

Ex.no: 1

BASIC NETWORK COMMANDS

Date:

AIM:

To execute an all types of command in command prompt

ALGORITHM:

Step 1: Open a terminal or command prompt.

Step 2: Use the "ping" command to test network connectivity to a specified host or IP address.

Step 3: Use the "tracert" or "tracert" command to trace the route of packets from the local system to a specified host or IP address.

Step 4: Use the "nslookup" or "dig" command to query DNS (Domain Name System) for information about a specified domain name or IP address.

Step 5: Use the "netstat" command to display active network connections, including open ports and listening services.

Step 6: Use the "ifconfig" or "ipconfig" command to display information about the network interfaces on the local system, including IP address, subnet mask, and MAC address.

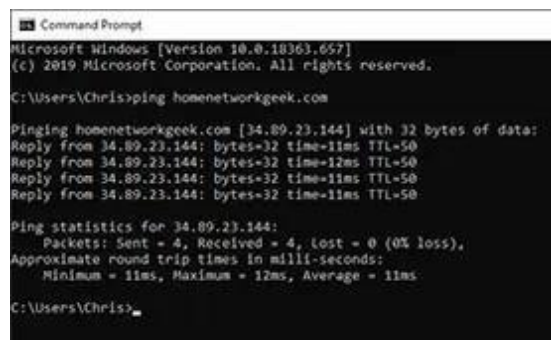
Step 7: Use the "tcpdump" or "wireshark" command-line tools to capture and analyze network traffic in real-time.

NETWORK COMMANDS:

1.PING

Command to enter: **ping**

Ping is used to test whether one network host is able to communicate with another



```
Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>ping homenetworkgeek.com

Pinging homenetworkgeek.com [34.89.23.144] with 32 bytes of data:
Reply from 34.89.23.144: bytes=32 time=11ms TTL=50
Reply from 34.89.23.144: bytes=32 time=12ms TTL=50
Reply from 34.89.23.144: bytes=32 time=11ms TTL=50
Reply from 34.89.23.144: bytes=32 time=11ms TTL=50

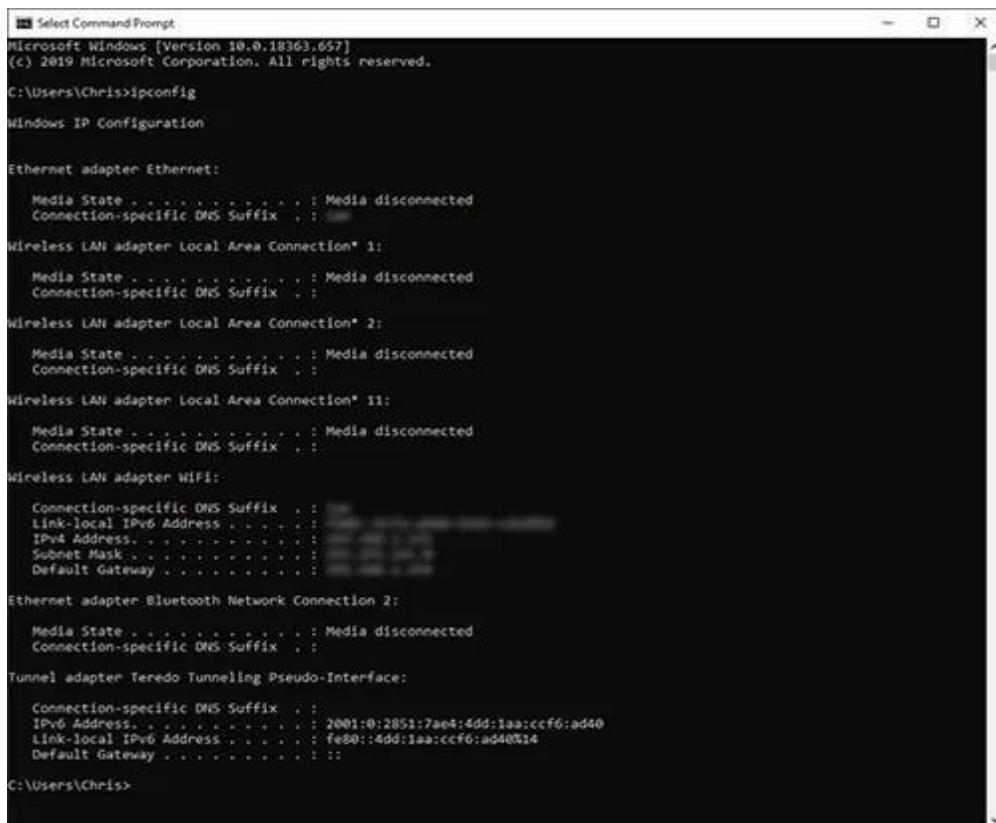
Ping statistics for 34.89.23.144:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 11ms, Maximum = 12ms, Average = 11ms

C:\Users\Chris>
```

2.IPCONFIG

Command to enter: **ipconfig**

The ipconfig command displays basic IP address configuration information for the Windows device you are working on.



```
Select Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : 

Wireless LAN adapter Local Area Connection* 1:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : 

Wireless LAN adapter Local Area Connection* 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : 

Wireless LAN adapter Local Area Connection* 11:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : 

Wireless LAN adapter WiFi:

    Connection-specific DNS Suffix . : 
    Link-local IPv6 Address . . . . . : 
    IPv4 Address. . . . . : 
    Subnet Mask . . . . . : 
    Default Gateway . . . . . : 

Ethernet adapter Bluetooth Network Connection 2:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix . : 

Tunnel adapter Teredo Tunneling Pseudo-Interface:

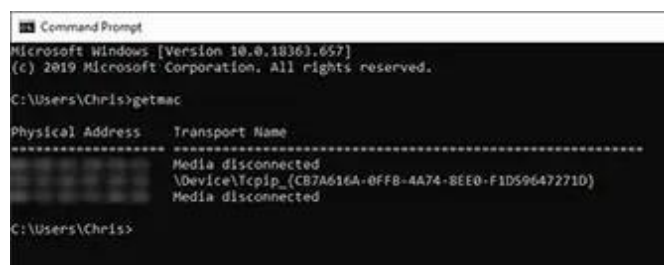
    Connection-specific DNS Suffix . : 
    IPv6 Address. . . . . : 2001:0:2051:7ae4:4dd:1aa:ccf6:ad40
    Link-local IPv6 Address . . . . . : fe80::4dd:1aa:ccf6:ad40%14
    Default Gateway . . . . . : 

C:\Users\Chris>
```

3.GETMAC

Command to enter: **getmac**

The getmac command provides an easy way to find the MAC address of your device.If you see more than one MAC address for your device, it will have multiple network adapters. As an example, a laptop with both Ethernet and Wi-Fi will have two separate MAC addresses.



```
Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>getmac

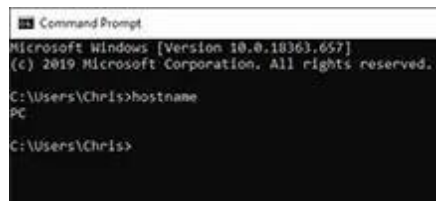
Physical Address    Transport Name
-----
Media disconnected  \Device\NPF{CB7A616A-0FFB-4A74-BEE0-F1D59647271D}
Media disconnected

C:\Users\Chris>
```

4.HOSTNAME:

Command to enter: **hostname**

The hostname command provides you with an easy way of identifying the hostname that has been assigned to your Windows device.



```
Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>hostname
PC

C:\Users\Chris>
```

5.ARP

Command to enter: **arp -a**

ARP stands for Address Resolution Protocol and the command is used to map an IP address to a MAC address.



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>arp

Displays and modifies the IP-to-Physical address translation tables used by
address resolution protocol (ARP).

ARP -s inet_addr eth_addr [if_addr]
ARP -d inet_addr [if_addr]
ARP -a [inet_addr] [-N if_addr] [-v]

-a          Displays current ARP entries by interrogating the current
            protocol data. If inet_addr is specified, the IP and Physical
            addresses for only the specified computer are displayed. If
            more than one network interface uses ARP, entries for each ARP
            table are displayed.
-g          Same as -a.
-v          Displays current ARP entries in verbose mode. All invalid
            entries and entries on the loop-back interface will be shown.
-i          Specifies an Internet address.
-N if_addr  Displays the ARP entries for the network interface specified
            by if_addr.
-d          Deletes the host specified by inet_addr. inet_addr may be
            wildcarded with * to delete all hosts.
-s          Adds the host and associates the Internet address inet_addr
            with the Physical address eth_addr. The Physical address is
            given as 6 hexadecimal bytes separated by hyphens. The entry
            is permanent.
eth_addr    Specifies a physical address.
if_addr     If present, this specifies the Internet address of the
            interface whose address translation table should be modified.
            If not present, the first applicable interface will be used.

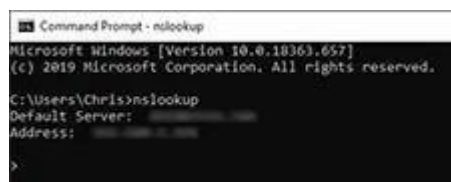
Example:
> arp -s 157.55.85.212 00-aa-00-62-c6-09 .... Adds a static entry.
> arp -a          .... Displays the arp table.

C:\Users\Chris>
```

6. NSLOOKUP:

Command to enter: **nslookup**

NSLookup is useful for diagnosing DNS name resolution problems



```
Command Prompt - nslookup
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

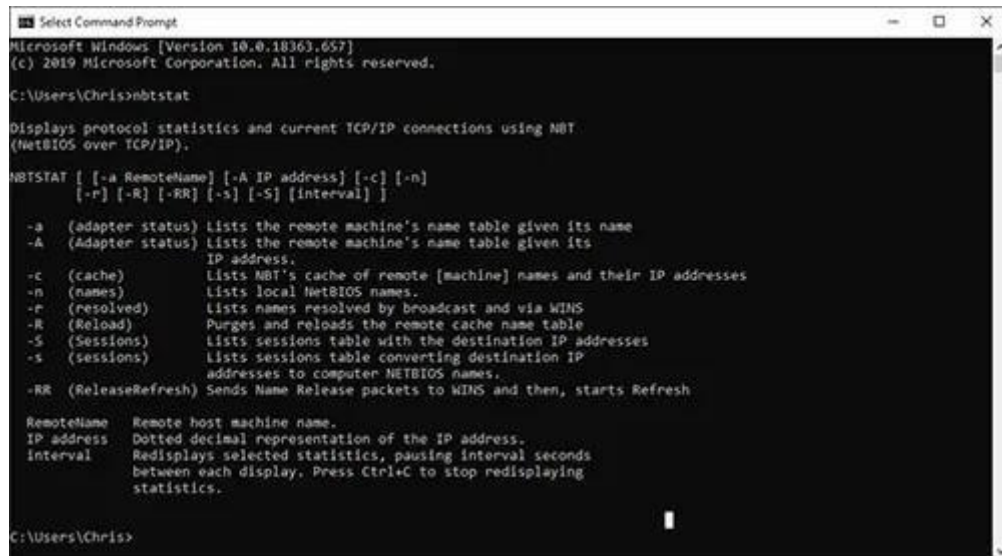
C:\Users\Chris>nslookup
Default Server: 192.168.1.1
Address: 192.168.1.1

>
```

7. NBTSTAT:

Command to enter: **nbtstat**

Windows uses different methods to associate NetBIOS names with IP addresses; these include broadcast and LMHost lookup.



```
Select Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>nbtstat

Displays protocol statistics and current TCP/IP connections using NBT
(NetBIOS over TCP/IP).

NBTSTAT [ [-a RemoteName] [-A IP address] [-c] [-n]
          [-r] [-R] [-RR] [-s] [-S] [interval] ]

-a (adapter status) Lists the remote machine's name table given its name
-A (Adapter status) Lists the remote machine's name table given its
IP address.
-c (cache) Lists NBT's cache of remote [machine] names and their IP addresses
-n (names) Lists local NetBIOS names.
-r (resolved) Lists names resolved by broadcast and via WINS
-R (Reload) Purges and reloads the remote cache name table
-s (Sessions) Lists sessions table with the destination IP addresses
-S (sessions) Lists sessions table converting destination IP
addresses to computer NETBIOS names.
-RR (ReleaseRefresh) Sends Name Release packets to WINS and then, starts Refresh

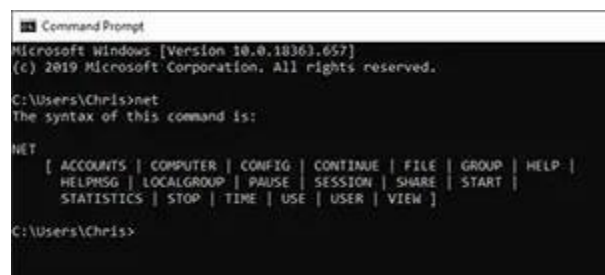
RemoteName Remote host machine name.
IP address Dotted decimal representation of the IP address.
interval Redisplays selected statistics, pausing interval seconds
between each display. Press Ctrl+C to stop redisplaying
statistics.

C:\Users\Chris>
```

8. NET:

Command to enter: **net**

The net command is definitely a versatile one, allowing you to manage many different aspects of a network and its settings such as network shares, users and print jobs



```
Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>net

The syntax of this command is:

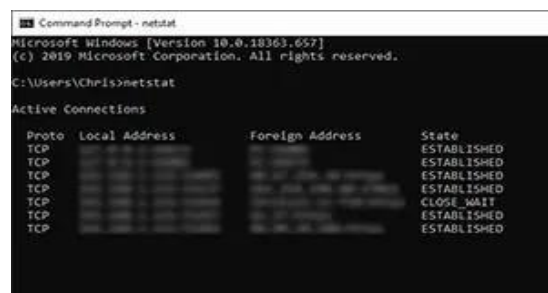
NET
[ ACCOUNTS | COMPUTER | CONFIG | CONTINUE | FILE | GROUP | HELP |
  HELPHSG | LOCALGROUP | PAUSE | SESSION | SHARE | START |
  STATISTICS | STOP | TIME | USE | USER | VIEW ]

C:\Users\Chris>
```

9. NETSTAT:

Command to enter: **netstat**

Run netstat and you'll see a list of active connections, with more being added every few seconds. It will describe the protocol being used, the local address, the foreign address, and the connection state



```
Command Prompt - netstat
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>netstat

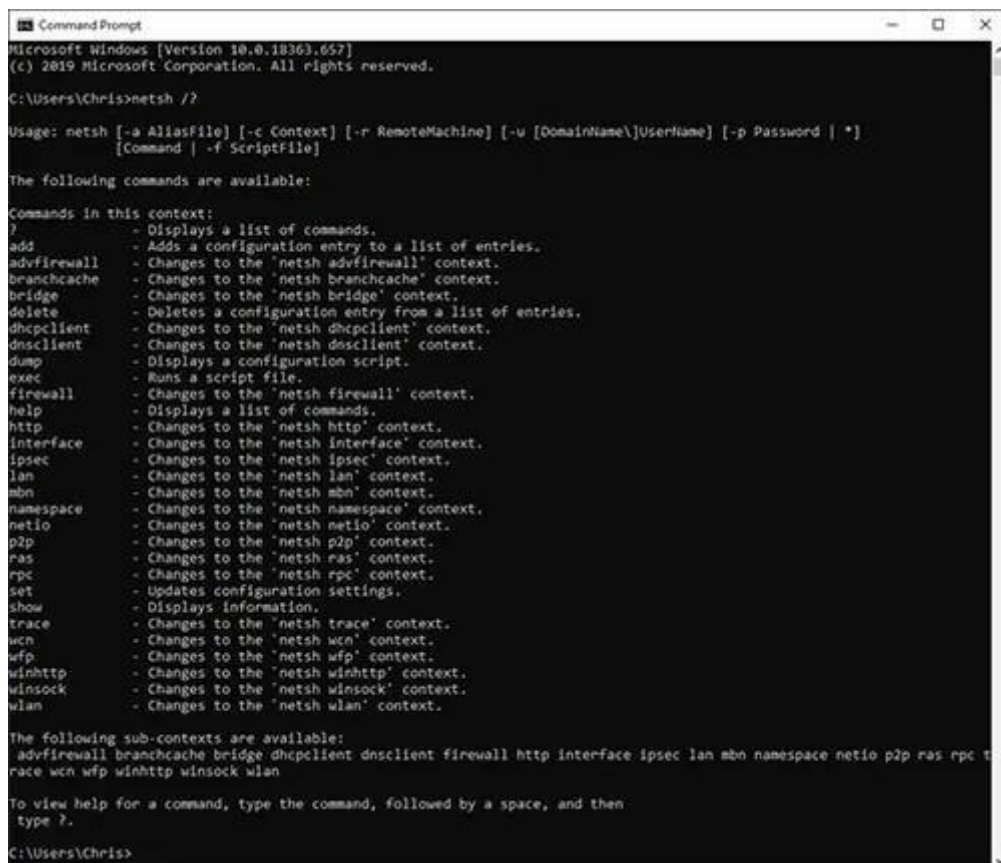
Active Connections

Proto Local Address Foreign Address State
TCP 0.0.0.0:80 0.0.0.0: LISTENING
TCP 0.0.0.0:443 0.0.0.0: LISTENING
TCP 0.0.0.0:8080 0.0.0.0: LISTENING
TCP 192.168.1.10:80 192.168.1.1: ESTABLISHED
TCP 192.168.1.10:443 192.168.1.1: ESTABLISHED
TCP 192.168.1.10:8080 192.168.1.1: ESTABLISHED
TCP 192.168.1.10:80 192.168.1.1: CLOSE_WAIT
TCP 192.168.1.10:80 192.168.1.1: ESTABLISHED
TCP 192.168.1.10:80 192.168.1.1: ESTABLISHED
```

10. NETSH:

Command to enter: **netsh**

When you run the netsh command on its own, the command prompt will be shifted into network shell mode. Within this mode, there are several different “contexts”, such as one for DHCP-related commands, one for diagnostics and one for routing.



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>netsh /?

Usage: netsh [-a AliasFile] [-c Context] [-r RemoteMachine] [-u [DomainName\]UserName] [-p Password | *]
[Command] [-f ScriptFile]

The following commands are available:

Commands in this context:
) - Displays a list of commands.
add - Adds a configuration entry to a list of entries.
advfirewall - Changes to the 'netsh advfirewall' context.
branchcache - Changes to the 'netsh branchcache' context.
bridge - Changes to the 'netsh bridge' context.
delete - Deletes a configuration entry from a list of entries.
dhcpcclient - Changes to the 'netsh dhcpcclient' context.
dnsclient - Changes to the 'netsh dnsclient' context.
dump - Displays a configuration script.
exec - Runs a script file.
firewall - Changes to the 'netsh firewall' context.
help - Displays a list of commands.
http - Changes to the 'netsh http' context.
interface - Changes to the 'netsh interface' context.
ipsec - Changes to the 'netsh ipsec' context.
lan - Changes to the 'netsh lan' context.
mbn - Changes to the 'netsh mbn' context.
namespace - Changes to the 'netsh namespace' context.
netio - Changes to the 'netsh netio' context.
p2p - Changes to the 'netsh p2p' context.
ras - Changes to the 'netsh ras' context.
rpc - Changes to the 'netsh rpc' context.
set - Updates configuration settings.
show - Displays information.
trace - Changes to the 'netsh trace' context.
wcn - Changes to the 'netsh wcn' context.
wfp - Changes to the 'netsh wfp' context.
winhttp - Changes to the 'netsh winhttp' context.
winsock - Changes to the 'netsh winsock' context.
wlan - Changes to the 'netsh wlan' context.

The following sub-contexts are available:
advfirewall branchcache bridge dhcpcclient dnsclient firewall http interface ipsec lan mbn namespace netio p2p ras rpc t
race wcn wfp winhttp winsock wlan

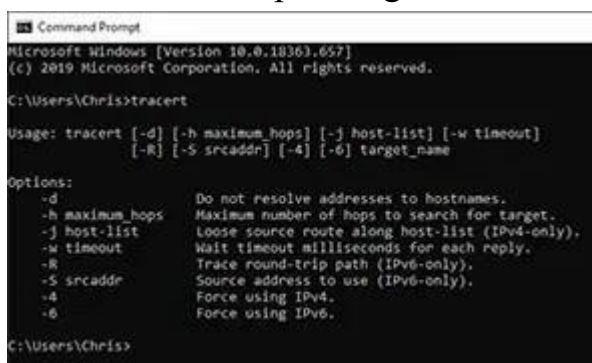
To view help for a command, type the command, followed by a space, and then
type ?.

C:\Users\Chris>
```

11. TRACERT:

Command to enter: **tracert**

By using the tracert command you can trace the route a packet takes before reaching its destination, and see information on each “hop” along the route



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>tracert

Usage: tracert [-d] [-h maximum_hops] [-j host-list] [-w timeout]
[-R] [-S srcaddr] [-4] [-6] target_name

Options:
-d Do not resolve addresses to hostnames.
-h maximum_hops Maximum number of hops to search for target.
-j host-list Loose source route along host-list (IPv4-only).
-w timeout Wait timeout milliseconds for each reply.
-R Trace round-trip path (IPv6-only).
-S srcaddr Source address to use (IPv6-only).
-4 Force using IPv4.
-6 Force using IPv6.

C:\Users\Chris>
```

12. TASKKILL:

Command to enter: **taskkill**

Instead of using task manager, we may use taskkill to kill the process in command prompt.



```
Select Command Prompt
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>taskkill /?

TASKKILL [/S system [/U username [/P [password]]]]
    { [/FI filter] [/PID processid | /IM imagename] } [/T] [/F]

Description:
    This tool is used to terminate tasks by process id (PID) or image name.

Parameter List:
    /S system           Specifies the remote system to connect to.
    /U [domain\]user    Specifies the user context under which the
                        command should execute.
    /P [password]       Specifies the password for the given user
                        context. Prompts for input if omitted.
    /FI filter          Applies a filter to select a set of tasks.
                        Allows "*" to be used. ex. imagename eq acme*
    /PID processid      Specifies the PID of the process to be terminated.
                        Use Tasklist to get the PID.
    /IM imagename       Specifies the image name of the process
                        to be terminated. Wildcard "*" can be used
                        to specify all tasks or image names.
    /T                 Terminates the specified process and any
                        child processes which were started by it.
    /F                 Specifies to forcefully terminate the process(es).
    /?                 Displays this help message.

Filters:
    Filter Name  Valid Operators  Valid Value(s)
    -----
    STATUS      eq, ne                RUNNING |
                        NOT RESPONDING | UNKNOWN
    IMAGENAME   eq, ne                Image name
    PID         eq, ne, gt, lt, ge, le  PID value
    SESSION    eq, ne, gt, lt, ge, le  Session number.
    CPU TIME   eq, ne, gt, lt, ge, le  CPU time in the format
                        of hh:mm:ss.
                        hh - hours,
                        mm - minutes, ss - seconds
    MEMUSAGE   eq, ne, gt, lt, ge, le  Memory usage in KB
    USERNAME   eq, ne                User name in [domain\]user
                        format
    MODULES    eq, ne                DLL name
    SERVICES   eq, ne                Service name
    WINDOWTITLE eq, ne                Window title

NOTE
----
1) Wildcard "*" for /IM switch is accepted only when a filter is applied.
2) Termination of remote processes will always be done forcefully (/F).
3) "WINDOWTITLE" and "STATUS" filters are not considered when a remote
   machine is specified.

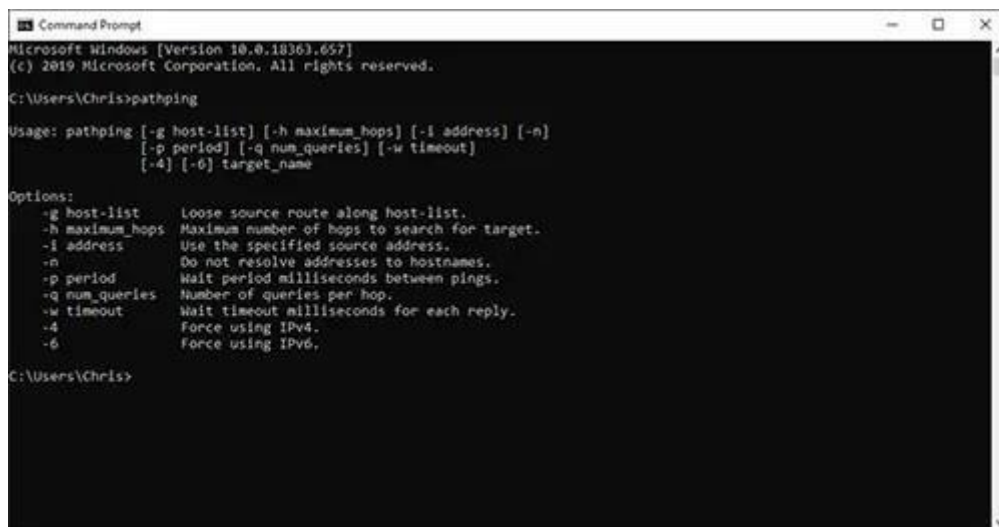
Examples:
TASKKILL /IM notepad.exe
TASKKILL /PID 1230 /PID 1241 /PID 1253 /T
TASKKILL /F /IM cmd.exe /T
TASKKILL /F /FI "PID ge 1000" /FI "WINDOWTITLE ne untitled*"
TASKKILL /F /FI "USERNAME eq NT AUTHORITY\SYSTEM" /IM notepad.exe
TASKKILL /S system /U domain\username /FI "USERNAME ne NT*" /IM *
TASKKILL /S system /U username /P password /FI "IMAGENAME eq note*"

C:\Users\Chris>
```

13. PATHPING:

Command to enter: **pathping**

Pathping combines that best of both ping and tracert into a single utility. Enter pathping followed by a hostname into the command prompt and it will initiate what looks like a regular old tracert command



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>pathping

Usage: pathping [-g host-list] [-h maximum_hops] [-i address] [-n]
               [-p period] [-q num_queries] [-w timeout]
               [-4] [-6] target_name

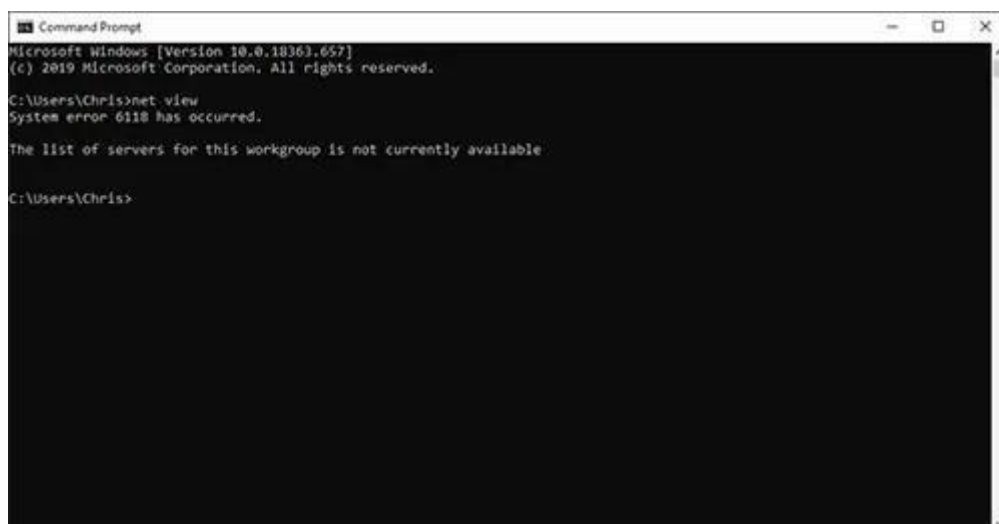
Options:
  -g host-list      Loose source route along host-list.
  -h maximum_hops  Maximum number of hops to search for target.
  -i address        Use the specified source address.
  -n               Do not resolve addresses to hostnames.
  -p period         Wait period milliseconds between pings.
  -q num_queries    Number of queries per hop.
  -w timeout        Wait timeout milliseconds for each reply.
  -4               Force using IPv4.
  -6               Force using IPv6.

C:\Users\Chris>
```

14. NET VIEW:

Command to enter: **net view**

The command will be presented with a list of devices that are connected to the same network as you



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>net view
System error 6118 has occurred.

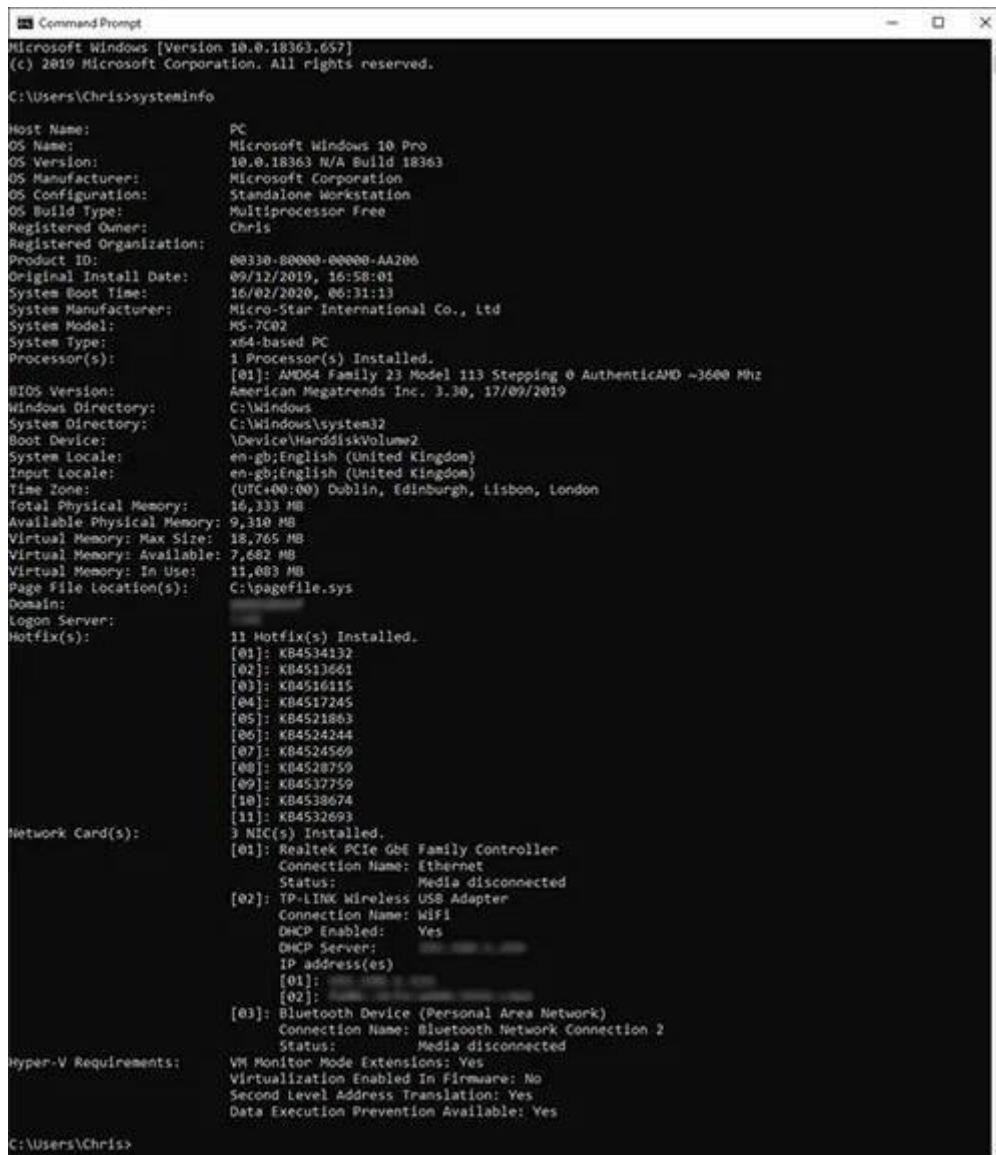
The list of servers for this workgroup is not currently available.

C:\Users\Chris>
```


15. SYSTEMINFO:

Command to enter: **systeminfo**

This command will poll your device and display the most important information in a clean, easy to read format.



```
Microsoft Windows [Version 10.0.18363.657]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Chris>systeminfo

Host Name:                             PC
OS Name:                               Microsoft Windows 10 Pro
OS Version:                            10.0.18363 N/A Build 18363
OS Manufacturer:                       Microsoft Corporation
OS Configuration:                      Standalone Workstation
OS Build Type:                          Multiprocessor Free
Registered Owner:                       Chris
Registered Organization:
Product ID:                             00330-80000-00000-AA206
Original Install Date:                  09/12/2019, 16:58:01
System Boot Time:                       16/02/2020, 06:31:13
System Manufacturer:                   Micro-Star International Co., Ltd
System Model:                           MS-7C02
System Type:                            x64-based PC
Processor(s):                           1 Processor(s) Installed.
[01]: AMD64 Family 23 Model 113 Stepping 0 AuthenticAMD ~3600 Mhz
BIOS Version:                           American Megatrends Inc. 3.30, 17/09/2019
Windows Directory:                     C:\Windows
System Directory:                       C:\Windows\system32
Boot Device:                            \Device\HarddiskVolume2
System Locale:                           en-gb:English (United Kingdom)
Input Locale:                           en-gb:English (United Kingdom)
Time Zone:                              (UTC+00:00) Dublin, Edinburgh, Lisbon, London
Total Physical Memory:                  16,333 MB
Available Physical Memory:               9,310 MB
Virtual Memory: Max Size:                18,765 MB
Virtual Memory: Available:               7,682 MB
Virtual Memory: In Use:                  11,083 MB
Page File Location(s):                  C:\pagefile.sys
Domain:
Logon Server:
Hotfix(s):                              11 Hotfix(s) Installed.
[01]: KB4534132
[02]: KB4513661
[03]: KB4516115
[04]: KB4517245
[05]: KB4521863
[06]: KB4524244
[07]: KB4524569
[08]: KB4528759
[09]: KB4537759
[10]: KB4538674
[11]: KB4532693
Network Card(s):                        3 NIC(s) Installed.
[01]: Realtek PCIe GbE Family Controller
Connection Name: Ethernet
Status: Media disconnected
[02]: TP-LINK Wireless USB Adapter
Connection Name: WIFI
DHCP Enabled: Yes
DHCP Server:
IP address(es)
[01]:
[02]:
[03]: Bluetooth Device (Personal Area Network)
Connection Name: Bluetooth Network Connection 2
Status: Media disconnected
Hyper-V Requirements: VM Monitor Mode Extensions: Yes
Virtualization Enabled In Firmware: No
Second Level Address Translation: Yes
Data Execution Prevention Available: Yes

C:\Users\Chris>
```

RESULT:

These commands help monitor, troubleshoot, configure network settings, and transfer files to ensure efficient network operation.

Ex.no: 2

HTTP web client

Date:

AIM:

To develop a Java program that uses the requests module to send an HTTP GET request to download a web page from a specified URL.

ALGORITHM:

Step 1: Import the requests module to send HTTP requests.

Step 2: Define the URL of the web page to download.

Step 3: Send an HTTP GET request to the URL using the requests.get() method.

Step 4: Retrieve the response from the HTTP request and store it in a variable.

Step 5: Print the status code of the response to confirm the request was successful.

Step 6: Optionally, save the HTML content of the web page to a file or manipulate the data as needed.

CLIENT:

- 1 Start the program, declare the variables
- 2 Create a socket using the socket structure socket(AF_INET, SOCK_STREAM,0)
- 3 Set the socket family, IP address and the port using the server address
- 4 Set the socket address of 8 bytes to zero using the memset() function
- 5 Establish the connection to the server
- 6 Read web address to be downloaded and send it to the server as a request
- 7 Receive the message from the server for a search result
- 8 Display the file which is downloaded
- 9 Terminate the connection and compile and execute.

SERVER:

- 1 Start the program, declare the variables
- 2 Create a socket using the socket structure socket(AF_INET, SOCK_STREAM,0)
- 3 Set the socket family, IP address and the port using the server address
- 4 Set the socket address of 8 bytes to zero using the memset() function
- 5 Bind and listen the socket structure.

PROGRAM:

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

public class Webpage {

    public static void DownloadWebPage(String webpage){

        try {

            // Create URL object

            URL url = new URL(webpage);

            BufferedReader readr =

                new BufferedReader(new InputStreamReader(url.openStream()));

            // Enter filename in which you want to download

            BufferedWriter writer =

                new BufferedWriter(new FileWriter("Download.html"));

            // read each line from stream till end

            String line;

            while ((line = readr.readLine()) != null) {

                writer.write(line);}

            readr.close();

            writer.close();

            System.out.println("Successfully Downloaded.");}

        // Exceptions

        catch (MalformedURLException mue) {

            System.out.println("Malformed URL Exception raised");}

        catch (IOException ie) {

            System.out.println("IOException raised");}

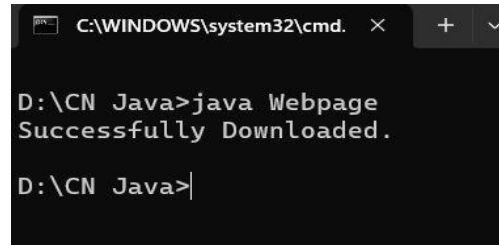
    }

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

```
String url = "https://www.espn.in/";  
DownloadWebPage(url);} }  
}
```

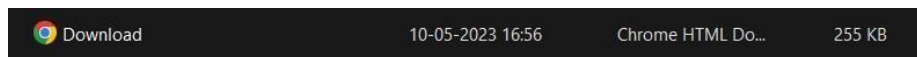
OUTPUT:

Screenshot 1-command to run program



```
C:\WINDOWS\system32\cmd.  
D:\CN Java>java Webpage  
Successfully Downloaded.  
D:\CN Java>
```

Screenshot 2- web page downloaded successfully



Screenshot 3- downloaded webpage



RESULT:

Thus, the java program to implement socket for HTTP for web page upload and download was executed successfully and output was verified

Ex.no: 3

a) TCP ECHO SERVER/CLIENT

Date:

AIM:

To implement Echo Server/Client in java using TCP socket.

ALGORITHM:

Server:

1. Create a server socket.
2. Wait for client to be connected.
3. Read text from the client
4. Echo the text back to the client.
5. Repeat steps 4-5 until 'bye' or 'null' is read.
6. Close the I/O streams
7. Close the server socket
8. Stop

Client:

1. Create a socket and establish connection with the server
2. Get input from user.
3. If equal to bye or null, then go to step 7.
4. Send text to the server.
5. Display the text echoed by the server
6. Repeat steps 2-4 .Close the I/O streams and Close the client socket
7. Stop

PROGRAM:

tcpechoserver.java

```
import java.net.*;
import java.io.*;
public class tcpechoserver
```

```

{
public static void main(String[] arg) throws IOException
{
ServerSocket sock = null; BufferedReader fromClient = null; OutputStreamWriter toClient =
null; Socket client = null;

try
{
sock = new ServerSocket(4000); System.out.println("Server Ready");
client = sock.accept();
System.out.println("Client Connected");
fromClient = new BufferedReader(new
InputStreamReader(client.getInputStream()));
toClient = new OutputStreamWriter(client.getOutputStream());
String line;
while (true)
{
line = fromClient.readLine();
if ( (line == null) || line.equals("bye")) break;
System.out.println ("Client [ " + line + " ]"); toClient.write("Server [ "+ line + " ]\n");
toClient.flush();
}
fromClient.close(); toClient.close(); client.close(); sock.close();
System.out.println("Client Disconnected");
}
catch (IOException ioe)
{
System.err.println(ioe);
}}
}

```

tcpechoclient.java

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

public class tcpechoclient
{
    public static void main(String[] args) throws IOException
    {
        BufferedReader fromServer = null, fromUser = null;
        PrintWriter toServer = null;
        Socket sock = null;

        try
        {
            if (args.length == 0)
                sock = new Socket(InetAddress.getLocalHost(), 4000);
            else
                sock = new Socket(InetAddress.getByName(args[0]), 4000);
            fromServer = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            fromUser = new BufferedReader(new InputStreamReader(System.in));
            toServer = new PrintWriter(sock.getOutputStream(), true);
            String Usrmsg, Srvmsg;
            System.out.println("Type \"bye\" to quit");
            while (true)
            {
                System.out.print("Enter msg to server : "); Usrmsg = fromUser.readLine();
                if (Usrmsg == null || Usrmsg.equals("bye"))
                {
                    toServer.println("bye");
                    break;
                }
            }
        }
    }
}
```



```
else toServer.println(Usrmsg);  
Srvmsg = fromServer.readLine(); System.out.println(Srvmsg);  
}  
fromUser.close();  
fromServer.close(); toServer.close(); sock.close();  
}  
catch (IOException ioe)  
{  
System.err.println(ioe);  
}}}
```

OUTPUT:

Server Console:

```
$ javac tcpechoserver.java $ java tcpechoserver Server Ready  
Client Connected Client [ hello ] Client [ how are you ] Client [ i am fine ] Client [ ok ]  
Client Disconnected
```

Client Console:

```
$ javac tcpechoclient.java $ java tcpechoclient  
Type "bye" to quit  
Enter msg to server : hello Server [ hello ]  
Enter msg to server : how are you Server [ how are you ]  
Enter msg to server : i am fine Server [ i am fine ]  
Enter msg to server : ok Server [ ok ]  
Enter msg to server : bye
```

RESULT:

Thus data from client to server is echoed back to the client to check reliability/noise level of the channel

Ex.no: 3

b) TCP CHAT SERVER/CLIENT

Date:

AIM:

To implement a chat server and client in java using TCP sockets

ALGORITHM:

Server:

1. Create a server socket
2. Wait for client to be connected.
3. Read Client's message and display it
4. Get a message from user and send it to client
5. Repeat steps 3-4 until the client sends "end"
6. Close all streams
7. Close the server and client socket
8. Stop

Client:

1. Create a client socket and establish connection with the server
2. Get a message from user and send it to server
3. Read server's response and display it
4. Repeat steps 2-3 until chat is terminated with "end" message
5. Close all input/output streams
6. Close the client socket
7. Stop

PROGRAM:

tcpchatserver.java

```
import java.io.*;
import java.net.*;
class tcpchatserver
{
    public static void main(String args[])throws Exception
    {
        PrintWriter toClient;
        BufferedReader fromUser, fromClient;

        try
        {
            ServerSocket Srv = new ServerSocket(5555);
            System.out.print("\nServer started\n");
            Socket Clt = Srv.accept();
            System.out.println("Client connected");
            toClient = new PrintWriter(new BufferedWriter(new
            OutputStreamWriter(Clt.getOutputStream())), true);
            fromClient = new BufferedReader(new InputStreamReader(Clt.getInputStream()));
            fromUser = new BufferedReader(new InputStreamReader(System.in));
            String CltMsg, SrvMsg;
            while(true)
            {
                CltMsg= fromClient.readLine();
                if(CltMsg.equals("end"))
                    break;
                else
                {
                    System.out.println("\nServer <<< " + CltMsg);
```

```

System.out.print("Message to Client : ");

SrvMsg = fromUser.readLine();
toClient.println(SrvMsg);
}
}

System.out.println("\nClient Disconnected");
fromClient.close();
toClient.close();
fromUser.close();
Clt.close();
Srv.close();
}

catch (Exception E)
{
System.out.println(E.getMessage());
}
}
}

```

tcpchatclient.java

```

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

class tcpchatclient
{
public static void main(String args[])throws Exception
{
Socket Clt;

PrintWriter toServer;

BufferedReader fromUser, fromServer;

try

```

```

{
if (args.length > 1)
{
System.out.println("Usage: java hostipaddr");
System.exit(-1);
}
if (args.length == 0)
Clc = new Socket(InetAddress.getLocalHost(),5555); else
Clc = new Socket(InetAddress.getByName(args[0]), 5555);
toServer = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(Clc.getOutputStream())), true);
fromServer = new BufferedReader(new InputStreamReader(Clc.getInputStream()));
fromUser = new BufferedReader(new InputStreamReader(System.in));
String ClcMsg, SrvMsg; System.out.println("Type \"end\" to Quit");
while (true)
{
System.out.print("\nMessage to Server : ");
ClcMsg = fromUser.readLine();
toServer.println(ClcMsg);
if (ClcMsg.equals("end")) break;
SrvMsg = fromServer.readLine();
System.out.println("Client <<< " + SrvMsg);
}
}
catch(Exception E)
{
System.out.println(E.getMessage());
}}

```

OUTPUT:

Server Console:

```
$ javac tcpchatserver.java
```

```
$ java tcpchatserver
```

Server started

Client connected

Server <<< hi

Message to Client : hello

Server <<< how r u?

Message to Client : fine

Server <<< me too

Message to Client : bye

Client Disconnected

Client Console:

```
$ javac tcpchatclient.java
```

```
$ java tcpchatclient
```

Type "end" to Quit

Message to Server : hi

Client <<< hello

Message to Server : how r u?

Client <<< fine

Message to Server : me too

Client <<< bye

Message to Server : end

RESULT:

Thus, both the client and server exchange data using TCP socket programming.

Ex.no: 4

UDP DNS SERVER/CLIENT

Date:

AIM:

To implement a DNS server and client in java using UDP sockets.

ALGORITHM:

Server:

1. Define an array of hosts and its corresponding IP address in another array
2. Create a datagram socket
3. Create a datagram packet to receive client request
4. Read the domain name from client to be resolved
5. Lookup the host array for the domain name
6. If found then retrieve corresponding address
7. Construct a datagram packet to send response back to the client
8. Repeat steps 3-7 to resolve further requests from clients
9. Close the server socket
10. Stop

Client:

1. Create a datagram socket
2. Get domain name from user
3. Construct a datagram packet to send domain name to the server
4. Create a datagram packet to receive server message
5. If it contains IP address then display it, else display "Domain does not exist"
6. Close the client socket
7. Stop

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 Console:

```
$ javac udpdnsserver.java
```

```
$ java udpdnsserver Press Ctrl + C to Quit
```

Request for host yahoo.com Request for host cricinfo.com Request for host youtube.com

Client Console:

```
$ javac udpdnsclient.java
```

```
$ java udpdnsclient
```

```
Enter the hostname : yahoo.com IP Address: 68.180.206.184
```

```
$ java udpdnsclient
```

```
Enter the hostname : cricinfo.com IP
```

```
Address: 80.168.92.140
```

```
$ java udpdnsclient
```

```
Enter the hostname : youtube.com
```

```
IP Address: Host Not Found
```

RESULT:

Thus domain name requests by the client are resolved into their respective logical address using lookup method

INTRODUCTION:

Wireshark is a free and open-source network protocol analyzer used for troubleshooting, analysis, software and protocol development, and education. It allows users to capture and view network traffic in real-time and analyze network packets to identify potential issues or vulnerabilities. Wireshark supports a wide range of protocols and can be used on various platforms including Windows, macOS, and Linux.

Wireshark captures packets in real-time and displays them in a user-friendly graphical interface. It supports hundreds of protocols and allows users to drill down to the lowest levels of a network packet to analyze individual fields and bits. Additionally, it offers powerful filtering capabilities, allowing users to focus on specific network traffic patterns or issues.

AIM:

To capture and examine packets in my computer network using Wireshark.

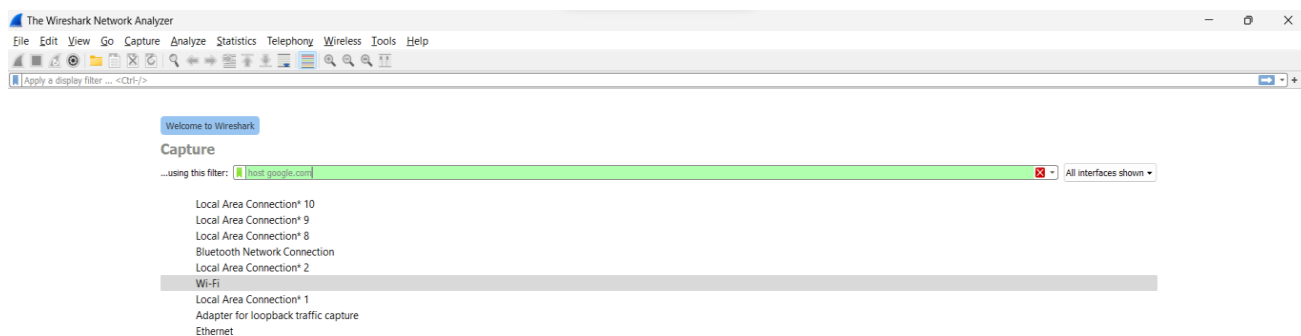
ALGORITHM:

1. Download and install Wireshark on your computer.
2. Launch Wireshark and select the network interface you want to capture packets from.
3. Start the packet capture by clicking the "Capture" button in the Wireshark user interface.
4. Use the Wireshark filters to capture only the packets you are interested in examining.
5. Stop the packet capture when you have captured enough packets for your analysis.
6. Analyse the captured packets by examining the packet headers and contents.
7. Use the various Wireshark tools and features to analyse the packet data, such as the packet list, packet details, and packet tree views.
8. Save the captured packet data and analysis results for further analysis or future reference.

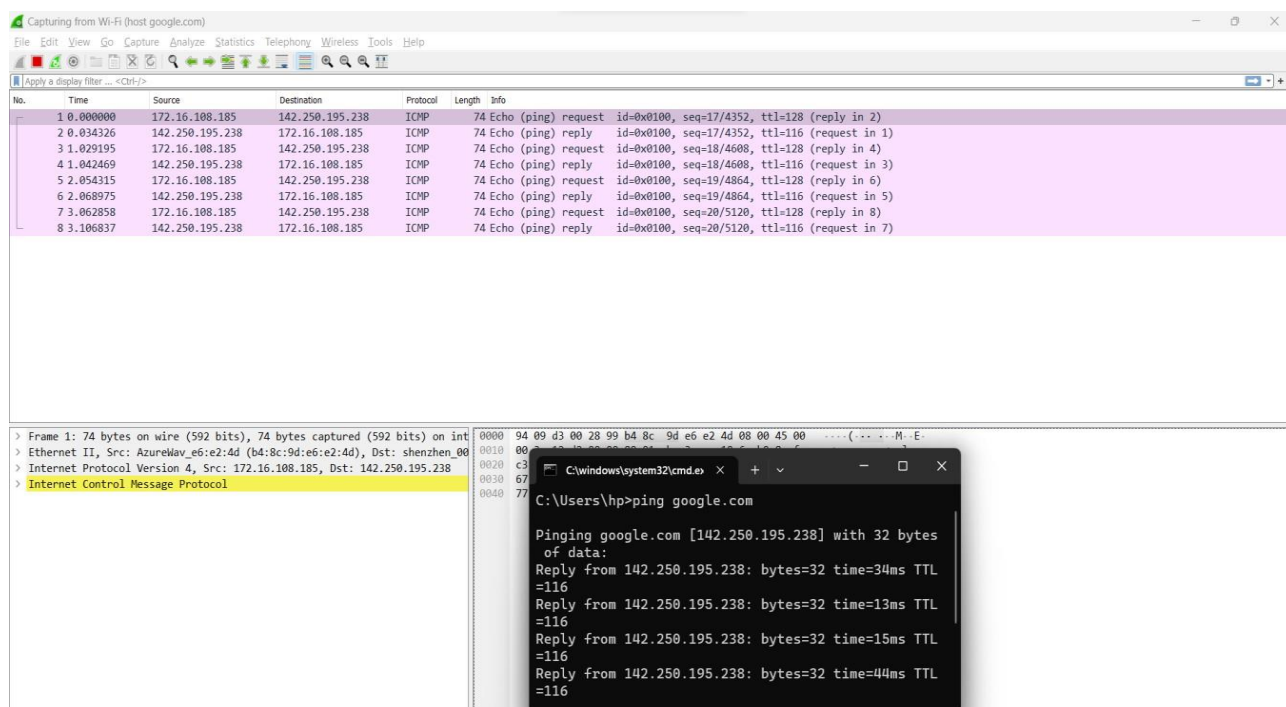
STEPS:

Open Wireshark and select the network interface you want to capture packets from.

1. Click the "Start" button in Wireshark to begin capturing packets.
2. Perform the network activity you want to capture packets for



3. Click the "Stop" button in Wireshark to stop capturing packets.
4. Analyse the captured packets by filtering and searching for specific data, such as IP addresses or protocols.
5. Identify any issues or anomalies in the network activity and troubleshoot as needed.



RESULT:

Thus, the packets were captured and examined using Wireshark tool

Ex.no: 6

a) ARP CLIENT/SERVER

Date:

AIM:

To know the physical address of a host when its logical address is known using ARP protocol.

ALGORITHM:

Server:

1. Create a server socket.
2. Accept client connection.
3. Read IP address from the client request
4. Check its configuration file and compare with its logical address.
5. If there is a match, send the host physical address.
6. Stop

Client:

1. Create a socket.
2. Send IP address to the target machine
3. Receive target's response
4. If it is a MAC address then display it and go to step 6
5. Display "Host not found"
6. Stop

PROGRAM:

Arpclient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientarp{
public static void main(String args[]){
try{
```

```

BufferedReader in=new
BufferedReader(new
InputStreamReader(System.in));
Socket clsct=new
Socket("127.0.0.1",139);
DataInputStream din=new DataInputStream(clsct.getInputStream());
DataOutputStream dout=new DataOutputStream(clsct.getOutputStream());
System.out.println("Enter the Logical address(IP):");
String str1=in.readLine();
dout.writeBytes(str1+'\n');
String str=din.readLine();
System.out.println("The Physical Address is: "+str);
clsct.close();}
catch (Exception e){
System.out.println(e);
}
}
}

```

arpserver.java

```

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

```

```

DataOutputStream dout=new DataOutputStream(obj1.getOutputStream());

String str=din.readLine();

String ip[]={"192.168.4.215","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)

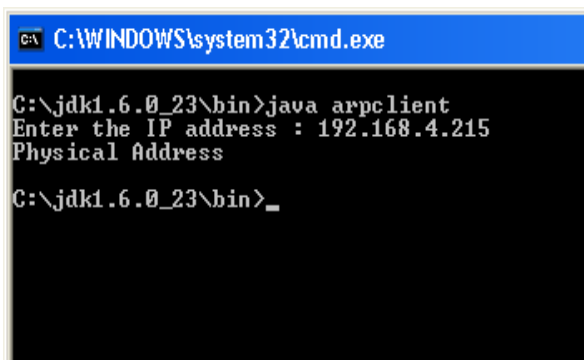
{

    System.out.println(e);

}}}}

```

OUTPUT:



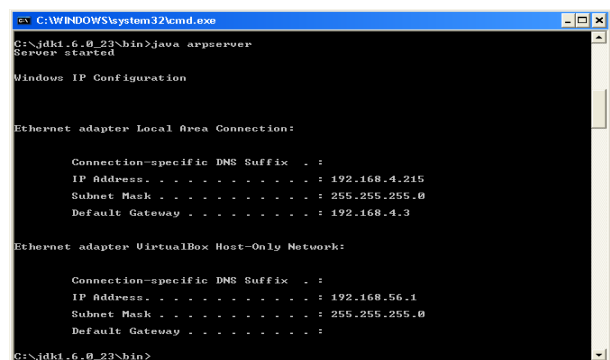
```

C:\ C:\WINDOWS\system32\cmd.exe

C:\jdk1.6.0_23\bin>java arpclient
Enter the IP address : 192.168.4.215
Physical Address

C:\jdk1.6.0_23\bin>_

```



```

C:\ C:\WINDOWS\system32\cmd.exe

C:\jdk1.6.0_23\bin>java arpserver
Server started

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.4.215
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.4.3

Ethernet adapter VirtualBox Host-Only Network:

    Connection-specific DNS Suffix  . : 
    IP Address. . . . . : 192.168.56.1
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 

C:\jdk1.6.0_23\bin>

```

RESULT:

Thus, using ARP protocol MAC address is obtained.

Ex.no: 6

b) RARP CLIENT/SERVER

Date:

AIM:

To know the logical address of a host when its physical address is known using RARP protocol.

ALGORITHM:

Server:

1. Create a server socket.
2. Accept client connection.
3. Read MAC address from the client request
4. Check its configuration file and compare with its physical address.
5. If there is a match, send the host logical address.
6. Stop

Client:

1. Create a socket.
2. Send physical address to the target machine
3. Receive target's response
4. If it is a IP address then display it and go to step 6
5. Display "Host not found"
6. Stop

Program:

Rarpserver.java

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

```

class Serverrarp
{
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<mac.length;i++)
{
if(str.equals(mac[i]))
{
dout.writeBytes(ip[i]+'\\n');
break;
}
}
obj.close();
}
}
catch(Exception e)
{
System.out.println(e);
}}
}

```

rarpclient.java

```
import java.io.*;
import java.net.*;
import java.util.*;
class Clientrarp
{
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 Physical Addres (MAC):");
String str1=in.readLine();
dout.writeBytes(str1+"\n");
String str=din.readLine();
System.out.println("The Logical address is(IP): "+str);
clsct.close();
}
catch (Exception e)
{
System.out.println(e);
}
}
}
```


OUTPUT:

Server:

```
$ javac rarpserver.java
```

```
$ java rarpserver
```

```
Server started
```

```
eth0 Link encap:Ethernet HWaddr B8:AC:6F:1B:AB:06
```

```
ip addr: 165.165.80.80 Bcast:172.255.255.255 Mask:255.0.0.0
```

```
inet6 addr: fe80::baac:6fff:fe1b:ab06/64 Scope:Link
```

```
UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1 RX packets:450 errors:0  
dropped:0 overruns:0 frame:0
```

```
TX packets:127 errors:0 dropped:0 overruns:0 carrier:0 collisions:0 txqueuelen:1000
```

```
RX bytes:48118 (46.9 KiB) TX bytes:21025 (20.5 KiB) Interrupt:16
```

Client:

```
$ javac rarpclient.java
```

```
$ java rarpclient
```

```
Enter the physical address : B8:AC:6F:1B:AB:06 Logical Address 165.165.80.80
```

RESULT:

Thus using RARP protocol IP address is obtained

Ex.no: 7

Date:

STUDY OF NETWORK SIMULATOR [NS2]

Aim:

The aim of studying Network Simulator (NS2) is to gain practical experience and understand network simulation principles, protocols, algorithms, and performance for wired and wireless networks.

Introduction:

A simulator is a device, software or system which behaves or operates like a given system when provided with a set of controlled inputs.

The need for simulators is:

- ✓ Provide users with practical feedback such as accuracy, efficiency, cost, etc., when designing real world systems.
- ✓ Permit system designers to study at several different levels of abstraction
- ✓ Simulation can give results that are not experimentally measurable with our current level of technology.
- ✓ Simulations take the building/rebuilding phase out of the loop by using the model already created in the design phase. Effective means for teaching or demonstrating concepts to students.

A few popular network simulators are NS-2, OPNET, GLOMOSIM, etc.

Network Simulator NS2:

NS2 is an object-oriented, discrete event driven network simulator developed at UC Berkley written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks. NS2 is an open-source simulation tool that primarily runs on Linux (cygwin for Windows). The features of NS2 are

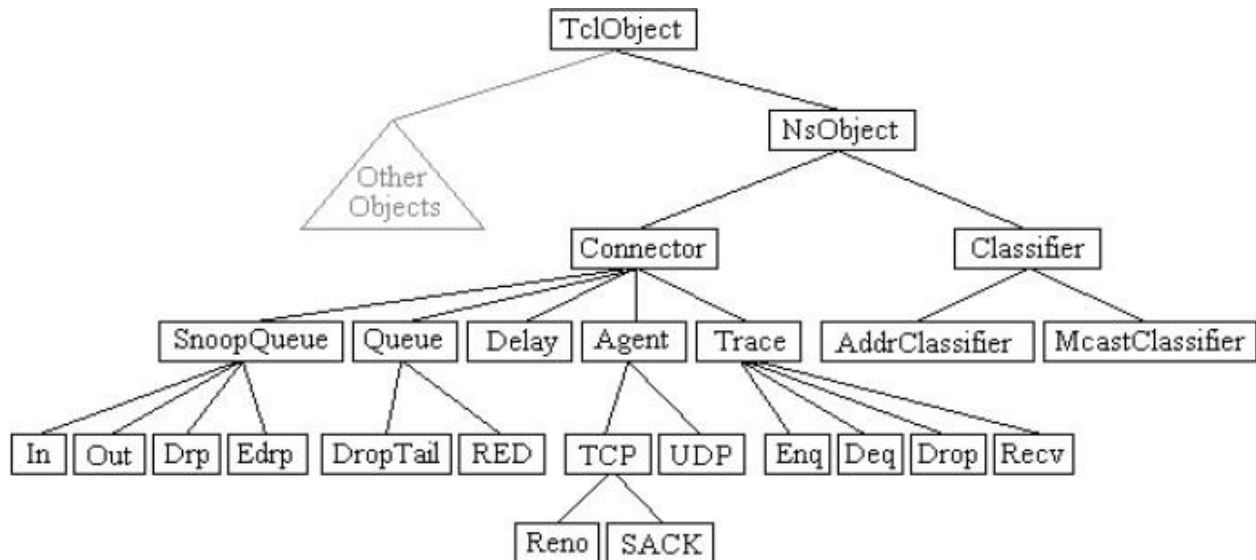
- ✓ Is a discrete event simulator for networking research
- ✓ Works at packet level.
- ✓ Provide support to simulate bunch of protocols like TCP, UDP, FTP, etc.
- ✓ Simulate wired and wireless network. Is a standard experiment environment in research community.

Network Animator (NAM):

NS together with its companion, NAM, form a very powerful set of tools for teaching networking concepts. NS contains all the IP protocols typically covered in undergraduate and

most graduate courses, and many experimental protocols contributed by its ever-expanding user base. With NAM, these protocols can be visualized as animations. The NAM graphical editor is the latest addition to NAM. With this editor, one can create their network topology and simulate various protocols and traffic sources by dragging the mouse.

NS2 Class Hierarchy:



The root of the hierarchy is the **TclObject** class that is the superclass of all OTcl library objects (scheduler, network components, timers and the other objects including NAM related ones). As an ancestor class of **TclObject**, **NsObject** class is the superclass of all basic network component objects that handle packets, which may compose compound network objects such as nodes and links. The basic network components are further divided into two subclasses, **Connector** and **Classifier**, based on the number of the possible output data paths. The basic network objects that have only one output data path are under the **Connector** class, and switching objects that have possible multiple output data paths are under the **Classifier** class.

Running NS2:

- ✓ A OTcl script is generally composed as follows:
- ✓ Create a new simulator object
- ✓ Turn on tracing
- ✓ Create network (physical layer)
- ✓ Create link and queue (data-link layer)
- ✓ Define routing protocol
- ✓ Create transport connection (transport layer)
- ✓ Create traffic (application layer)
- ✓ Insert errors

PROGRAM:

```
# Create a simulator object
set ns [new Simulator]

# Define different colors
# for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

# Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

# Define a 'finish' procedure
proc finish {} {
    global ns nf
    $ns flush-trace
    # Close the NAM trace file
    close $nf
    # Execute NAM on the trace file
    exec nam out.nam &
    exit 0
}

# Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

# Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

# Set Queue Size of link (n2-n3) to 10
```

```

$ns queue-limit $n2 $n3 10

# Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

# Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

# Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n3 $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 type_ CBR

```

```

$cbr set packet_size_ 1000

$cbr set rate_ 1mb
$cbr set random_ false

# Schedule events for the CBR and FTP agents

$ns at 0.1 "$cbr start"

$ns at 1.0 "$ftp start"

$ns at 4.0 "$ftp stop"

$ns at 4.5 "$cbr stop"

# Detach tcp and sink agents

# (not really necessary)

$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

# Call the finish procedure after

# 5 seconds of simulation time

$ns at 5.0 "finish"

# Print CBR packet size and interval

puts "CBR packet size = [$cbr set packet_size_]"

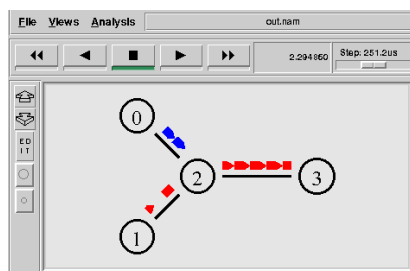
puts "CBR interval = [$cbr set interval_]"

# Run the simulation

$ns run

```

OUTPUT:



RESULT:

Thus, the concepts of Network Simulator [NS2] were studied successfully

Ex.no: 8

a) STUDY OF TCP PERFORMANCE

Date:

AIM:

To study and simulate the performance of TCP using NS2

Transmission Control Protocol (TCP):

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.

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.

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.

3. 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.

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.

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.

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.

TCP ports

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.

ALGORITHM:

1. Create a simulator object.
2. Open the nam and trace output files.
3. Create network nodes A, R, and B.
4. Create duplex links between A and R, and between R and B, with specified bandwidth and delay.
5. Set the queue size at node R to 7 packets, including the packet being sent.
6. Create a TCP sending agent and attach it to node A, and a TCP receive agent (a traffic sink) and attach it to node B.
7. Schedule the TCP connection to start sending data at T=0 and stop at T=10.0. Trace some variables for the sending agent, such as congestion window size and maximum sequence number.
8. Run the simulation and save the trace output to the nam and trace files. Close the files when finished.

PROGRAM:

```
# basic1.tcl simulation: A---R---B
#Create a simulator object
set ns [new Simulator]

#Open the nam file basic1.nam and the variable-trace file basic1.tr
set namfile [open basic1.nam w]

$ns namtrace-all $namfile

set tracefile [open basic1.tr w]

$ns trace-all $tracefile

#Define a 'finish' procedure
```

```

proc finish { } {
    global ns namfile tracefile
    $ns flush-trace
    close $namfile
    close $tracefile
    exit 0
}

#Create the network nodes
set A [$ns node]
set R [$ns node]
set B [$ns node]

#Create a duplex link between the nodes
$ns duplex-link $A $R 10Mb 10ms DropTail
$ns duplex-link $R $B 800Kb 50ms DropTail

# The queue size at $R is to be 7, including the packet being sent
$ns queue-limit $R $B 7

# some hints for nam
# color packets of flow 0 red
$ns color 0 Red

$ns duplex-link-op $A $R orient right
$ns duplex-link-op $R $B orient right
$ns duplex-link-op $R $B queuePos 0.5

# Create a TCP sending agent and attach it to A
set tcp0 [new Agent/TCP/Reno]

# We make our one-and-only flow be flow 0
$tcp0 set class_ 0
$tcp0 set window_ 100
$tcp0 set packetSize_ 960
$ns attach-agent $A $tcp0

# Let's trace some variables

```

```

$tcp0 attach $tracefile
$tcp0 tracevar cwnd_
$tcp0 tracevar ssthresh_
$tcp0 tracevar ack_
$tcp0 tracevar maxseq_

#Create a TCP receive agent (a traffic sink) and attach it to B
set end0 [new Agent/TCPSink]
$ns attach-agent $B $end0

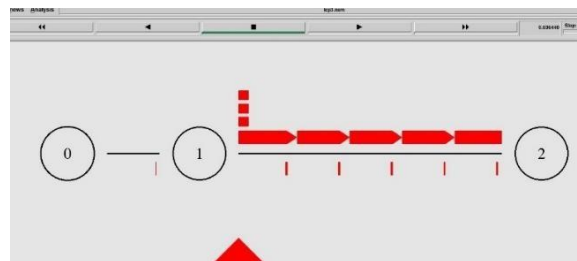
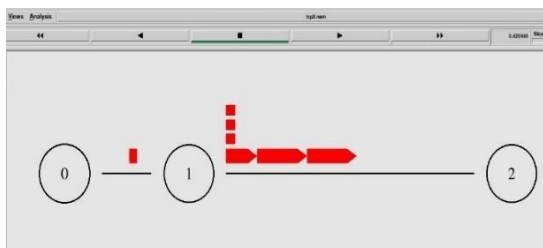
#Connect the traffic source with the traffic sink
$ns connect $tcp0 $end0

#Schedule the connection data flow; start sending data at T=0, stop at T=10.0
set myftp [new Application/FTP]
$myftp attach-agent $tcp0
$ns at 0.0 "$myftp start"
$ns at 10.0 "finish"

#Run the simulation
$ns run

```

OUTPUT:



RESULT:

Thus, the study of TCP performance and simulation using NS2 was performed successfully.

Ex.no: 8

b) STUDY OF UDP PERFORMANCE

Date:

AIM:

To study and simulate the performance of UCP using NS2.

User Datagram Protocol (UDP):

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 handshaking 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

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. 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.

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.

Comparison of UDP and TCP

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.

ALGORITHM:

1. Create a new network simulator instance and open trace files.
2. Create network nodes using node() function.
3. Create network links using duplex-link() and duplex-link-op() functions. Attach agents and applications to nodes:
 - a. Create UDP agent and CBR application and attach them to node n2. Set packet size and interval for CBR.
 - b. Create UDP agent and CBR application and attach them to node n1. Set packet size and interval for CBR.
 - c. Create Null agent and attach it to node n5.
 - d. Create TCP agent and CBR application and attach them to node n0. Set packet size and interval for CBR.
 - e. Create TCPSink agent and attach it to node n7.
4. Set link colors using color() function.
5. Schedule events using at() function:
 - a. Start and stop CBR applications for n2 and n1 agents.
 - b. Start and stop CBR application for TCP agent.
 - c. Define a finish procedure to clean up and close trace files.
6. Run the simulation using run() function.

PROGRAM:

```
set ns [new Simulator]

set nr [open thro.dt.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 DropTail
$ns duplex-link $n1 $n3 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n4 $n6 1Mb 10ms DropTail
$ns duplex-link $n4 $n7 1Mb 10ms DropTail
```


\$ns duplex-link-op \$n0 \$n3 orient right-up
\$ns duplex-link-op \$n1 \$n3 orient right
\$ns duplex-link-op \$n2 \$n3 orient right-down
\$ns duplex-link-op \$n3 \$n4 orient middle
\$ns duplex-link-op \$n4 \$n5 orient right-up
\$ns duplex-link-op \$n4 \$n7 orient right-down
\$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 udpl [new Agent/UDP]
\$ns attach-agent \$n1 \$udpl
set cbr1 [new Application/Traffic/CBR]
\$cbr1 set packetSize_ 500
\$cbr1 set interval_ 0.005
\$cbr1 attach-agent \$udpl
set null1 [new Agent/Null]
\$ns attach-agent \$n6 \$null1
set tcp0 [new Agent/TCP]
\$ns attach-agent \$n0 \$tcp0
set cbr2 [new Application/Traffic/CBR]
\$cbr2 set packetSize_ 500
\$cbr2 set interval_ 0.005
\$cbr2 attach-agent \$tcp0

```

set tcpsink0 [new Agent/TCPSink]

$ns attach-agent $n7 $tcpsink0

$ns connect $tcp0 $tcpsink0

$udp0 set fid_ 1
$udp1 set fid_ 2
$tcp0 set fid_ 3

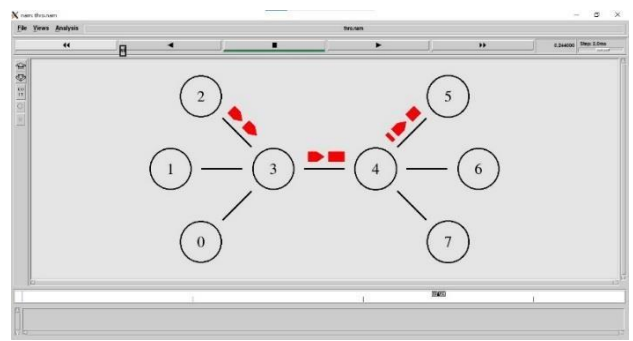
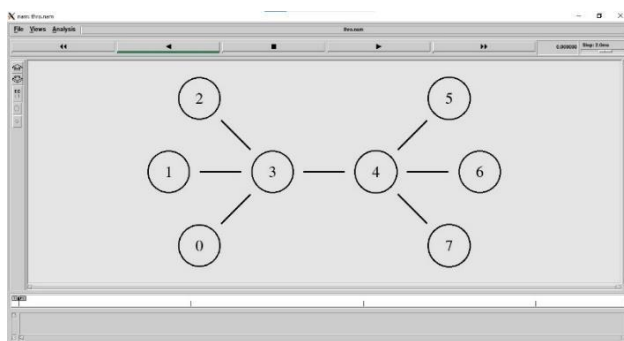
$ns color 1 Red
$ns color 2 Green
$ns color 3 Blue

$ns at 0.2 "$cbr0 start"
$ns at 3.5 "$cbr0 stop"
$ns at 0.3 "$cbr1 start"
$ns at 4.5 "$cbr1 stop"
$ns at 0.4 "$cbr2 start"
$ns at 4.5 "$cbr2 stop"
$ns at 5.0 "finish"

$ns run

```

OUTPUT:



RESULT:

Thus, the study of UCP performance and simulation using NS2 was performed successfully.

Ex.no: 9

a) SIMULATION OF DISTANCE VECTOR ROUTING PROTOCOL

Date:

AIM:

To create distance vector routing protocol using NS2 simulator.

ALGORITHM:

Step: 1. Create a simulator object ns.

Step: 2. Create 4 nodes namely n0 to n3, then create the duplex link between these nodes using the object ns.

Step:3. Assign node positions for each and every node.

Step:4. Set the queue size for the link and monitor the queue to avoid packet loss.

Step:5. Create agent and sources for each end nodes to establish the TCP connection.

Step:6. Schedule events for the CBR and FTP agents based on routing information protocol.

Step:7. Create finish procedure.

Step:8. Run the simulator.

PROGRAM:

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
set tr [open out.tr w]
$ns trace-all $tr
proc finish {} {
    global nf ns tr
    $ns flush-trace
    close $tr
    exec nam out.nam &
    exit 0
}
```

```

}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n0 $n1 10Mb 10ms DropTail
$ns duplex-link $n1 $n3 10Mb 10ms DropTail
$ns duplex-link $n2 $n1 10Mb 10ms DropTail
$ns duplex-link-op $n0 $n1 orient right-down
$ns duplex-link-op $n1 $n3 orient right
$ns duplex-link-op $n2 $n1 orient right-up
set tcp [new Agent/TCP]
$ns attach-agent $n0 $tcp
set ftp [new Application/FTP]
$ftp attach-agent $tcp
set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
set udp [new Agent/UDP]
$ns attach-agent $n2 $udp
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
set null [new Agent/Null]
$ns attach-agent $n3 $null
$ns connect $tcp $sink
$ns connect $udp $null
$ns rtmodel-at 1.0 down $n1 $n3
$ns rtmodel-at 2.0 up $n1 $n3
$ns rtproto DV
$ns at 0.0 "$ftp start"

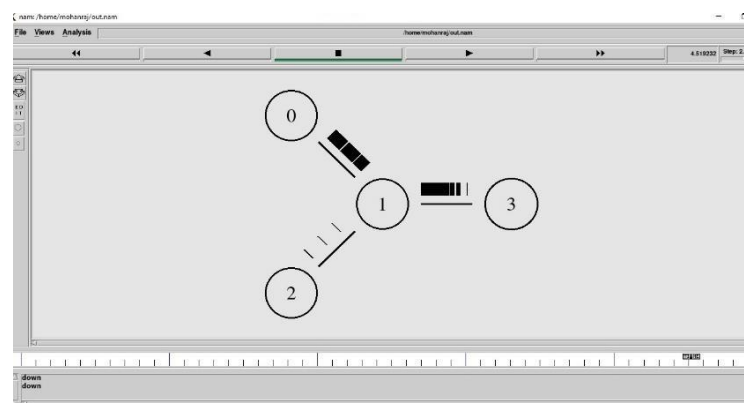
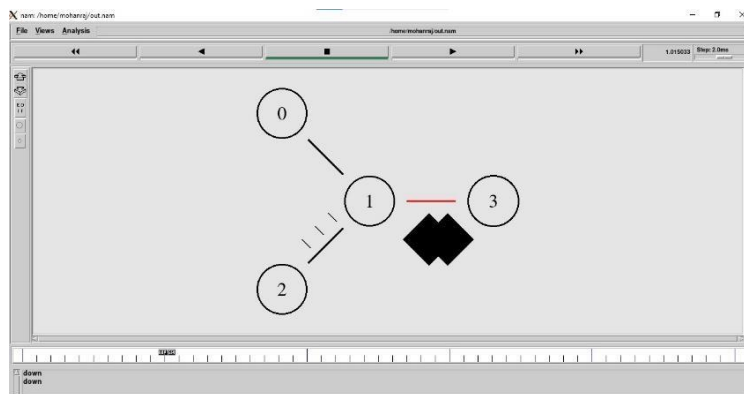
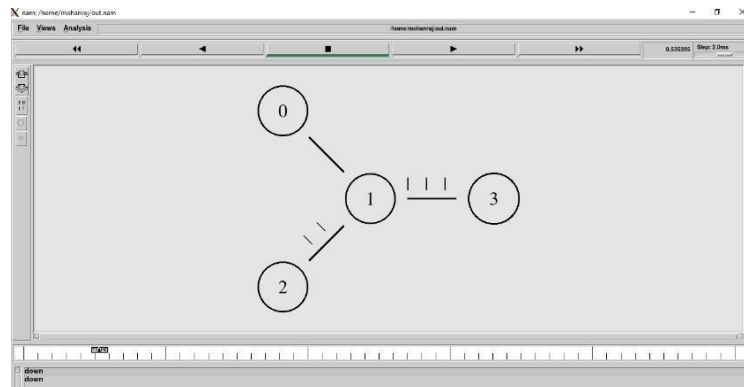
```

\$ns at 0.0 "\$cbr start"

\$ns at 5.0 "finish"

\$ns run

OUTPUT:



RESULT:

Thus, the above program to do the Distance Vector Routing Protocol was executed and output is verified.

Ex.no: 9

Date:

b) SIMULATION OF LINK STATE ROUTING PROTOCOL USING NS2

AIM:

To create link state routing protocol using NS2 simulator.

ALGORITHM:

Step1.Create a simulator object ns.

Step2.Create 6 nodes namely n0 to n5, then create the duplex link between these nodes using the object ns.

Step3.Assign node positions for each and every node.

Step4.Set the queue size for the link and monitor the queue to avoid packet loss.

Step5.Create agent and sources for each end nodes to establish the TCP connection.

Step6.Schedule events for the CBR and FTP agents based on link state routing protocol.

Step7.Create finish procedure.

Step8.Run the simulator.

PROGRAM:

```
set ns [new Simulator]
set nr [open thro.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}
for { set i 0 } { $i < 12 } { incr i 1 } {
    set n($i) [$ns node]}
```

```

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

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

set udp0 [new Agent/UDP]
$ns attach-agent $n(0) $udp0

set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0

set null0 [new Agent/Null]
$ns attach-agent $n(5) $null0
$ns connect $udp0 $null0

set udp1 [new Agent/UDP]
$ns attach-agent $n(1) $udp1

set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1

set null1 [new Agent/Null]
$ns attach-agent $n(5) $null1
$ns connect $udp1 $null1

$ns rtproto LS

$ns rtmodel-at 10.0 down $n(11) $n(5)
$ns rtmodel-at 15.0 down $n(7) $n(6)
$ns rtmodel-at 30.0 up $n(11) $n(5)

```

\$ns rtmodel-at 20.0 up \$n(7) \$n(6)

\$udp0 set fid_ 1

\$udp1 set fid_ 2

\$ns color 1 Red

\$ns color 2 Green

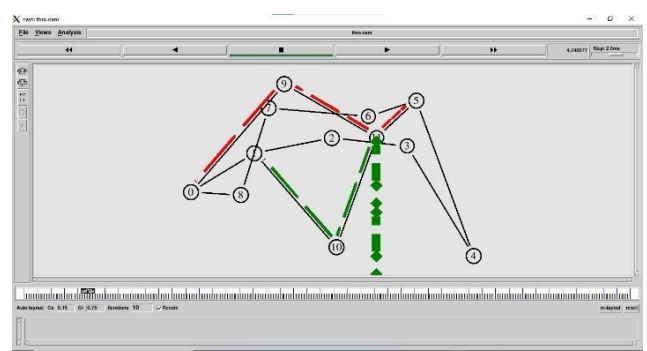
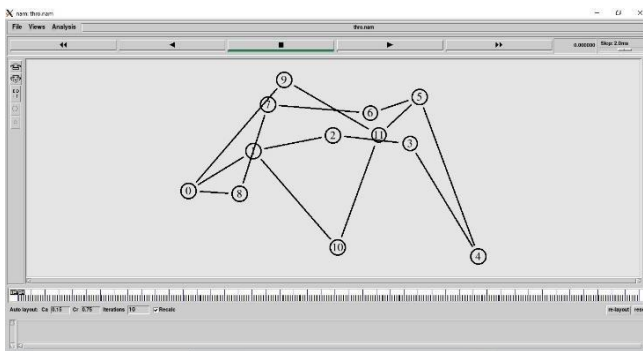
\$ns at 1.0 "\$cbr0 start"

\$ns at 2.0 "\$cbr1 start"

\$ns at 45 "finish"

\$ns run

OUTPUT:



RESULT:

Thus, the above program to do the Link state Routing Protocol was executed and output is verified.

AIM:

To study Error detecting and error correcting codes (CRC, Hamming code and Checksum).

Cyclic Redundancy Check (CRC):

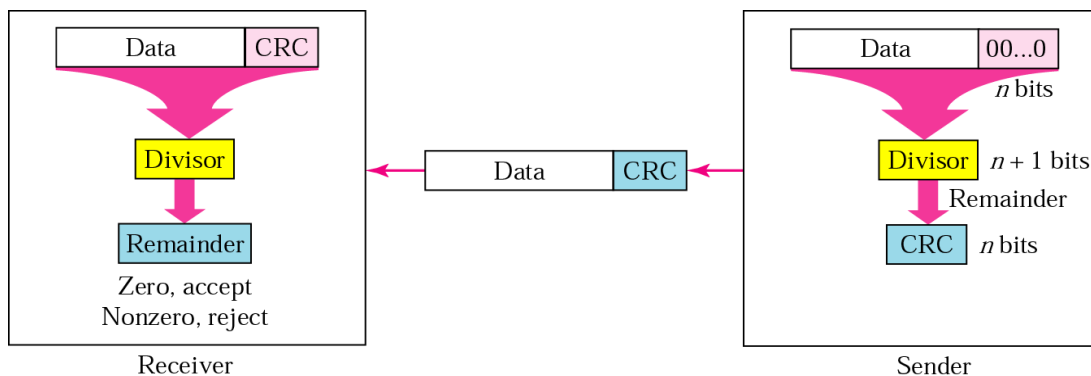
CRC is a redundancy error technique used to determine the error.

Following are the steps used in CRC for error detection:

- In CRC technique, a string of n 0s is appended to the data unit, and this n number is less than the number of bits in a predetermined number, known as division which is $n+1$ bits.
- Secondly, the newly extended data is divided by a divisor using a process is known as binary division. The remainder generated from this division is known as CRC remainder.
- Thirdly, the CRC remainder replaces the appended 0s at the end of the original data. This newly generated unit is sent to the receiver.
- The receiver receives the data followed by the CRC remainder. The receiver will treat this whole unit as a single unit, and it is divided by the same divisor that was used to find the CRC remainder.

If the resultant of this division is zero which means that it has no error, and the data is accepted.

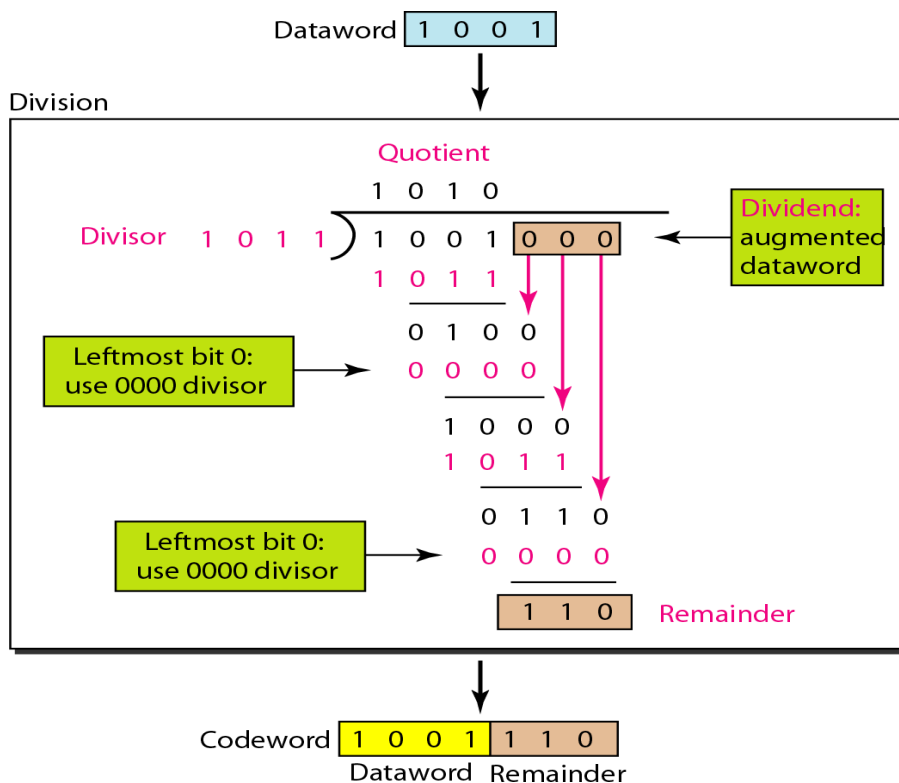
If the resultant of this division is not zero which means that the data consists of an error. Therefore, the data is discarded.



Example: Suppose the original data is 1001 and divisor is 1011.

CRC Generator :

- A CRC generator uses a modulo-2 division. Firstly, three zeroes are appended at the end of the data as the length of the divisor is 4 and we know that the length of the string 0s to be appended is always one less than the length of the divisor.
- Now, the string becomes 1001000, and the resultant string is divided by the divisor 1011.
- The remainder generated from the binary division is known as CRC remainder. The generated value of the CRC remainder is 110.
- CRC remainder replaces the appended string of 0s at the end of the data unit, and the final string would be 1001110 which is sent across the network.



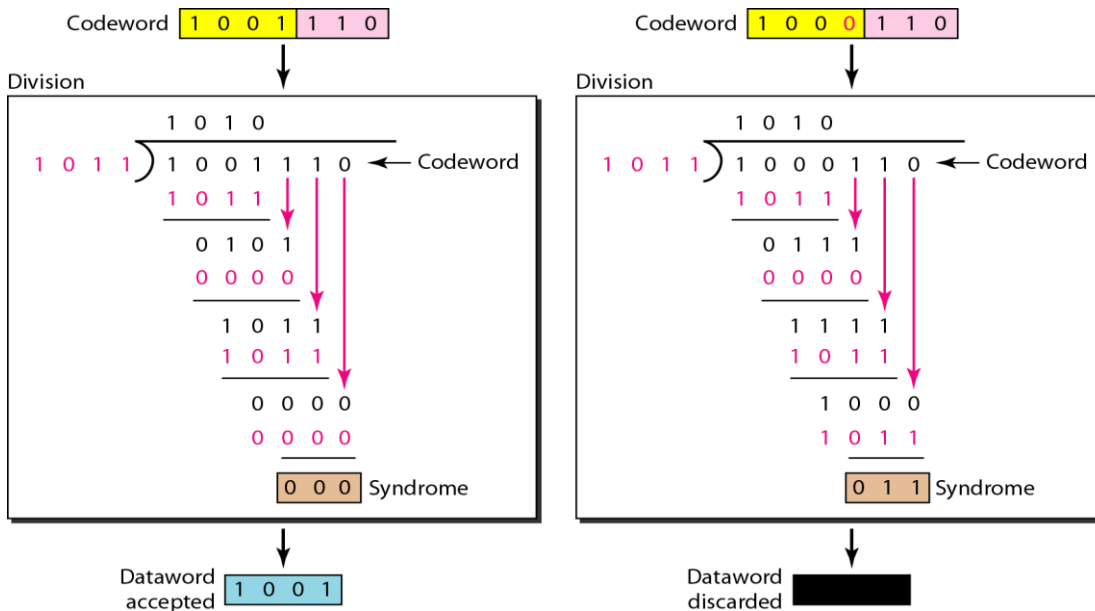
CRC Checker

Case 1:

- The functionality of the CRC checker is similar to the CRC generator.
- When the string 1001110 is received at the receiving end, then CRC checker performs the modulo-2 division.
- A string is divided by the same divisor, i.e., 1011.
- In this case, CRC checker generates the remainder of zero. Therefore, the data is accepted.

Case 2:

- The functionality of the CRC checker is similar to the CRC generator.
- When the string 1000110 is received at the receiving end, then CRC checker performs the modulo-2 division.
- A string is divided by the same divisor, i.e., 1011.
- In this case, CRC checker generates the non-zero remainder (011). Therefore, the data is discarded.



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:

```
import java.io.*;

class CRC{

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

```

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter Generator:");
String gen = br.readLine();
System.out.println("Enter Data:");
String data = br.readLine();
String code = data;
while(code.length() < (data.length() + gen.length() - 1))
code = code + "0";
code = data + div(code,gen);
System.out.println("The transmitted Code Word is: " + code);
System.out.println("Please enter the received Code Word: ");
String rec = br.readLine();
if(Integer.parseInt(div(rec,gen)) == 0)
System.out.println("The received code word contains no errors.");
else
System.out.println("The received code word contains errors.");}
static String div(String num1,String num2) {
int pointer = num2.length();
String result = num1.substring(0, pointer);
String remainder = "";
for(int i = 0; i < num2.length(); i++)
{
if(result.charAt(i) == num2.charAt(i))
remainder += "0";
else
remainder += "1";
}
while(pointer < num1.length())
{

```

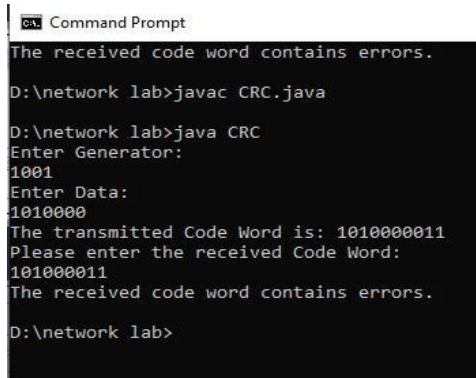
```

if(remainder.charAt(0) == '0')
{
remainder = remainder.substring(1, remainder.length());
remainder = remainder + String.valueOf(num1.charAt(pointer));
pointer++;}
result = remainder;
remainder = "";
for(int i = 0; i < num2.length(); i++)
{
if(result.charAt(i) == num2.charAt(i))
remainder += "0";
else
remainder += "1";
}
}
return remainder.substring(1,remainder.length());
}}

```

OUTPUT:

Case-1

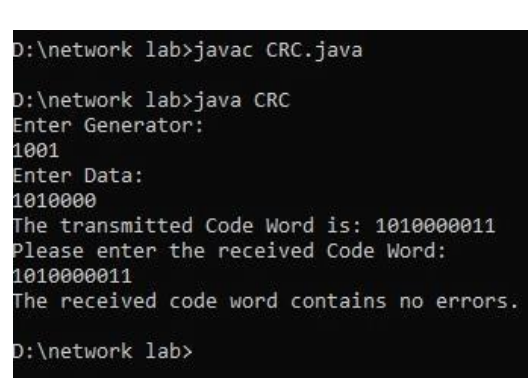


```

cs Command Prompt
The received code word contains errors.
D:\network lab>javac CRC.java
D:\network lab>java CRC
Enter Generator:
1001
Enter Data:
1010000
The transmitted Code Word is: 1010000011
Please enter the received Code Word:
101000011
The received code word contains errors.
D:\network lab>

```

Case-2



```

D:\network lab>javac CRC.java
D:\network lab>java CRC
Enter Generator:
1001
Enter Data:
1010000
The transmitted Code Word is: 1010000011
Please enter the received Code Word:
1010000011
The received code word contains no errors.
D:\network lab>

```

RESULT:

Tues the study of Error correction code CRC Cyclic Redundancy Check (CRC) was completed successfully.