

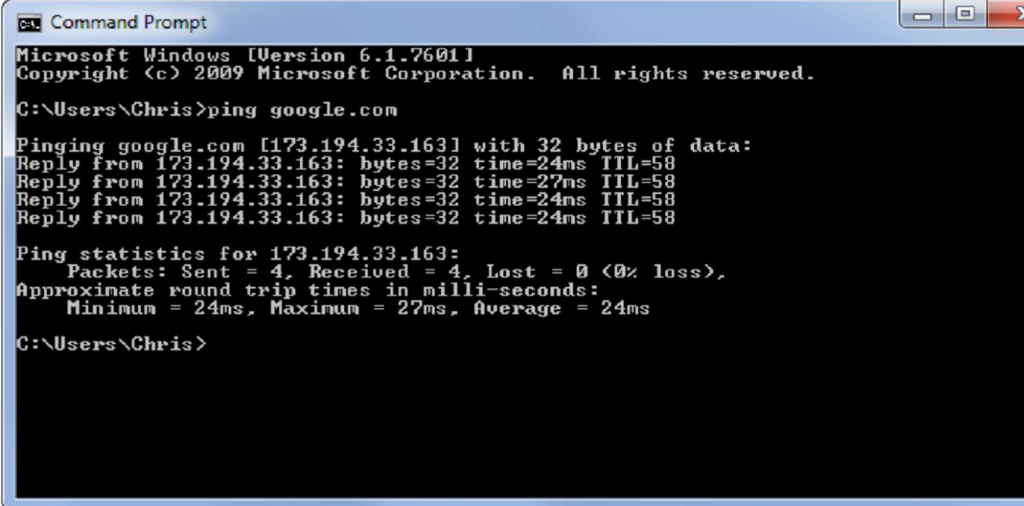
Aim:

1. Ping

RELATED: How To Troubleshoot Internet Connection Problems

The ping command sends ICMP echo request packets to a destination. For example, you could run ping **google.com** or **ping 173.194.33.174** to ping a domain name or IP address.

These packets ask the remote destination to reply. If the remote destination is configured to reply, it will respond with packets of its own. You'll be able to see how long the round-trip time is between your computer and the destination. You'll see a "request timed out" message if packet loss is occurring, and you'll see an error message if your computer can't communicate with the remote host at all. This tool can help you troubleshoot Internet connection problems, but bear in mind that many servers and devices are configured not to reply to pings.



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

C:\Users\Chris>ping google.com

Pinging google.com [173.194.33.163] with 32 bytes of data:
Reply from 173.194.33.163: bytes=32 time=24ms TTL=58
Reply from 173.194.33.163: bytes=32 time=27ms TTL=58
Reply from 173.194.33.163: bytes=32 time=24ms TTL=58
Reply from 173.194.33.163: bytes=32 time=24ms TTL=58

Ping statistics for 173.194.33.163:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 24ms, Maximum = 27ms, Average = 24ms

C:\Users\Chris>
```

2. traceroute / tracert / tracepath

The traceroute, tracert, or tracepath command is similar to ping, but provides information about the path a packet takes. traceroute sends packets to a destination, asking each Internet router along the way to reply when it passes on the packet. This will show you the path packets take when you send them between your location and a destination.

This tool can help troubleshoot connection problems. For example, if you can't communicate with a server, running traceroute may show you where the problem is occurring between your computer and the remote host.

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

C:\Users\Chris>tracert google.com

Tracing route to google.com [173.194.33.169]
over a maximum of 30 hops:

  0  1 ms  1 ms  1 ms  192.168.1.254
  1  7 ms  7 ms  9 ms  10.31.188.1
  2  11 ms 34 ms 19 ms STILLWAWBC101.bb.telus.com [75.154.217.108]
  3  11 ms 11 ms 10 ms 74.125.49.177
  4  11 ms 11 ms 10 ms 209.85.249.34
  5  11 ms 11 ms 11 ms 209.85.244.65
  6  11 ms 10 ms 11 ms sea09s18-in-f9.1e100.net [173.194.33.169]

Trace complete.

C:\Users\Chris>
```

3.ipconfig / ifconfig

The ipconfig command is used on Windows, while the ifconfig command is used on Linux, Mac OS X,

and other Unix-like operating systems. These commands allow you to configure your network interfaces and view information about them.

For example, you can use the ipconfig /all command on Windows to view all your configured network interfaces, their IP addresses, DNS servers, and other information. Or, you can use the ipconfig /flushdns command to flush your DNS cache, forcing Windows to get new addresses from its DNS servers every time you contact a new hostname. Other commands can force your computer to release its IP address and get a new one from its DHCP server. This utility can quickly display your computers IP address or help you troubleshoot problems.

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

C:\Users\Chris>ipconfig /all

Windows IP Configuration

    Host Name . . . . . : Laptop
    Primary Dns Suffix . . . . . :
    Node Type . . . . . : Hybrid
    IP Routing Enabled. . . . . : No
    WINS Proxy Enabled. . . . . : No
    DNS Suffix Search List. . . . . : telus

Wireless LAN adapter Wireless Network Connection:

    Connection-specific DNS Suffix . : telus
    Description . . . . . : Intel(R) Centrino(R) Wireless-N 2230
    Physical Address. . . . . : 68-5D-43-66-0B-0C
    DHCP Enabled. . . . . : Yes
    Autoconfiguration Enabled . . . . : Yes
    Link-local IPv6 Address . . . . . : fe80::799d:c5a7:c72:b925%11(Preferred)
    IPv4 Address. . . . . : 192.168.1.66(Preferred)
    Subnet Mask . . . . . : 255.255.255.0
    Lease Obtained. . . . . : June-01-14 12:41:11 PM
    Lease Expires . . . . . : June-03-14 12:41:11 AM
    Default Gateway . . . . . : 192.168.1.254
    DHCP Server . . . . . : 192.168.1.254
    DNS Servers . . . . . : 8.8.8.8
                           8.8.4.4
    NetBIOS over Tcpip. . . . . : Enabled
```

4. nslookup

The nslookup command will look up the IP addresses associated with a domain name. nslookup also allows you to perform a reverse lookup to find the domain name associated with an IP address. For example, **nslookup 208.43.115.82** will show you that this IP address is associated with howtogeek.com.

```
chris@ubuntu1404vbox: ~
chris@ubuntu1404vbox:~$ nslookup howtogeek.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   howtogeek.com
Address: 208.43.115.82

chris@ubuntu1404vbox:~$ nslookup 208.43.115.82
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
82.115.43.208.in-addr.arpa      name = howtogeek.com.

Authoritative answers can be found from:

chris@ubuntu1404vbox:~$
```

Netstat is a useful tool for checking network and Internet connections.
Some useful applications for the average PC user are considered, including checking for malware connections. Syntax and switches

The command syntax is netstat [-a] [-b] [-e] [-f] [-n] [-o] [-p proto] [-r] [s] [-t] [-v] [interval] A brief description of the switches is given in Table I below. Some switches are only in certain

Windows versions, as noted in the table..Note that switches for Netstat use the dash symbol "-" rather than the slash "/".

Switch	Description
-a	Displays all connections and listening ports
	Displays the executable involved in creating each connection or listening port.
-b	(Added in XP SP2.)
-e	Displays Ethernet statistics
	Displays Fully Qualified Domain Names for foreign addresses. (In Windows
-f	Vista/7 only)
n.	Displays addresses and port numbers in numerical form
.	Displays the owning process ID associated with each connection
	Shows connections for the protocol specified by proto; proto may be any of:
p. proto	TCP, UDP, TCPv6, or UDPv6.
r.	Displays the routing table
	Displays per-protocol statistics
.	Displays the current connection offload state, (Windows Vista/7)
	When used in conjunction with -b, will display sequence of
components -v	involved in creating the connection or listening port for all executables. (Windows XP SP2, SP3)
	An integer used to display results multiple times with specified
number of [interval]	seconds between displays. Continues until stopped by command ctrl+c.
	Default setting is to display once,

Checking TCP/IP connections

TCP and UDP connections and their IP and port addresses can be seen by entering a command combining two switches: netstat -an An example of the output that is obtained is shown in Figure 1.

The information that is displayed includes the protocol, the local address, the remote (foreign) address, and the connection state. Note that the various IP addresses include port information as well. An explanation of the different connection states is given in Table II>

State	Description
CLOSED	Indicates that the server has received an ACK signal from the client and the connection is closed
CLOSE_WAIT	Indicates that the server has received the first FIN signal from the client and the connection is in the process of being closed Indicates that the server received the SYN signal from the client and the
ESTABLISHED	session is established
FIN_WAIT_1	Indicates that the connection is still active but not currently being used
FIN_WAIT_2	Indicates that the client just received acknowledgment of the first FIN signal from the server
LAST_ACK	Indicates that the server is in the process of sending its own FIN signal
LISTENING	Indicates that the server is ready to accept a connection
SYN_RECEIVED	Indicates that the server just received a SYN signal from the client
SYN_SEND	Indicates that this particular connection is open and active
TIME_WAIT	Indicates that the client recognizes the connection as still active but not currently being used

Checking for malware by looking at which programs initiate connections

```

C:\WINDOWS\system32\cmd.exe - netstat
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\roman.rafacz>netstat
Active Connections
Proto Local Address           Foreign Address         State
TCP    NRKJMW-dxp14080:1828    nycnbx44.na.corp.ipgnetwork.com:5012 ESTABLISHE
D    TCP    NRKJMW-dxp14080:1830    nycnbx44.na.corp.ipgnetwork.com:5012 ESTABLISHE
D    TCP    NRKJMW-dxp14080:1831    nycnbx44.na.corp.ipgnetwork.com:5012 ESTABLISHE
D    TCP    NRKJMW-dxp14080:1834    nycgdc16.na.corp.ipgnetwork.com:5001 ESTABLISHE
TCP    NRKJMW-dxp14080:1839    b-sntp.jackmorton.com:1533 ESTABLISHED
TCP    NRKJMW-dxp14080:1843    174.36.30.27-static.reverse.softlayer.com:http
ESTABLISHED
TCP    NRKJMW-dxp14080:1961    nrkfls04.na.corp.ipgnetwork.com:microsoft-ds ES
TABLISHED
TCP    NRKJMW-dxp14080:3385    nycnfp01.na.corp.ipgnetwork.com:5012 ESTABLISHE
D    TCP    NRKJMW-dxp14080:3394    qw-in-f17.google.com:http ESTABLISHED
TCP    NRKJMW-dxp14080:3443    qw-in-f103.google.com:http ESTABLISHED
TCP    NRKJMW-dxp14080:3450    8.21.194.129:http      ESTABLISHED
TCP    NRKJMW-dxp14080:3471    8.21.194.129:http      ESTABLISHED
TCP    NRKJMW-dxp14080:3472    8.21.194.129:http      ESTABLISHED
TCP    NRKJMW-dxp14080:3484    wiki.answers.com:http   ESTABLISHED
TCP    NRKJMW-dxp14080:3488    qw-in-f155.google.com:http ESTABLISHED
TCP    NRKJMW-dxp14080:3489    qw-in-f155.google.com:http ESTABLISHED

```

6. TCPDUMP

tcpdump command is also called as packet analyzer. tcpdump command will work on most flavors of unix operating system. tcpdump allows us to save the packets that are captured, so that we can use it for future analysis. The saved file can be viewed by the same tcpdump command. We can also use open source software like wireshark to read the tcpdump pcap files.

In this tcpdump tutorial, let us discuss some practical examples on how to use the tcpdump command.

1. Capture packets from a particular ethernet interface using tcpdump -i When you execute tcpdump command without any option, it will capture all the packets flowing through all the interfaces. -i option with tcpdump command, allows you to filter on a particular ethernet interface. `$ tcpdump i eth1`

```
14:59:26.608728 IP xx.domain.netbcp.net.52497 > valh4.lell.net.ssh: . ack 540 win 16554
```

```
14:59:26.610602 IP resolver.lell.net.domain > valh4.lell.net.24151: 4278 1/0/0 (73)
```

```
14:59:26.611262 IP valh4.lell.net.38527 > resolver.lell.net.domain: 26364+
```

```
PTR? 244.207.104.10.in-addr.arpa. (45)
```

In this example, tcpdump captured all the packets flows in the interface eth1 and displays in the standard output.

Result:

Aim:**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()));

        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:

When you execute the client code, the following output appears on client side –

```

Client is running.
Reading image from disk.
Sending image to server.
Image sent to server.

```

When you execute the server code, the following output appears on server side –

```

Server Waiting for image
Client connected.
Image Size: 29KB

```


Result:

EXP No: 3a

Date :

APPLICATIONS USING TCP SOCKETS

A. ECHO CLIENT AND ECHO SERVER

Aim :

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);
            w.println("Welcome to the Java EchoServer.

            Type 'bye' to
            close.");
            String line;
            do
            {
                line = .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:

CLIENT

Enter the IP address 127.0.0.1

CONNECTION ESTABLISHED

Enter the data SRM Client received SRM

SERVER

CONNECTION ACCEPTED

Server received SRM

Result:

Aim:**Procedure :****Establish a Socket Connection**

To connect to another machine we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port. The java.net.Socket class represents a Socket. To open a socket:

```
Socket socket = new Socket("127.0.0.1", 5000)
```

- The first argument – **IP address of Server**. (127.0.0.1 is the IP address of localhost, where code will run on the single stand-alone machine).
- The second argument – **TCP Port**. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

Client Programming :

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

public class Client {
    // initialize socket and input output streams
    private Socket socket = null;    private
    DataInputStream input = null;    private
    DataOutputStream out = null;

    // constructor to put ip address and port    public
    Client(String address, int port)
    {
        // establish a connection
        try {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input = new DataInputStream(System.in);
```

```

        // sends output to the socket
out = new DataOutputStream(
    socket.getOutputStream());
    }
    catch (UnknownHostException u) {
System.out.println(u);
    return;
    }
    catch (IOException i) {
System.out.println(i);
    return;
    }

    // string to read message from input
String line = "";

    // keep reading until "Over" is input
while (!line.equals("Over")) {
    try {
        line = input.readLine();
out.writeUTF(line);
    }
    catch (IOException i) {
System.out.println(i);
    }
    }

    // close the connection
    try {
        input.close();
out.close();
        socket.close();
    }
    catch (IOException i) {
System.out.println(i);
    }
    }

    public static void main(String args[])
    {
        Client client = new Client("127.0.0.1", 5000);
    }
}

```

Server

Program:

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

public class Server
{
    //initialize socket and input stream
    private Socket    socket = null; private
    ServerSocket  server  = null;
    private DataInputStream in    = null;

    // constructor with port
    public Server(int port)
    {
        // starts server and waits for a connection    try
        {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");
            socket = server.accept();
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));

            String line = "";
            // reads message from client until "Over" is sent    while
            (!line.equals("Over"))
            {
                try
                {
                    line = in.readUTF();
                    System.out.println(line);
                }
                catch(IOException i)
                {
                    System.out.println(i);
                }
            }
            System.out.println("Closing connection");

            // close connection
            socket.close();    in.close();
        }
    }
}
```



```
    }  
    catch(IOException i)  
    {  
        System.out.println(i);  
    }  
}  
public static void main(String args[])  
{  
    Server server = new Server(5000);  
}  
}
```

Output :

\$ java Server

Server started

Waiting for a client ...

\$ java Client

It will show – Connected and the server accepts the client and shows, Client accepted
Server :

Hello

I made my first socket connection

Over

Client :

Hello

I made my first socket connection

Over

Closing connection

Result:

Exp.no:4
Date:

SIMULATION OF DNS USING UDP SOCKETS

Aim :

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

UDP DNS Server

```
Import java.io.*;
import java.net.*;
public class dnsserver
{ private static int indexOf(String[] array, String str)
    {
        str = str.trim();
        for (int i=0; i < array.length; i++)
        {
            if (array[i].equals(str)) return i;
        }
        return -1;
    }
}
```

```

        public static void main(String arg[])throws IOException
        {
            String[]
hosts = {"zoho.com", "gmail.com","google.com", "facebook.com"};    String[]
ip = {"172.28.251.59", "172.217.11.5","172.217.11.14", "31.13.71.36"};
System.out.println("Press Ctrl + C to Quit");

            while (true)
            {

                DatagramSocket serversocket=new DatagramSocket(1362);

byte[] senddata = new byte[1021];
byte[] receivedata = new byte[1021];
                DatagramPacket recvpack = new DatagramPacket(receivedata,
                                                                receivedata.length);

                serversocket.receive(recvpack);
                String sen = new String(recvpack.getData());
                InetAddress ipaddress = recvpack.getAddress();
                int port = recvpack.getPort();
                String capsent;

                System.out.println("Request for host " + sen);

                if(indexOf (hosts, sen) != -1)

                    capsent = ip[indexOf (hosts, sen)];
                else

                    capsent = "Host Not Found";
                    senddata = capsent.getBytes();

                    DatagramPacketpack=newDatagramPacket(senddata,
senddata.length,ipaddress,port);
serversocket.send(pack);

                    serversocket.close();

                }

            }

        }

```

UDP DNS Client

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

public class
dnsclient
{
    public static void main(String args[])throws IOException
    {
        BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
        DatagramSocket clientsocket = new DatagramSocket();   InetAddress
        ipaddress;
        if (args.length == 0) ipaddress = InetAddress.getLocalHost();
        else
            ipaddress = InetAddress.getByName(args[0]);
        byte[] senddata = new byte[1024];
        byte[] receivedata = new byte[1024];
        int portaddr = 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
        =newDatagramPacket(receivedata,receivedata.le ngth);
        clientsocket.receive(recvpack);
        String modified = new String(recvpack.getData());
        System.out.println("IP Address: " + modified);    clientsocket.close();
    }
}
```

OUTPUT

Server

E:\nwlab>java dnsserver

Press Ctrl + C to Quit

Request for host google.com

Request for host flipkart.com

Client

E:\nwlab>java dnsclient

Enter the hostname : google.com IP

Address:

172.217.11.14

E:\nwlab>java dnsclient

Enter the hostname : flipkart.com

IP Address: Host Not Found

Result:

EX.NO:5
DATE:

Use a tool like Wireshark to capture packets and examine the packets.

Aim :

Getting Wireshark :

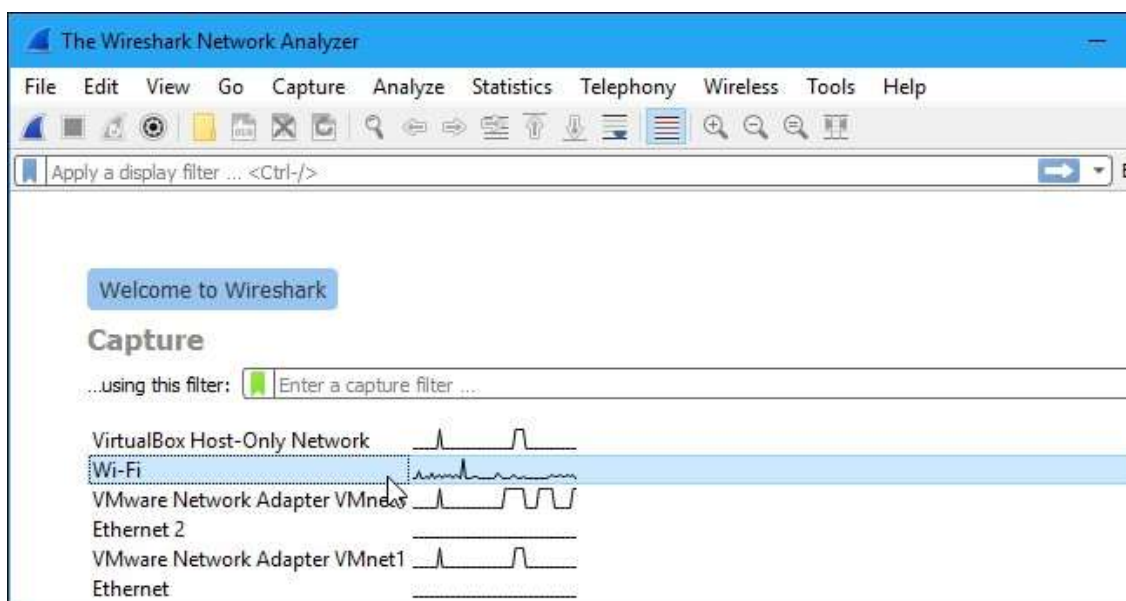
You can download Wireshark for Windows or macOS from its [official website](#). If you're using Linux or another UNIX-like system, you'll probably find Wireshark in its package repositories. For example, if you're using Ubuntu, you'll find Wireshark in the Ubuntu Software Center.

Just a quick warning: Many organizations don't allow Wireshark and similar tools on their networks. Don't use this tool at work unless you have permission.

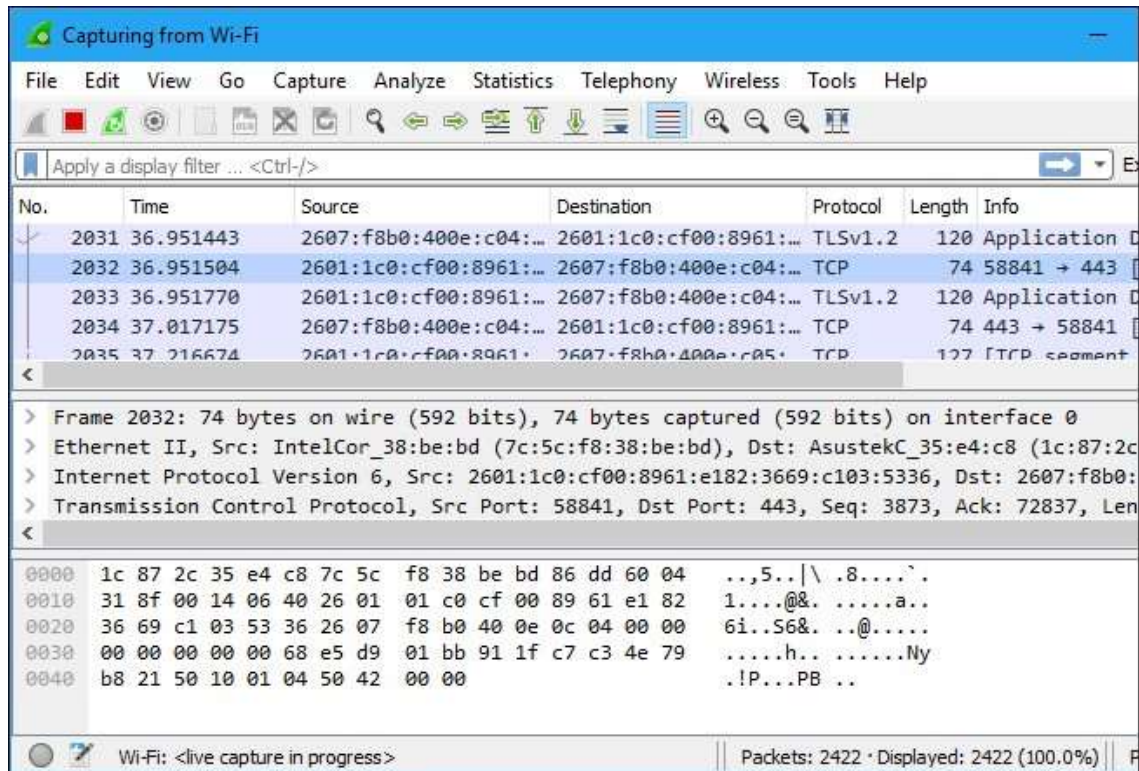
Capturing Packets :

After downloading and installing Wireshark, you can launch it and double-click the name of a network interface under Capture to start capturing packets on that interface. For example, if you want to capture traffic on your wireless network, click your wireless interface. You can configure advanced features by clicking Capture > Options, but this isn't necessary for now.

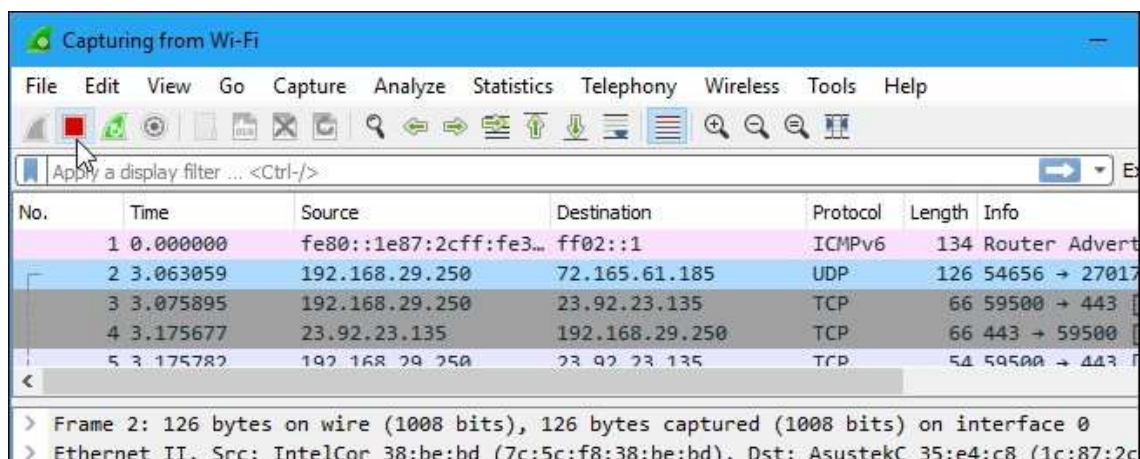
As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system.



If you have promiscuous mode enabled—it’s enabled by default—you’ll also see all the other packets on the network instead of only packets addressed to your network adapter. To check if promiscuous mode is enabled, click Capture > Options and verify the “Enable promiscuous mode on all interfaces” checkbox is activated at the bottom of this window.



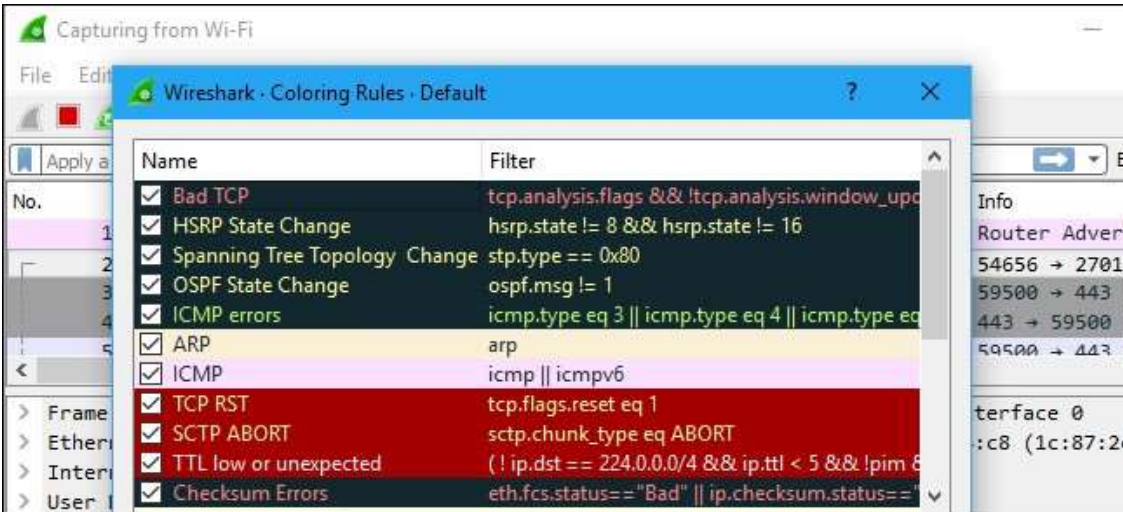
Click the red “Stop” button near the top left corner of the window when you want to stop capturing traffic.



Color Coding

By default, light purple is TCP traffic, light blue is UDP traffic, and black identifies packets with errors—for example, they could have been delivered out of order.

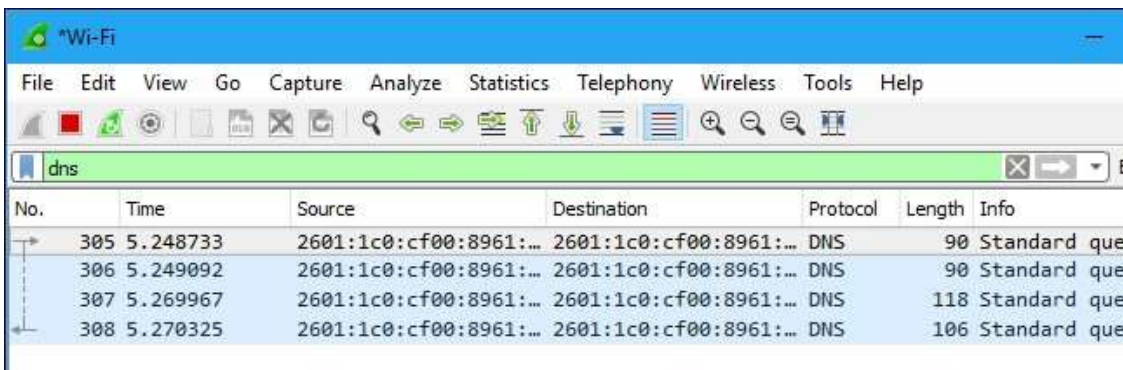
To view exactly what the color codes mean, click View > Coloring Rules. You can also customize and modify the coloring rules from here, if you like.



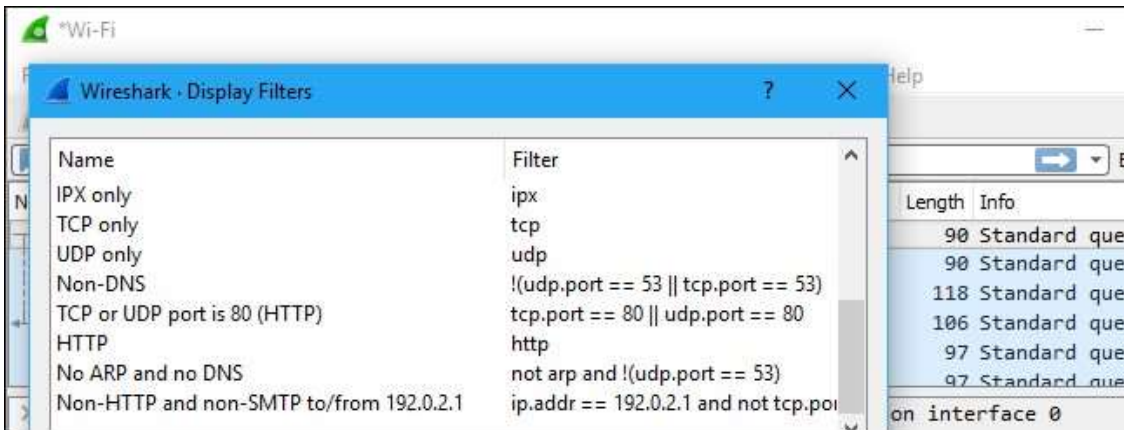
Filtering Packets

If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through. That's where Wireshark's filters come in.

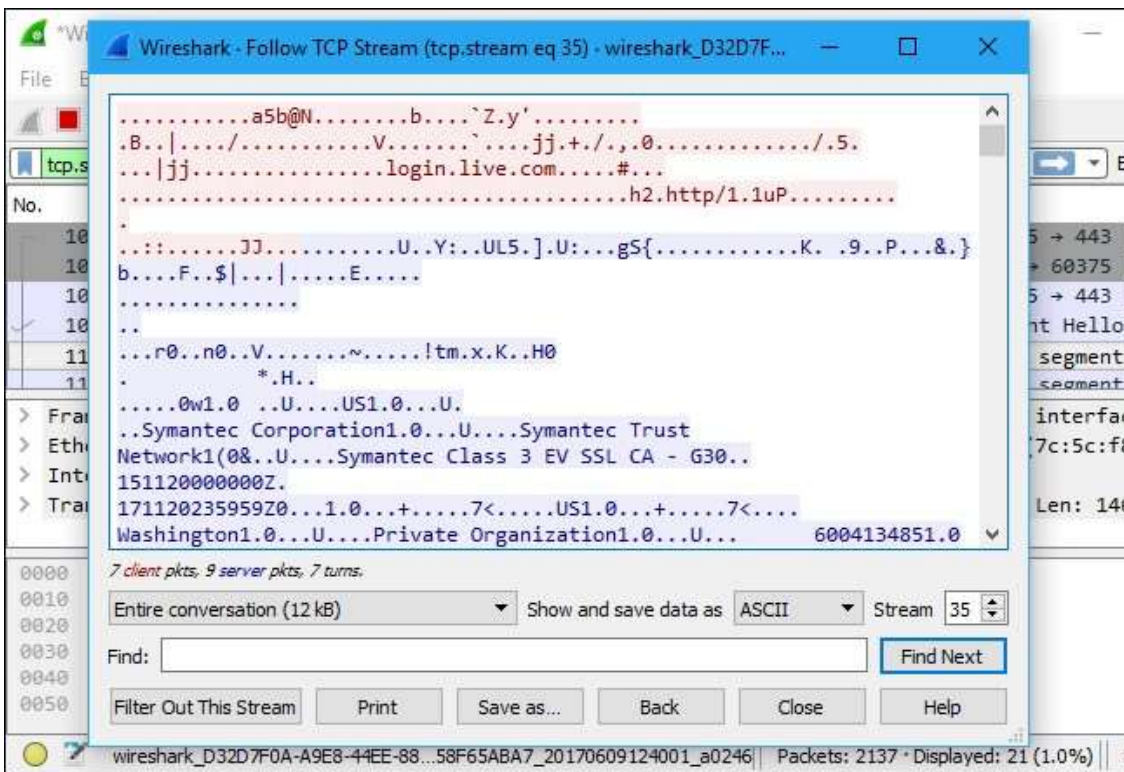
The most basic way to apply a filter is by typing it into the filter box at the top of the window and clicking Apply (or pressing Enter). For example, type "dns" and you'll see only DNS packets. When you start typing, Wireshark will help you autocomplete your filter.



For more information on Wireshark's display filtering language, read the Building display filter expressions page in the official Wireshark documentation.



Another interesting thing you can do is right-click a packet and select Follow > TCP Stream.



Inspecting Packets

Click a packet to select it and you can dig down to view its details.

You can also create filters from here — just right-click one of the details and use the Apply as Filter submenu to create a filter based on it.

Wi-Fi

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

tcp.stream eq 35

No.	Time	Source	Destination	Protocol	Length	Info
1054	2.798483	192.168.29.250	131.253.61.66	TCP	66	60375 → 443
1078	2.891263	131.253.61.66	192.168.29.250	TCP	58	443 → 60375
1079	2.891359	192.168.29.250	131.253.61.66	TCP	54	60375 → 443
1080	2.891527	192.168.29.250	131.253.61.66	TLSv1.2	288	Client Hello

Frame 1054: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface 0
 Interface id: 0 (\Device\NPF_{D32D7F0A-A9E8-44EE-88DC-DFD58F65ABA7})
 Encapsulation type: Ethernet (1)
 Arrival Time: Jun 9, 2017 12:40:04.140141000 Pacific Daylight Time
 [Time shift for this packet: 0.000000000 seconds]
 Epoch Time: 1497037204.140141000 seconds

```

0000  1c 87 2c 35 e4 c8 7c 5c f8 38 be bd 08 00 45 00  ..,5..|\.8....E.
0010  00 34 0b 5d 40 00 80 06 4f 85 c0 a8 1d fa 83 fd  .4.]@... O.....
0020  3d 42 eb d7 01 bb 22 52 7b 69 00 00 00 00 80 02  =B...."R {i.....
0030  fa f0 48 ef 00 00 02 04 05 b4 01 03 03 08 01 01  ..H.....
0040  04 02  ..
  
```

Encapsulation type (frame.encap_type) | Packets: 8136 · Displayed: 21 (0.3%)

Result:

EX.NO:6 (a)
DATE:

WRITE A CODE SIMULATING ARP PROTOCOL.

AIM:

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

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:

EXP.NO: 6(b)

DATE:

**PROGRAM FOR REVERSE ADDRESS RESOLUTION PROTOCOL
(RARP) USING UDP**

Aim

ALGORITHM:

Client

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

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=newDatagramPacket(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 Serverarp12

{

```



```

        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();

                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=newDatagramPacket(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:**AIM:**

EX.NO:7
DATE:

STUDY OF NETWORK (NS) AND SIMULATION OF CONGESTION CONTROL ALGORITHM USING NS

INTRODUCTION:

A device, computer program or system used during software verification, which behaves or operates like a given system when provided with a set of controlled inputs.

NEED OF SIMULATORS:

- Customers with the simulator spend less time debugging simple program errors
- The simulator makes it easy to write and test code
- It is easier for our support engineers to explain complex problems if you have a simulator.
- The simulator requires no setup time. An emulator may require configuration and a target board before you can debug.

EXAMPLES OF SIMULATORS:

Some of the famous network simulators available are listed below-

- NS-2
- OPNET
- PADN
- S
- GTNETS
- GLOMOSIM
- M5-SIMULATOR
- DARMOUTH SSF

OVERVIEW OF NS

The overall simulation procedure in the NS is shown in Fig.1. First of all, there is a language named OTcl, and this is an object oriented version language which was converted from the conventional Tcl (Tool Command Language). As shown in Fig.2, an OTcl Script representing a given network configuration is written. NS is simply composed of OTcl Script and OTcl Interpreter. OTcl Interpreter can translate the code related to NS using NS Simulation Library.

NS simulation results can be observed through graphs for analysis or animations with NAM (Network Animator). NAM is a network animator that is developed for education purposes. It can display the detailed procedure of the simulation. NAM is not used in this experiment.

Very simply speaking, NS is the Object-oriented Tcl (OTcl) script interpreter that has a simulation event scheduler and network component object libraries, and network set-up module libraries. Users write an OTcl script using OTcl.

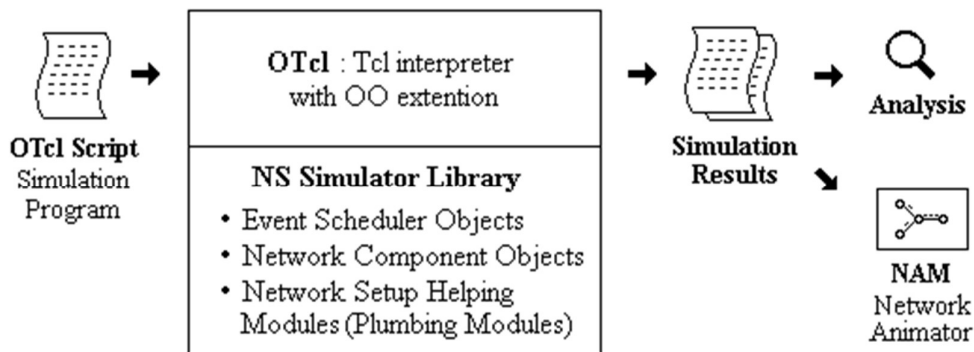


Fig.1 Simplified User's View of NS

(1) Duality of OTcl and C++

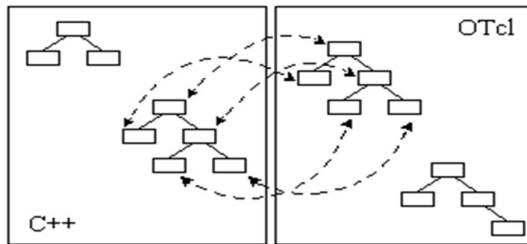


Fig.2 C++ and OTcl: The Duality

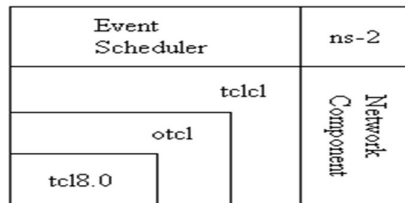


Figure 3. Architectural View of NS

As shown in Fig.3, NS has a one-to-one mapping structure between C++ and OTcl. For example, if there exists a class which implements WFQ (Weighted Fair Queuing) in C++, there exists a similar Class in OTcl. This structure is introduced for improving the simulation speed and providing the convenience. And the core components in network simulation, such as Node, Link, and Queue are written using C++ Classes. Since OTcl performs at a slow processing speed, the components for Network Simulation are written using C++ Classes, and the process that comprises Network by connecting these components with each other is written in OTcl Script. Moreover, NS is implemented using a dual structure as shown in the above figure for accessing components written in C++ Classes through OTcl Script.

Consequently, if a user makes a new Network Component, he/she should write C++ Classes and OTcl Classes and the part to connect each other. Since it is somewhat difficult to write C++ Classes and to perform Simulation, it is omitted in this experiment.

(2) NS DIRECTORY STRUCTURE

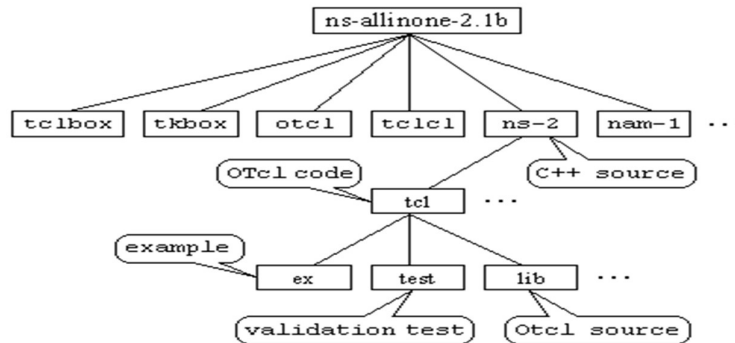


Fig.4 NS Directory Structure

Among the sub-directories of ns-allinone-2.1b shown in Fig.4, ns-2 has all of the simulator implementations (either in C++ or in OTcl), validation test OTcl scripts and example OTcl scripts. Within this directory, all OTcl codes and test/example scripts are located under the sub-directory called tcl, and most of C++ code.

LEARNING OTCL

The first step to utilize NS is to understand the OTcl language. Fig.5 shows a program example written in Tcl language. In Tcl, the keyword „proc“ is used to define a procedure, followed by a procedure name and arguments in the first brace and definition in the second brace. At this moment, note that you must use the brace, differently from C++. Note that you’ll encounter an error if you move the last brace “{“ of “proc test {} {“ to the next line.

If you run “set a 43”, NS checks whether or not the variable “a” is already declared. If the variable name has already been declared, NS inserts 43 into

```
# Writing a procedure called "test"
proc test {} {
    set a 43
    set b 27
    set c [expr $a + $b]
    set d [expr [expr $a - $b] * $c]
    for {set k 0} {$k < 10} {incr k} {
        if {$k < 5} {
            puts "k < 5, pow = [expr pow($d, $k)]"
        } else {
            puts "k >= 5, mod = [expr $d % $k]"
        }
    }
}

# Calling the "test" procedure created above
test
```

Fig.5 A Sample Tcl Script

“a”. Otherwise, a new one is declared and NS substitutes 43 into it. In other words, declaration and assignment are combined together. “set a” means that the instance “a” is declared only in case that “a” doesn’t exist. In “set c [expr \$a + \$b]”, one thing to note is that to get the value assigned to a variable, \$ is used with the variable name, and another

thing is that “expr” is used to do mathematical computations. Therefore, $43+27=70$ is inserted into c. When we use “for” command, we initialize k with “{set k 0}”, and give a condition with “{ $k < 10$ }”, and update for Index variable with “{incr k}”. Finally, the keyword „puts” is to print out the following string within double quotation marks like “printf()” in C.

OTcl

OTcl is the language which modifies the conventional Tcl to an objectoriented version. Fig.6 shows an example of an OTcl script that defines two object classes, "mom" and "kid", where "kid" is the child class of "mom", and a member function called "greet" for each class. After defining the classes, each object instance is declared, the "age" variable of each instance is set to 45 (for mom) and 15 (for kid), and the "greet" member function of each object instance is called.

```
# add a member function call "greet"
Class mom
mom instproc greet {} {
    $self instvar age_
    puts "$age_ year old mom say:
    How are you doing?"
}

# Create a child class of "mom" called "kid"
# and override the member function "greet"
Class kid -superclass mom
kid instproc greet {} {
    $self instvar age_
    puts "$age_ year old kid say:
    What's up, dude?"
}

# Create a mom and a kid object, set each age
set a [new mom]
$a set age_ 45
set b [new kid]
$b set age_ 15

# Calling member function "greet" of each object
$a greet
$b greet
```

Fig.6 A Sample OTcl Script

NS CLASS HIERARCHY

If you understood OTcl in some extent, you need to look into the Class Hierarchy of NS. NS has the Class Hierarchy which is implemented using both C++ and OTcl, as shown in Fig.7.

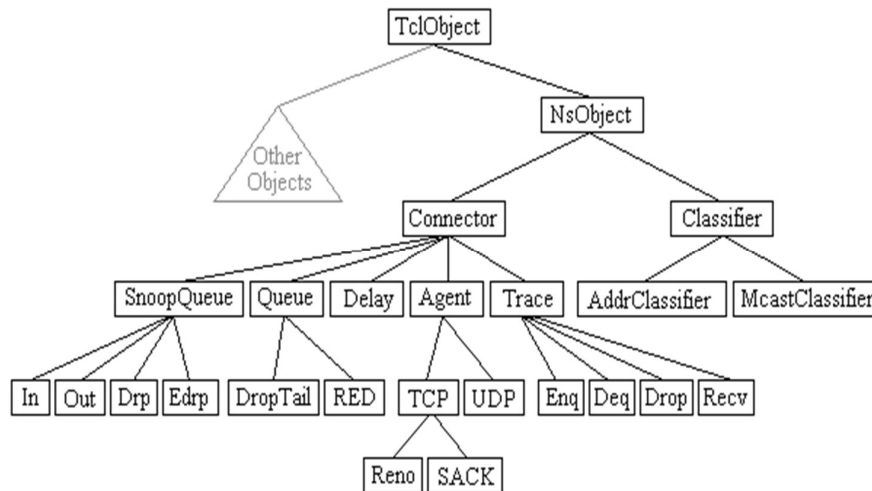


Fig.7 NS Class Hierarchy

The root of the hierarchy is the Tcl Object class that is the superclass of all OTcl libraries.

HARDWARE/SOFTWARE REQUIREMENTS

To build ns you need a computer and a C++ compiler. The all-in-one package requires about 320MB of disk space to build. Building ns from pieces can save some disk space. There are two ways to build ns: from all the pieces or all at once. If you just want to try it out quickly, you might try all at once. If you want to do C-level development, or save download time or disk space, or have trouble with all-in-one you should build it from the pieces

Ns-allinone is a package, which contains enquired components and some optional components used in running ns. The package contains an "install" script to automatically configure, compile and install these components. After downloading, run the install script. If you haven't installed ns before and want to quickly try ns out, ns-allinone may be easier than getting all the pieces by hand.

Result:

AIM:

EX.NO:8(a)

DATE:

STUDY OF UDP PERFORMANCE

INTRODUCTION:

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. UDP is sometimes called the Universal Datagram Protocol. 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 guaranteeing 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 that answer 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.

APPLICATIONS:

Numerous key Internet applications use UDP, including: the Domain Name System (DNS), where queries must be fast and only consist of a single request followed by a single reply packet, the Simple Network Management Protocol (SNMP), the Routing Information Protocol (RIP)[1] and the Dynamic Host Configuration Protocol (DHCP).

Voice and video traffic is generally transmitted using UDP. Real-time video and audio streaming protocols are designed to handle occasional lost packets, so only slight degradation in quality occurs, rather than large delays if lost packets were retransmitted. Because both TCP and UDP run over the same network, many businesses are finding that a recent increase in UDP traffic from these real-time applications is hindering the performance of applications using TCP, such as point of sale, accounting, and database systems. When TCP detects packet loss, it will throttle back its data rate usage. Since both real-time and business applications are important to businesses, developing quality of service solutions is seen as crucial by some.

IMPLEMENTATION

ALGORITHM

Step 1: Define different color for data flows for NAM

Step 2: Open the NAM trace file

Step 3: Define a finished procedure

Step 4: Define a desired Number of nodes

Step 5: Create a links between the nodes

Step 6: Specifies the position of the node for NAM

Step 7: Set the queue size between desired nodes

Step 8: Setup a udpConnection

Step 9: Setup a FTP over udp Connection

Step 10: Define a procedure for plotting window size

Step 11: Specify the end of simulation

PROGRAM

FOR UDP

```
set ns [new Simulator]

$ns color 1 Blue
$ns color 2 Red
set tracefile1
[openlanudp.tr
w] set winfile
[open
WinFilew]

$ns trace-all $tracefile1 set
namfile [open lanudp.nam w]
$ns namtrace-all $namfile

Set n0 [$ns node]
Set n1[$ns node]
Set n2[$ns node]
Set n3[$ns node]
Set n4[$ns node]
```

Set n5[\$ns node]

\$ns duplex-link \$n0 \$n2 2Mb 10ms DropTail

\$ns duplex-link \$n1 \$n2 2Mb 10ms DropTail

\$ns simplex-link \$n2 \$n3 0.3Mb
100ms DropTail \$ns simplex-
link \$n3 \$n2 0.3Mb
100ms DropTail

set lan [\$ns newLan "\$n4 \$n5 \$n3" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd
Channel]

\$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

\$ns duplex-link-op \$n3 \$n2 orient left

\$ns queue-limit \$n2 \$n3 20 set cbr

[new Application/Traffic/CBR]

\$cbr attach-agent \$udp

\$cbr set packetSize_ 1000

\$cbr set rate_ 0.01Mb

\$cbr set random_ false

\$ns at 0.1 "\$cbr start" \$ns at 124.5

"\$cbr stop" proc plotWindow

{tcpSource file} { global ns set

time 0.1 set now

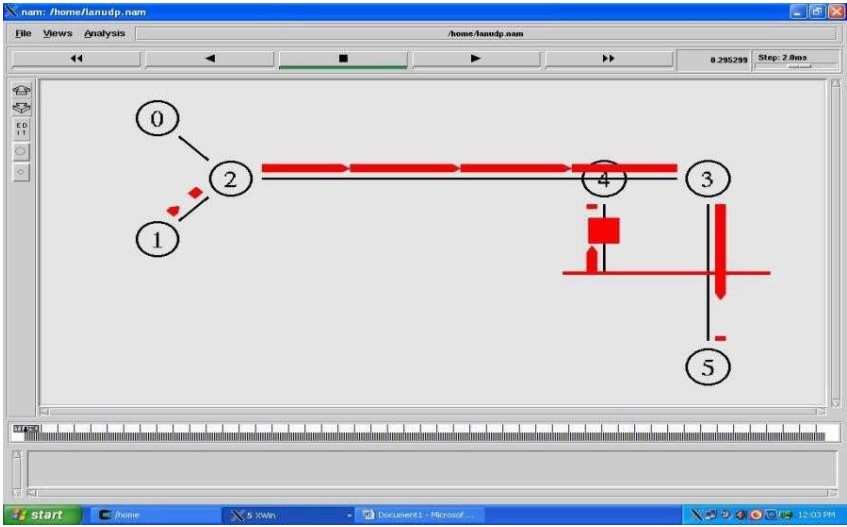
[\$ns now] set cwnd

\$file "\$now \$cwnd"

\$ns at [expr \$now+\$time]"plot Windows \$tcp sources \$file"

} \$ns run

OUTPUT:



Result:

EX.NO:8(b)
DATE:

STUDY OF TCP PERFORMANCE

AIM:

To study the performance comparison of Transmission Control Protocol.

INTRODUCTION:

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 (the other being Internet Protocol, or IP), so the entire suite is commonly referred to as TCP/IP.

TCP SEGMENT STRUCTURE:

A TCP segment consists of a segment header and a data section. The TCP header contains 10 mandatory fields, and an optional extension field (Options, pink background in table).

The data section follows the header. Its contents are the payload data carried for the application. The length of the data section is not specified in the TCP segment header. It can be calculated by subtracting the combined length of the TCP header and the encapsulating IP segment header from the total IP segment length (specified in the IP segment header).

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:

LISTEN : In case of a server, waiting for a connection request from any remote client.

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)

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) **ESTABLISHED** : the port is ready to receive/send data from/to the remote peer.

FIN-WAIT-1

FIN-WAIT-2

CLOSE-WAIT

CLOSING

LAST-ACK

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 TIMEWAIT for a maximum of four minutes. 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:

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.

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.

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

IMPLEMENTATION

ALGORITHM

Step 1: Define different color for data flows for NAM

Step 2: Open the NAM trace file

Step 3: Define a finished procedure

Step 4: Define a desired Number of nodes

Step 5: Create a links between the nodes

Step 6: Specifies the position of the node for NAM

Step 7: Set the queue size between desired nodes

Step 8: Setup a TCP Connection

Step 9: Setup a FTP over TCP Connection

Step 10: Define a procedure for plotting window size

Step 11: Specify the end of simulation

PROGRAM FOR TCP PERFORMANCE

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

```
$ns color 1 Blue
```

```
$ns color 2 Red
```

```
set tracefile1  
[open lantcp.tr  
w] set winfile  
[open WinFile  
w]
```

```
$ns trace-all $tracefile1
```

```
set namfile [open lantcp.nam w]
```

```
$ns namtrace-all $namfile
```

```
$ns duplex-link $n0 $n2 2Mb 10ms DropTail $ns
```

```
duplex-link $n1 $n2 2Mb 10ms DropTail
```

```
$ns simplex-link $n2 $n3 0.3Mb  
100ms DropTail $ns simplex-  
link $n3 $n2 0.3Mb 100ms  
DropTail
```

```
set lan [$ns newLan "$n4 $n5 $n3" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd  
Channel]
```

```
$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
```

```
$ns duplex-link-op $n3 $n2 orient left
```

```
$ns queue-limit $n2 $n3 20
```

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

```
$ns attach-agent $n0 $tcp
```

```
Set sink [new]
```

```
$ns connect $tcp $sink
```

```
$tcp set fid _1
```



```
set ftp [new Application/FTP]
```

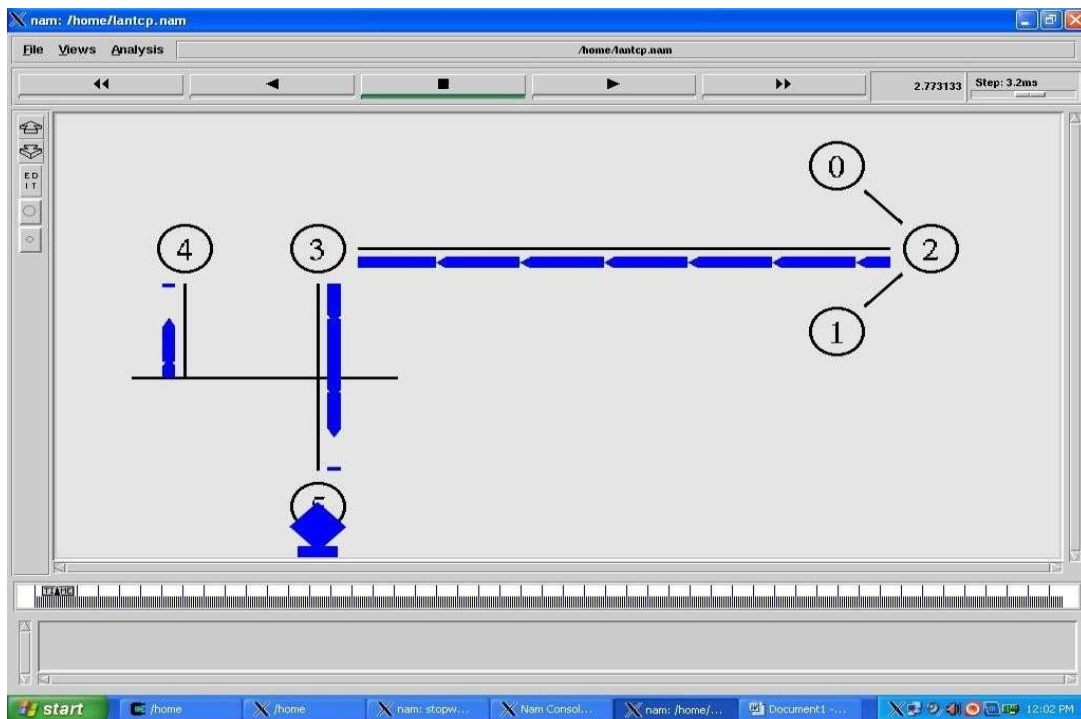
```
$ftp attach-agent $tcp
```

```
$ns at 1.0 "$ftp start" $ns at 124.0
```

```
"$ftp stop" proc plotWindow
```

```
{tcpSource file}
```

OUTPUT:



Result:

EX NO : 9(a)
DATE:

SIMULATION OF DISTANCE VECTOR ALGORITHM

AIM:

Algorithm:

Step 1: start the program.

Step 2: declare the global variables ns for creating a new simulator.

Step 3: set the color for packets.

Step 4: open the network animator file in the name of file2 in the write mode.

Step 5: open the trace file in the name of file 1 in the write mode.

Step 6: set the unicast routing protocol to transfer the packets in network.

Step 7: create the required no of nodes.

Step 8: create the duplex-link between the nodes including the delay time, bandwidth and dropping queue mechanism.

Step 9: give the position for the links between the nodes.

Step 10: set a twp. reno connection for source node.

Step 11: set the destination node using tcp sink.

Step 12: setup a ftp connection over the tcp connection.

Step 13: down the connection between any nodes at a particular time.

Step 14: reconnect the downed connection at a particular time.

Step 15: define the finish procedure.

Step 16: in the definition of the finish procedure declare the global variables ns,file1, file2.

Step 17: close the trace file and namefile and execute the network animation file. Step

18: at the particular time call the finish procedure. Step 19: stop the program.

Program:

```
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"

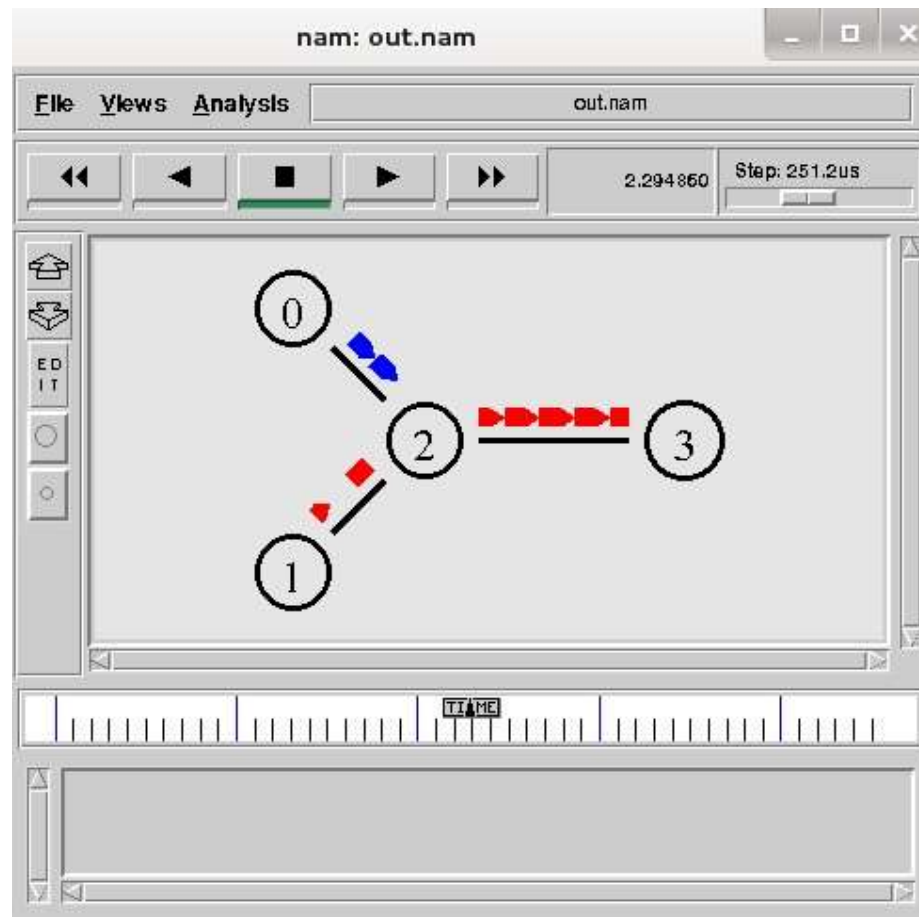
# Call the finish procedure after
# 5 seconds of simulation time
$ns at 5.0 "finish"

# 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:

EXP NO:9(b)
DATE:

SIMULATION OF LINK STATE ROUTING ALGORITHM

Aim:

Algorithm :

- Step 1: start the program.
- Step 2: declare the global variables ns for creating a new simulator.
- Step 3: set the color for packets.
- Step 4: open the network animator file in the name of file2 in the write mode.
- Step 5: open the trace file in the name of file 1 in the write mode.
- Step 6: set the multicast routing protocol to transfer the packets in network.
- Step 7: create the multicast capable no of nodes.
- Step 8: create the duplex-link between the nodes including the delay time, bandwidth and dropping.
- Step 9: give the position for the links between the nodes.
- Step 10: set a udp connection for source node.
- Step 11: set the destination node ,port and random false for the source and destination files.
- Step 12: setup a traffic generator CBR for the source and destination files.
- Step 13: down the connection between any nodes at a particular time. Step 14: create the receive agent for joining and leaving if the nodes in the group.
- Step 15: define the finish procedure.
- Step 16: in the definition of the finish procedure declare the global variables. Step 17: close the trace file and namefile and execute the network animation file. Step 18: at the particular time call the finish procedure. Step 19: stop the program.

Program:

```
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"

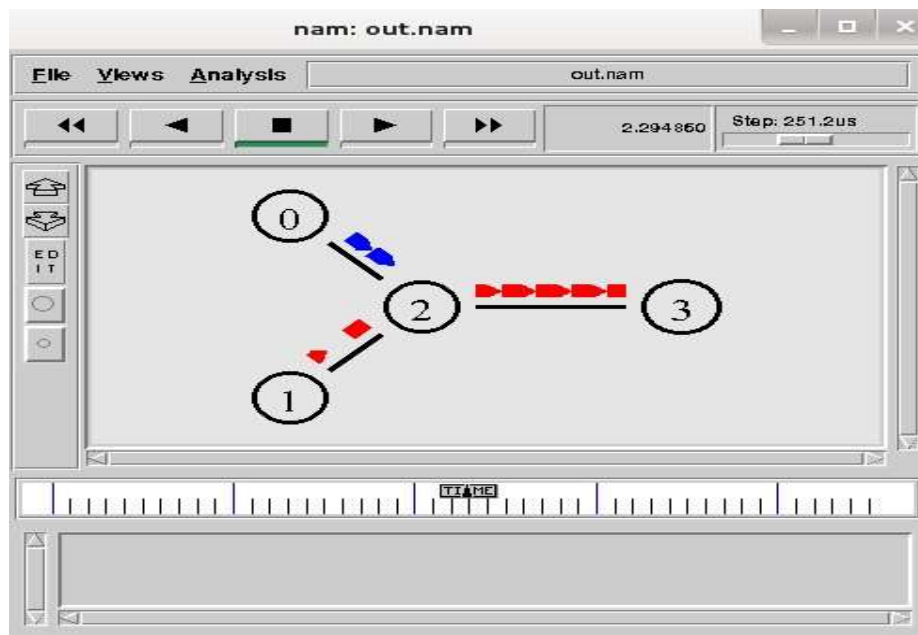
```



```
# 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:

EXP NO:10
DATE:

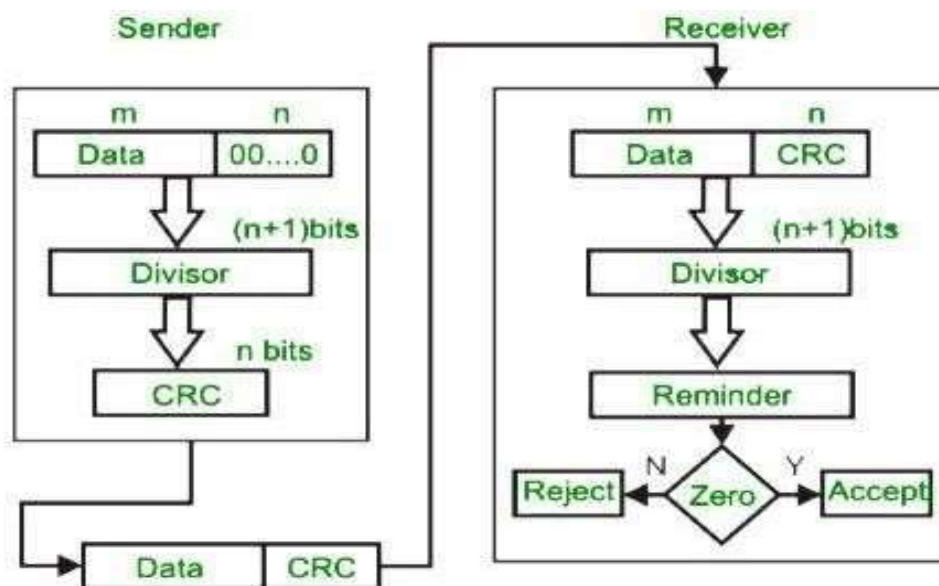
**IMPLEMENTATION OF CRC
USING RAW SOCKETS DATE:**

AIM :

Theory:

Cyclic redundancy check (CRC)

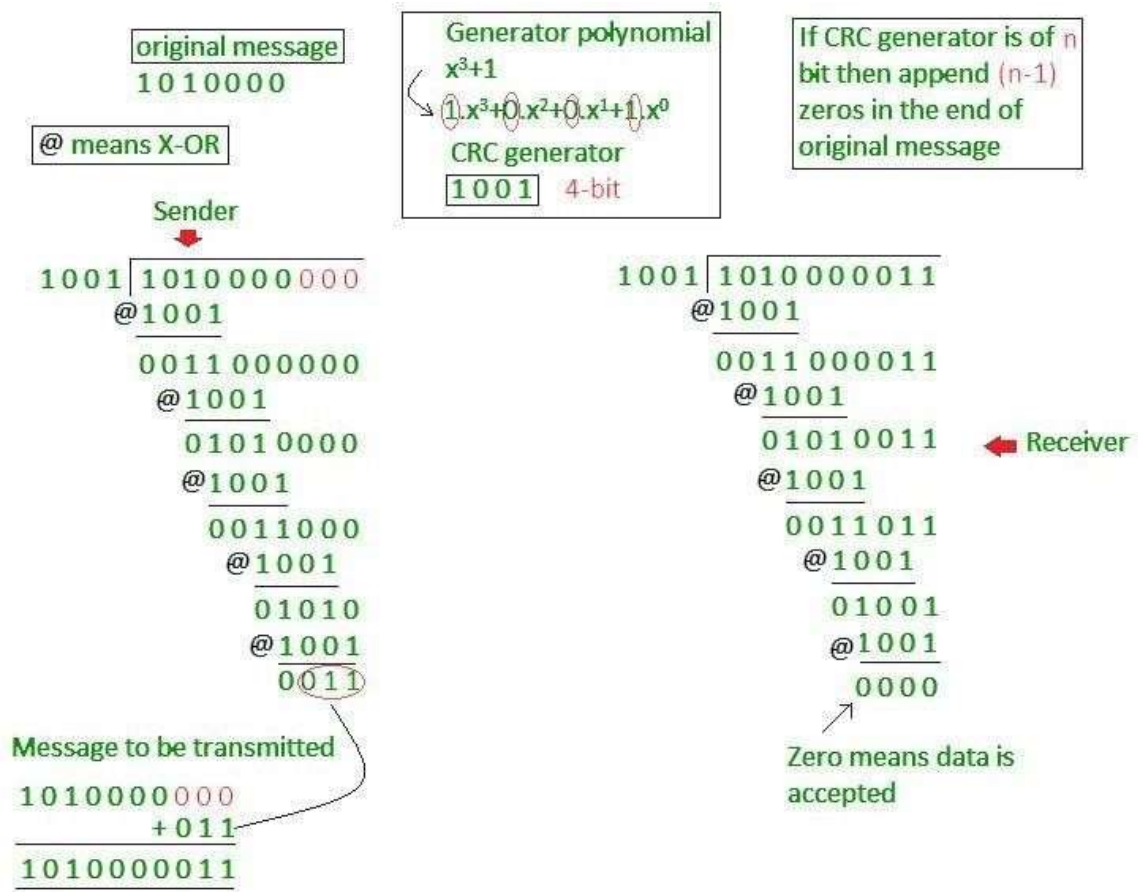
- Unlike checksum scheme, which is based on addition, CRC is based on binary division.
- In CRC, a sequence of redundant bits, called cyclic redundancy check bits, are appended to the end of data unit so that the resulting data unit becomes exactly divisible by a second, predetermined binary number.
- 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.



Procedure:

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

Example :



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();

    ring 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
r;
remainder
r =
""; 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());
}
}

```

Result: