet should be forwarded.
2. Distance from its destination

Distance-vector protocols are based on calculating the direction and distance to any link in a network. "Direction" usually means the next hop address and the exit interface. "Distance" is a measure of the cost to reach a certain node. The least cost route between any two nodes is the route with minimum distance. Each node maintains a vector (table) of minimum distance to every node. The cost of reaching a destination is calculated using various route metrics. RIP uses the hop count of the destination whereas IGRP takes into account other information such as node delay and available bandwidth.

Updates are performed periodically in a distance-vector protocol where all or part of a router's routing table is sent to all its neighbors that are configured to use the same distance-vector routing protocol. RIP supports cross-platform distance vector routing whereas IGRP is a Cisco Systems proprietary distance vector routing protocol. Once a router has this information it is able to amend its own routing table to reflect the changes and then inform its neighbors of the changes. This process has been described as ‗routing by rumor' because routers are relying on the information they receive from other routers and cannot determine if the information is actually valid and true. There are a number of features which can be used to help with instability and inaccurate routing information.

EGP and BGP are not pure distance-vector routing protocols because a distance-vector protocol calculates routes based only on link costs whereas in BGP, for example, the local route preference value takes priority over the link cost.

Count-to-infinity problem

The Bellman–Ford algorithm does not prevent routing loops from happening and suffers from the **count-to-infinity problem**. The core of the count-to-infinity problem is that if A tells B that it has a path somewhere, there is no way for B to know if the path has B as a part of it. To see the problem clearly, imagine a subnet connected like A–B–C–D–E–F, and let the metric between the routers be "number of jumps". Now suppose that A is taken offline. In the vector-update-process B notices that the route to A, which was distance 1, is down – B does not receive the vector update from A. The problem is, B also gets an update from C, and C is still not aware of the fact that A is down – so it tells B that A is only two jumps from C (C to B to A), which is false. This slowly propagates through the network until it reaches infinity (in which case the algorithm corrects itself, due to the relaxation property of Bellman–Ford).

RESULT

Perform evaluation of a different routing algorithms to select the network path with its optimum and economical during data transfer was completed.

Ex.No:10

Date:

SIMULATION OF ERROR CORRECTION CODE (like CRC)

AIM:

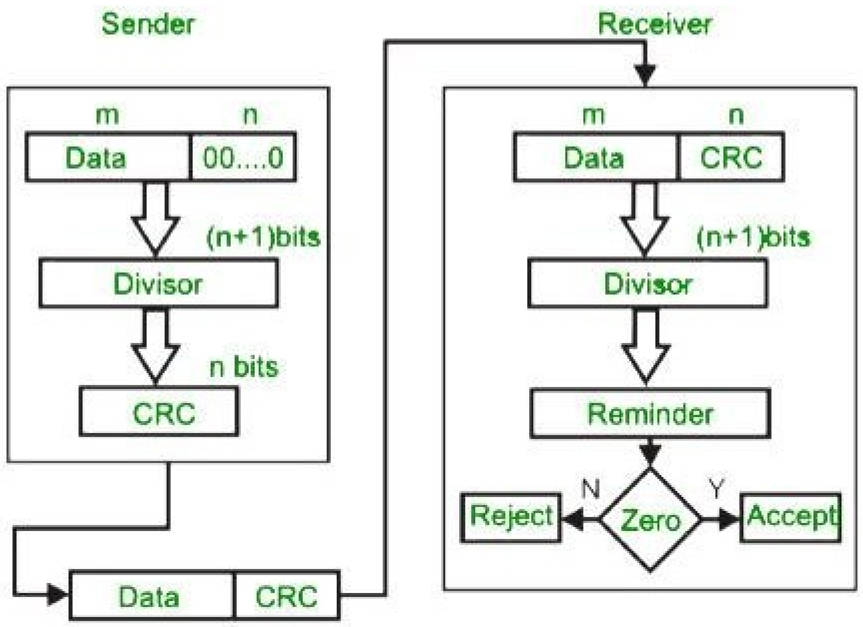
To Simulation of Error Correction Code using C

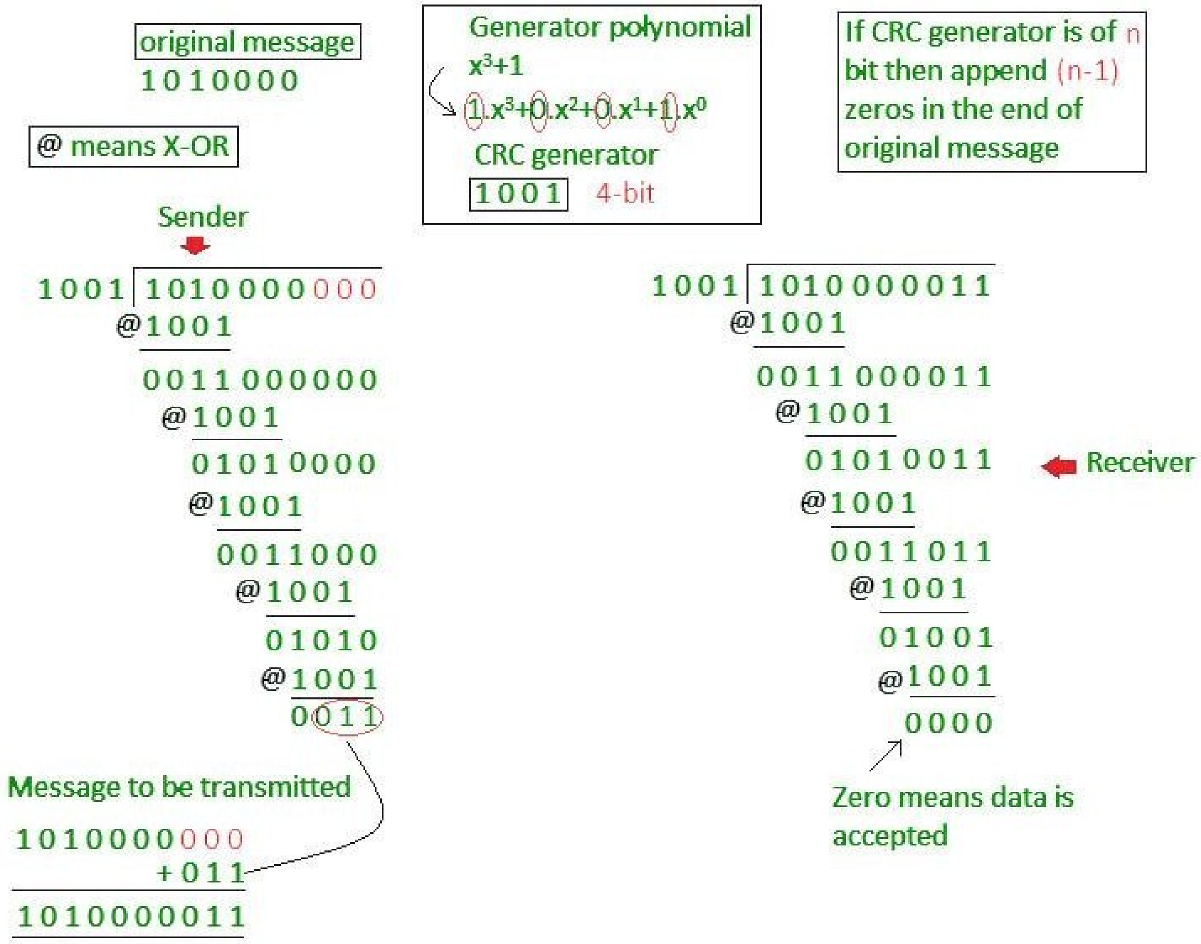
Theory:

Cyclic redundancy check (CRC)

- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- At the destination, the incoming data unit is divided by the same number. If at this step there is no remainder, the data unit is assumed to be correct and is therefore accepted.

A remainder indicates that the data unit has been damaged in transit and therefore must be rejected.





ALGORITHM:

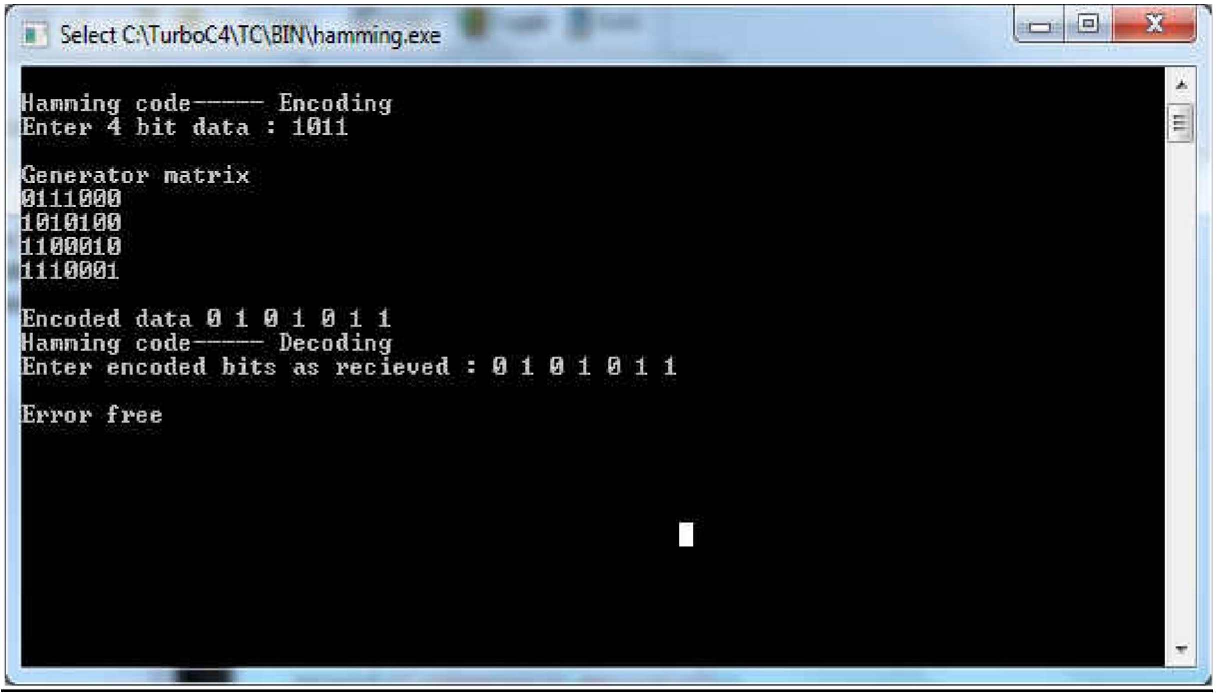
1. Open the editor and type the program for error detection
2. Get the input in the form of bits.
3. Append the redundancy bits.
4. Divide the appended data using a divisor polynomial.
5. The resulting data should be transmitted to the receiver.
6. At the receiver the received data is entered.
7. The same process is repeated at the receiver.
8. If the remainder is zero there is no error otherwise there is some error in the received bits
9. Run the program.

PROGRAM:

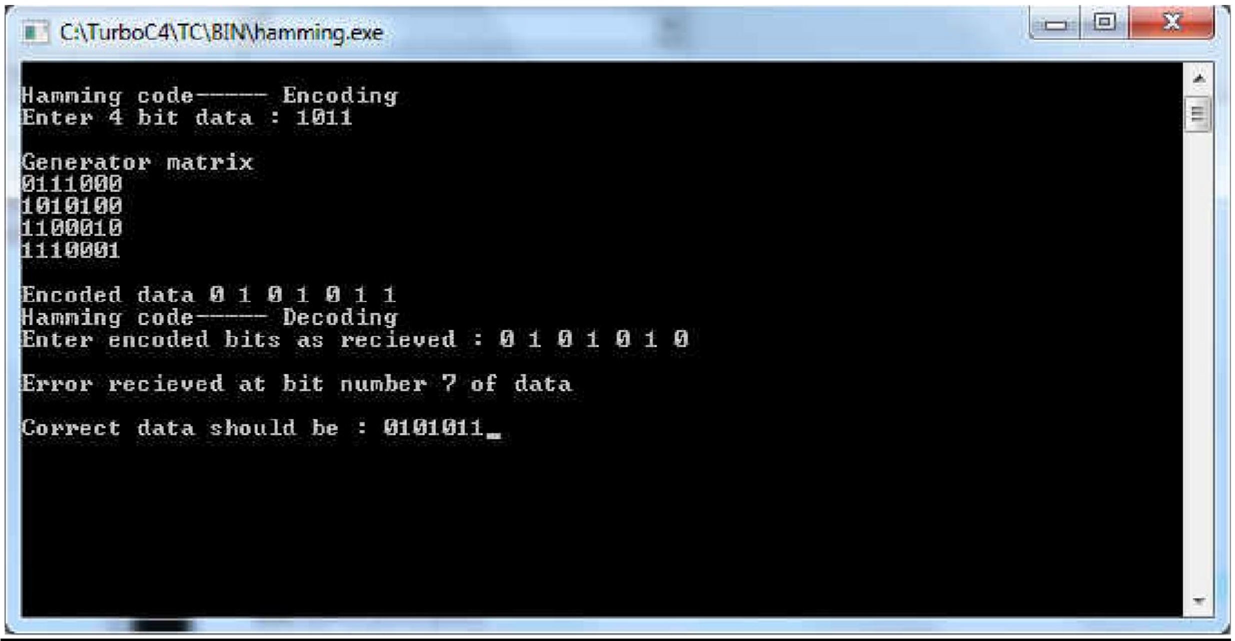
```
#include<stdio.h>
#include<stdlib.h>
char data[5];
int  encoded[8];
int edata[7];
syndrome[3];
int hmatrix[3][7] = {
1,0,0,0,1,1,1,
0,1,0,1,0,1,1,
0,0,1,1,1,0,1
};

char gmatrix[4][8]={"0111000","1010100","1100010","1110001"};

 int main()
 {
    int i,j;
   printf("\nHamming code-----Encoding\n");
   printf("Enter 4 bit data : ");
   scanf("%s",data);
   printf("\nGenerator matrix\
   n");
   for(i=0;i<4;i++)
   printf("%s\n",gmatrix[i]);
   printf("\nEncoded data ");
    for(i=0;i<7;i++)
   {
      for(j=0;j<4;j++)
```

```
        encoded[i]+=((data[j]-'0')*(gmatrix[j][i]-'0'));
        encoded[i]=encoded[i]%2;
        printf("%d ",encoded[i]);
    }
    printf("\nHamming code-----Decoding\n");
    printf("Enter encoded bits as recieved : ");
    for(i=0;i<7;i++)
    scanf("%d",&edata[i]); for(i=0;i<3;i+
    +)
    {
        for(j=0;j<7;j++) syndrome[i]
        +=(edata[j]*hmatrix[i][j]);
        syndrome[i]=syndrome[i]%2;
    }
    for(j=0;j<7;j++)
    if((syndrome[0]==hmatrix[0][j]) && (syndrome[1]==hmatrix[1][j])&& (syndrome[2]==hmatrix[2][j]))
    break;
    if(j==7)
    printf("\nError free\n");
   else
    {
        printf("\nError recieved at bit number %d of data\n",j+1);
        edata[j]=!edata[j];
        printf("\nCorrect data should be : ");
        for(i=0;i<7;i++)
        printf("%d",edata[i]);
    }
    getch();
    return 0;

}
```

OUTPUT:



```
Hamming code----- Encoding
Enter 4 bit data : 1011

Generator matrix
0111000
1010100
1100010
1110001

Encoded data 0 1 0 1 0 1 1
Hamming code----- Decoding
Enter encoded bits as recieved : 0 1 0 1 0 1 1

Error free
```



```
Hamming code----- Encoding
Enter 4 bit data : 1011

Generator matrix
0111000
1010100
1100010
1110001

Encoded data 0 1 0 1 0 1 1
Hamming code----- Decoding
Enter encoded bits as recieved : 0 1 0 1 0 1 0

Error recieved at bit number 7 of data

Correct data should be : 0101011_
```

RESULT:

Thus the program of Error Detection and Correction was implemented successfully.