
CS8581 NETWORKS LABORATORY

OBJECTIVES:

- To learn and use network commands.
- To learn socket programming.
- To implement and analyze various network protocols.
- To learn and use simulation tools.
- To use simulation tools to analyze the performance of various network protocols.

LIST OF EXPERIMENTS

1. Learn to use commands like tcpdump, netstat, ifconfig, nslookup and traceroute. Capture ping and trace route PDUs using a network protocol analyzer and examine.
2. Write a HTTP web client program to download a web page using TCP sockets.
3. Applications using TCP sockets like:
 - Echo client and echo server
 - Chat
 - File Transfer
4. Simulation of DNS using UDP sockets.
5. Write a code simulating ARP /RARP protocols.
6. Study of Network simulator (NS) and Simulation of Congestion Control Algorithms using NS.
7. Study of TCP/UDP performance using Simulation tool.
8. Simulation of Distance Vector/ Link State Routing algorithm.
9. Performance evaluation of Routing protocols using Simulation tool.
10. Simulation of error correction code (like CRC).

COURSE OUTCOMES

- CO 1. Implement various protocols using TCP and UDP
- CO 2. Implement applications using socket programming
- CO 3. Compare the performance of different transport layer protocols
- CO 4. Use simulation tools to analyze the performance of various network protocols
- CO 5. Analyze various routing algorithms.
- CO 6. Implement error correction codes.

CONTENT BEYOND SYLLABUS

Study of network programming in python

EX NO 1A: TO USE COMMANDS LIKE TCPDUMP, NETSTAT, IPCONFIG, NSLOOKUP AND TRACEROUTE.

DATE:

Aim

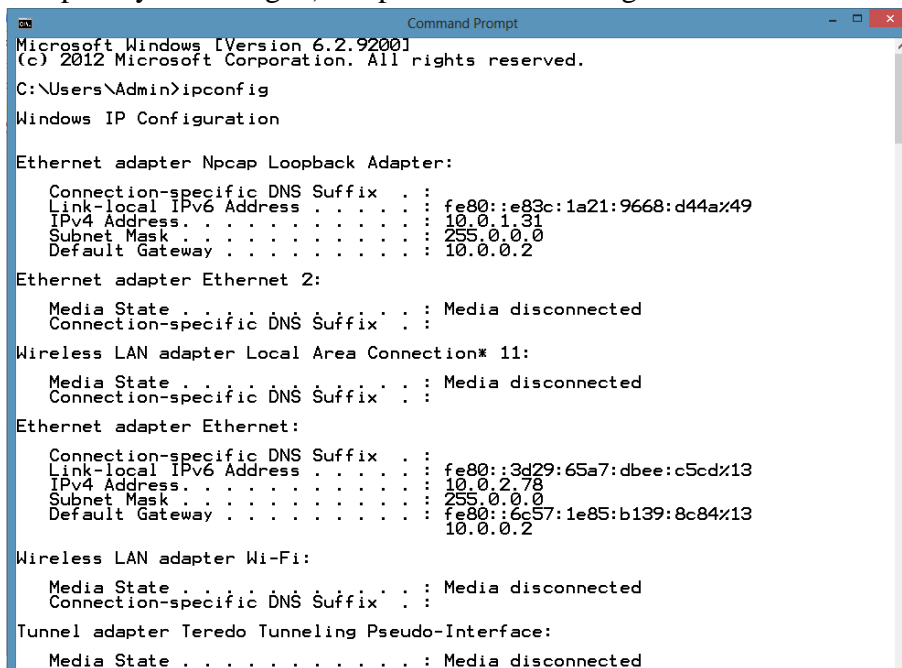
To study the basic networking commands

Algorithm

- Step 1.** Open command prompt
- Step 2.** Type the basic network commands
- Step 3.** Observe and note the output for each commands

Commands

- **C:\>ipconfig:** The ipconfig command displays information about the host (the computer your sitting at)computer TCP/IP configuration.



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\Admin>ipconfig

Windows IP Configuration

Ethernet adapter Npcap Loopback Adapter:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::e83c:1a21:9668:d44a%49
    IPv4 Address. . . . . : 10.0.1.31
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : 10.0.0.2

Ethernet adapter Ethernet 2:

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

Wireless LAN adapter Local Area Connection* 11:

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

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : 
    Link-local IPv6 Address . . . . . : fe80::3d29:65a7:dbec:c5cd%13
    IPv4 Address. . . . . : 10.0.2.78
    Subnet Mask . . . . . : 255.0.0.0
    Default Gateway . . . . . : fe80::6c57:1e85:b139:8c84%13
                                10.0.0.2

Wireless LAN adapter Wi-Fi:

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

Tunnel adapter Teredo Tunneling Pseudo-Interface:

    Media State . . . . . : Media disconnected
```

- **C:\>ipconfig /all**: This command displays detailed configuration information about your TCP/IP connection including Router, Gateway, DNS, DHCP, and type of Ethernet adapter in your system.

```
C:\>ipconfig/all
Windows IP Configuration

Host Name . . . . . : Fujitsu
Primary Dns Suffix . . . . . : 
Node Type . . . . . : Hybrid
IP Routing Enabled. . . . . : No
WINS Proxy Enabled. . . . . : No

Ethernet adapter Npcap Loopback Adapter:

Connection-specific DNS Suffix . : 
Description . . . . . : Microsoft KM-TEST Loopback Adapter
Physical Address. . . . . : 02-00-4C-4F-4F-50
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes
Link-local IPv6 Address . . . . . : fe80::e83c:1a21:9668:d44a%49(Preferred)
IPv4 Address. . . . . : 10.0.1.31(Preferred)
Subnet Mask . . . . . : 255.0.0.0
Lease Obtained. . . . . : Wednesday, July 24, 2019 9:28:19 AM
Lease Expires . . . . . : Thursday, August 1, 2019 9:28:19 AM
Default Gateway . . . . . : 10.0.0.2
DHCP Server . . . . . : 10.0.11.11
DHCPv6 IAID . . . . . : 822214732
DHCPv6 Client DUID. . . . . : 00-01-00-01-1E-D3-09-CE-E0-18-77-04-24-1D

DNS Servers . . . . . : 8.8.8.8
                        4.2.2.2
                        4.2.2.1
NetBIOS over Tcpip. . . . . : Enabled

Ethernet adapter Ethernet 2:

Media State . . . . . : Media disconnected
Connection-specific DNS Suffix . : 
Description . . . . . : Kaspersky Security Data Escort Adapter
Physical Address. . . . . : 00-FF-FB-2F-C0-9E
DHCP Enabled. . . . . : Yes
Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Local Area Connection* 11:

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

- **C:\>Ipconfig /renew**: Using this command will renew all your IP addresses that you are currently (leasing) borrowing from the DHCP server. This command is a quick problem solver if you are having connection issues, but does not work if you have been configured with a static IP address.

```
C:\>ipconfig/renew
Windows IP Configuration

An error occurred while renewing interface Npcap Loopback Adapter : unable to co
ntact your DHCP server. Request has timed out.
No operation can be performed on Ethernet 2 while it has its media disconnected.
No operation can be performed on Local Area Connection* 11 while it has its medi
a disconnected.
No operation can be performed on Wi-Fi while it has its media disconnected.

Ethernet adapter Npcap Loopback Adapter:

Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::e83c:1a21:9668:d44a%49
IPv4 Address. . . . . : 10.0.1.31
Subnet Mask . . . . . : 255.0.0.0
Default Gateway . . . . . : 10.0.0.2

Ethernet adapter Ethernet 2:

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

Wireless LAN adapter Local Area Connection* 11:

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

Ethernet adapter Ethernet:

Connection-specific DNS Suffix . : 
Link-local IPv6 Address . . . . . : fe80::3d29:65a7:dbee:c5cd%13
IPv4 Address. . . . . : 10.0.2.78
Subnet Mask . . . . . : 255.0.0.0
Default Gateway . . . . . : 10.0.0.2

Wireless LAN adapter Wi-Fi:

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

Tunnel adapter Teredo Tunneling Pseudo-Interface:
```

- **C:\>netstat**: Netstat displays a variety of statistics about a computers active TCP/IP connections. This tool is most useful when you're having trouble with TCP/IP applications such as HTTP, and FTP.

```

Command Prompt
Tunnel adapter {54D042AB-9962-46EE-841E-E4353F8C917D}:
Media State . . . : Media disconnected
Connection-specific DNS Suffix . : 
C:\Users\Admin>netstat
Active Connections
Proto Local Address Foreign Address State
TCP 10.0.1.31:50963 10.0.1.106:netbios-ssn SYN_SENT
TCP 10.0.2.78:2869 DESKTOP-LFV5J0A:59930 TIME_WAIT
TCP 10.0.2.78:2869 ITPRLABSYS34:1404 TIME_WAIT
TCP 10.0.2.78:2869 ITPRLABSYS34:1405 ESTABLISHED
TCP 10.0.2.78:50169 sa-in-f188:5228 ESTABLISHED
TCP 10.0.2.78:50180 maa05s05-in-f3:https ESTABLISHED
TCP 10.0.2.78:50282 172.217.194.189:https ESTABLISHED
TCP 10.0.2.78:50542 maa05s31-in-f5:https ESTABLISHED
TCP 10.0.2.78:50604 a184-26-168-178:http CLOSE_WAIT
TCP 10.0.2.78:50605 a184-26-168-178:http CLOSE_WAIT
TCP 10.0.2.78:50606 a104-108-152-88:http CLOSE_WAIT
TCP 10.0.2.78:50607 a104-108-152-88:http CLOSE_WAIT
TCP 10.0.2.78:50927 a104-108-152-88:http ESTABLISHED
TCP 10.0.2.78:50935 a23-59-29-72:http ESTABLISHED
TCP 10.0.2.78:50939 maa05s03-in-f14:https ESTABLISHED
TCP 10.0.2.78:50943 maa05s04-in-f10:https ESTABLISHED
TCP 10.0.2.78:50944 maa05s04-in-f10:https ESTABLISHED
TCP 10.0.2.78:50945 ec2-54-243-77-163:http TIME_WAIT
TCP 10.0.2.78:50959 65.55.252.93:https ESTABLISHED
TCP 10.0.2.78:50962 10.0.1.106:microsoft-ds SYN_SENT
TCP 10.0.2.78:50964 10.0.1.106:netbios-ssn SYN_SENT
TCP [fe80::3d29:65a7:dbec:5cd%13]:2869 Silicon:53879 TIME_WAIT
TCP [fe80::3d29:65a7:dbec:5cd%13]:2869 Silicon:53882 ESTABLISHED
C:\Users\Admin>

```

- **C:\>nslookup:** Nslookup is used for diagnosing DNS problems. If you can access a resource by specifying an IP address but not it's DNS you have a DNS problem.

```

Command Prompt - nslookup
C:\Users\Admin>nslookup
Default Server: dns.google
Address: 8.8.8.8
>

```

- **C:\>pathping:** Pathping is unique to Window's, and is basically a combination of the Ping and Tracert commands. Pathping traces the route to the destination address then launches a 25 second test of each router along the way, gathering statistics on the rate of data loss along each hop.

```

Command Prompt
Connection-specific DNS Suffix . : 
Tunnel adapter {54D042AB-9962-46EE-841E-E4353F8C917D}:
Media State . . . : Media disconnected
Connection-specific DNS Suffix . : 
C:\>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:\>ping:** Ping is the most basic TCP/IP command, and it's the same as placing a phone call to your best friend. You pick up your telephone and dial a number, expecting your best friend to reply with "Hello" on the other end. Computers make phone calls to each other over a network by using a Ping command. The Ping

commands main purpose is to place a phone call to another computer on the network, and request an answer. Ping has 2 options it can use to place a phone call to another computer on the network. It can use the computers name or IP address.

```
C:\>ping 10.0.2.123

Pinging 10.0.2.123 with 32 bytes of data:
Reply from 10.0.2.123: bytes=32 time=6ms TTL=128
Reply from 10.0.2.123: bytes=32 time<1ms TTL=128
Reply from 10.0.2.123: bytes=32 time=0ms TTL=128
Reply from 10.0.2.123: bytes=32 time=1ms TTL=128

Ping statistics for 10.0.2.123:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 8ms, Average = 3ms

C:\>
```

- **C:\>route:** The route command displays the computers routing table. A typical computer, with a single network interface, connected to a LAN, with a router is fairly simple and generally doesn't pose any network problems. But if you're having trouble accessing other computers on your network, you can use the route command to make sure the entries in the routing table are correct.

```
C:\>route

Manipulates network routing tables.

ROUTE [-f] [-p] [-4|-6] command [destination]
      [MASK netmask] [gateway] [METRIC metric] [IF interface]

-f          Clears the routing tables of all gateway entries. If this is
            used in conjunction with one of the commands, the tables are
            cleared prior to running the command.

-p          When used with the ADD command, makes a route persistent across
            boots of the system. By default, routes are not preserved
            when the system is restarted. Ignored for all other commands,
            which always affect the appropriate persistent routes.

-4          Force using IPv4.

-6          Force using IPv6.

command     One of these:
            PRINT    Prints a route
            ADD      Adds a route
            DELETE   Deletes a route
            CHANGE   Modifies an existing route

destination Specifies the host.

MASK         Specifies that the next parameter is the 'netmask' value.

netmask      Specifies a subnet mask value for this route entry.
            If not specified, it defaults to 255.255.255.255.

gateway      Specifies gateway.

interface    the interface number for the specified route.

METRIC       specifies the metric, ie. cost for the destination.

All symbolic names used for destination are looked up in the network database
file NETWORKS. The symbolic names for gateway are looked up in the host name
database file HOSTS.

If the command is PRINT or DELETE, Destination or gateway can be a wildcard,
(wildcard is specified as a star '*'), or the gateway argument may be omitted.

If Dest contains a * or ?, it is treated as a shell pattern, and only
matching destination routes are printed. The '*' matches any string,
and '?' matches any one char. Examples: 157.*.1. 157.*. 127.*. *224*.
```

- **C:\>tracert:** The tracert command displays a list of all the routers that a packet has to go through to get from the computer where tracert is run to any other computer on the internet.

```
C:\>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.
```

Result

Thus the above list of commands has been studied.

EX NO 1B: CAPTURE PING PDUS USING A NETWORK PROTOCOL ANALYZER AND EXAMINE.

DATE:

Wireshark Protocol Analyzer

Wireshark is the world's foremost and widely-used network protocol analyzer. It lets you see what's happening on your network at a microscopic level and is the de facto standard across many commercial and non-profit enterprises, government agencies, and educational institutions.

Wireshark has a rich feature set which includes the following:

- Deep inspection of hundreds of protocols, with more being added all the time
- Live capture and offline analysis
- Standard three-pane packet browser
- Multi-platform: Runs on Windows, Linux, macOS, Solaris, FreeBSD, NetBSD, and many others
- Captured network data can be browsed via a GUI, or via the TTY-mode TShark utility
- The most powerful display filters in the industry
- Coloring rules can be applied to the packet list for quick, intuitive analysis
- Output can be exported to XML, PostScript®, CSV, or plain text

Aim:

To capture ping PDUS using a network protocol analyzer and examine.

Activity 1 - Capture Ping Echo Traffic

To capture ICMP Echo traffic:

- Step 1. Start a Wireshark capture.
- Step 2. Use ping <default gateway address> to ping the default gateway address.
- Step 3. Stop the Wireshark capture.

Activity 2 - Analyze ICMP Echo Request Traffic

To analyze ICMP Echo Request traffic:

Observe the traffic captured in the top Wireshark packet list pane. Look for traffic with ICMP listed as the protocol. To view only ICMP traffic, type icmp (lower case) in the Filter box and press Enter.

- Step 1.** Select the first ICMP packet, labeled Echo (ping) request.
- Step 2.** Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
- Step 3.** Expand Internet Control Message Protocol to view ICMP details.
- Step 4.** Observe the Type. Notice that the type is 8 (Echo (ping) request).
- Step 5.** Select Data in the middle Wireshark packet details pane to highlight the data portion of the frame.

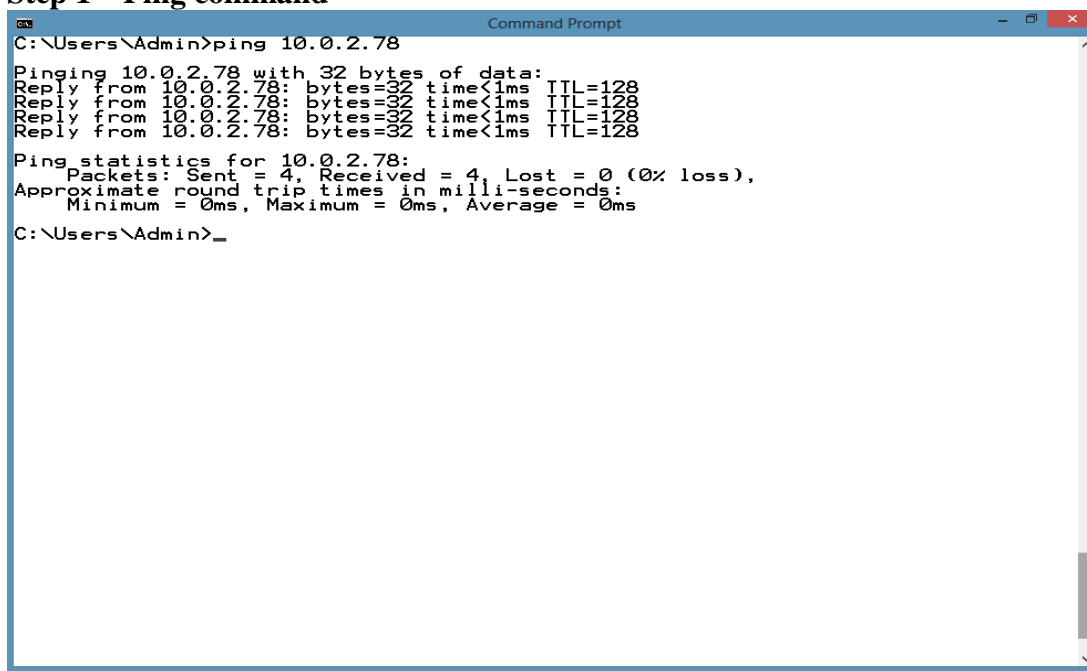
Step 6. Observe the packet contents in the bottom Wireshark packet bytes pane.
Notice that Windows sends an alphabet sequence during ping requests.

Activity 3 - Analyze ICMP Echo Reply Traffic

To analyze ICMP Echo Reply traffic:

- Step 1.** In the top Wireshark packet list pane, select the second ICMP packet, labeled Echo (ping) reply.
- Step 2.** Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
- Step 3.** Expand Internet Control Message Protocol to view ICMP details.
- Step 4.** Observe the Type. Notice that the type is 0 (Echo (ping) reply).
- Step 5.** Select Data in the middle Wireshark packet details pane to highlight the data portion of the frame.
- Step 6.** Observe the packet contents in the bottom Wireshark packet bytes pane.
Notice that the reply echoes the request sequence.
- Step 7.** Close Wireshark to complete this activity. Quit without Saving to discard the captured traffic.

Step 1 – Ping command



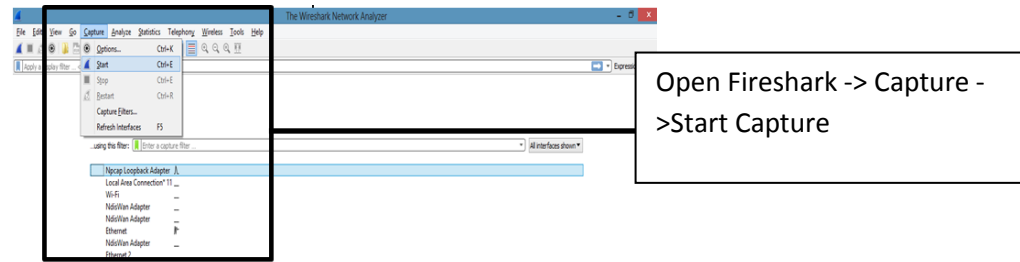
```
C:\Users\Admin>ping 10.0.2.78

Pinging 10.0.2.78 with 32 bytes of data:
Reply from 10.0.2.78: bytes=32 time<1ms TTL=128
Reply from 10.0.2.78: bytes=32 time<1ms TTL=128
Reply from 10.0.2.78: bytes=32 time<1ms TTL=128
Reply from 10.0.2.78: bytes=32 time<1ms TTL=128

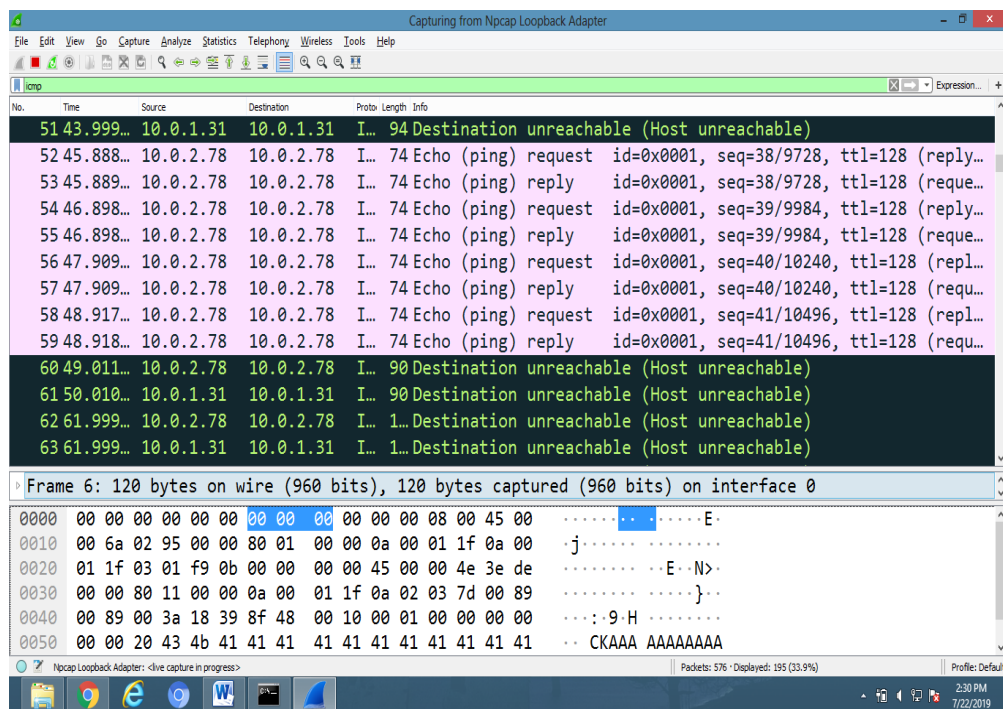
Ping statistics for 10.0.2.78:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\Users\Admin>_
```

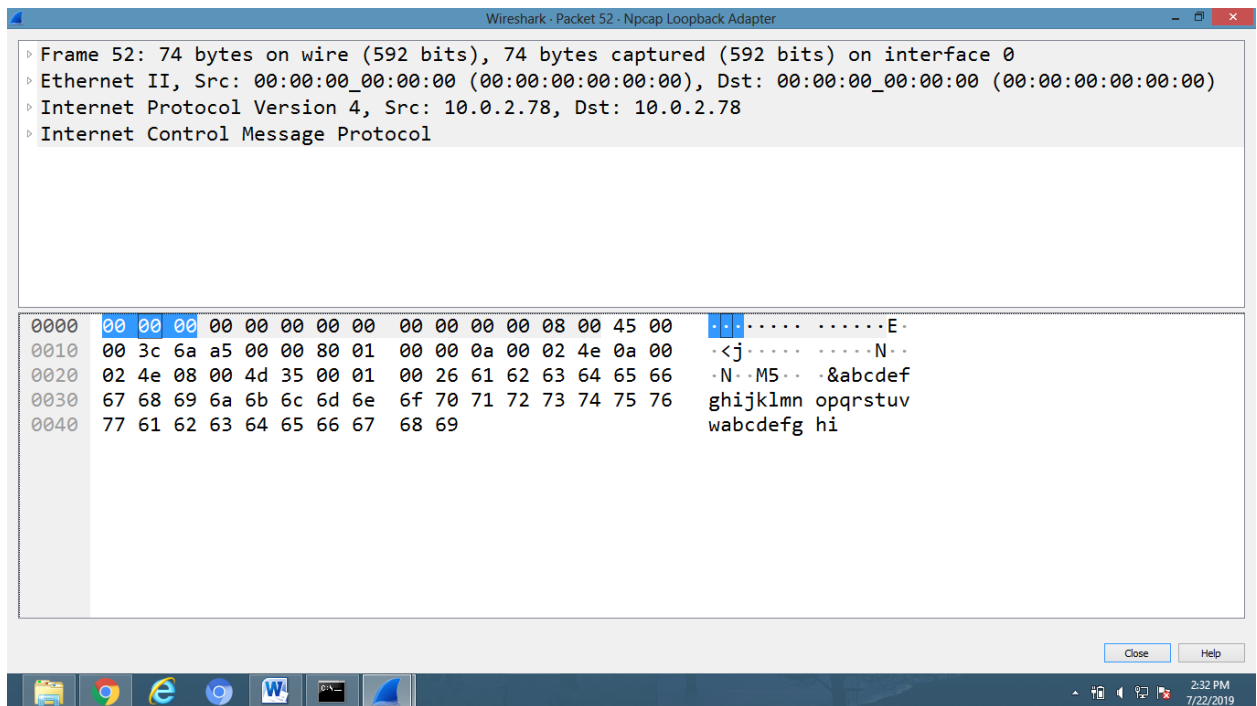
Step 2 – Wireshark Capture



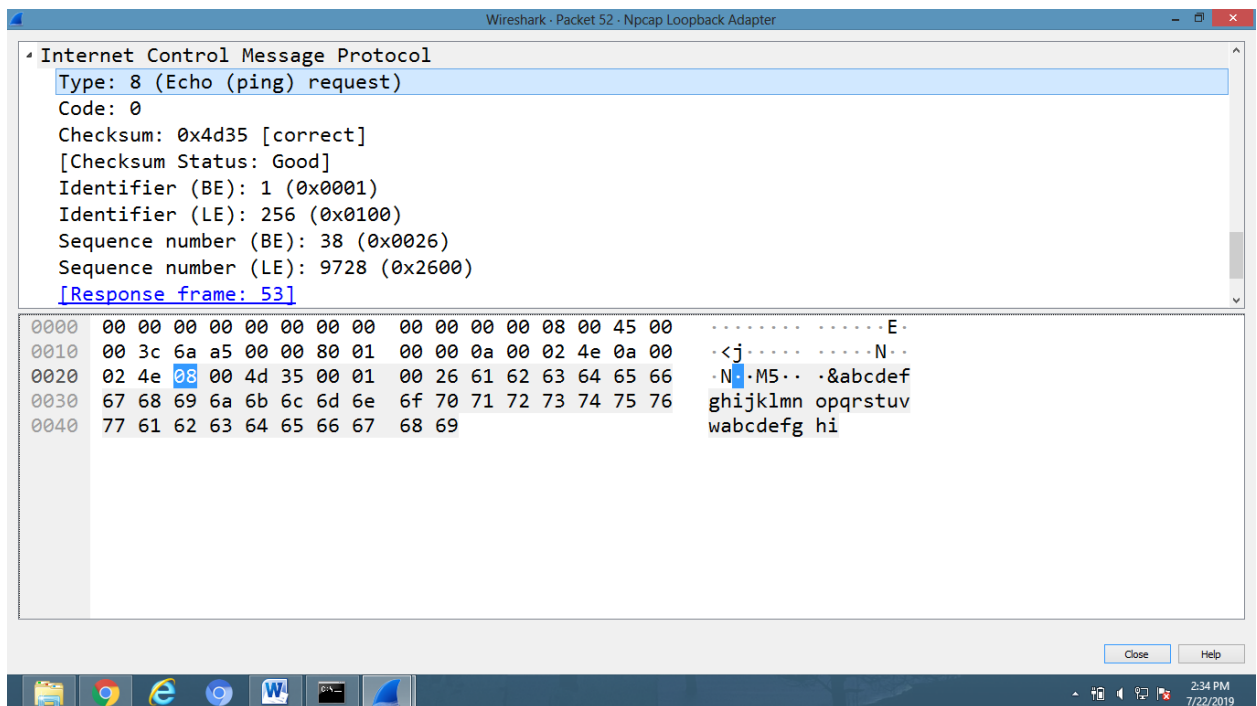
Step 3 – Select the Ping Request – By double clicking



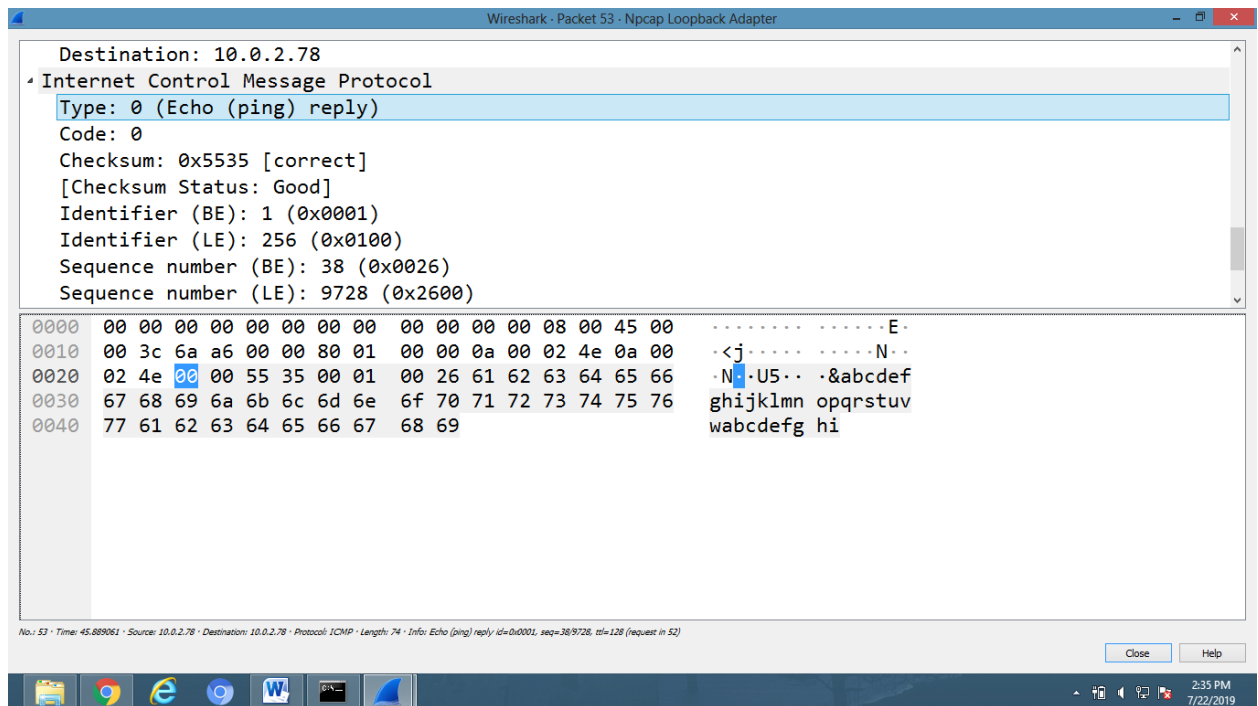
Ping Request PDU



Ping PDU– Type 8 for Ping Request



Ping PDU– Type 0 for Ping Request



Result

Thus the ping PDUS has been studied and examined using a wireshark network protocol analyzer.

EX NO 1C: CAPTURE TRACE ROUTE PDUS USING A NETWORK PROTOCOL ANALYZER AND EXAMINE.

DATE:

Aim:

To capture TRACE ROUTE PDUS using a network protocol analyzer and examine.

Activity 1 - Capture Tracert Traffic

To capture ICMP tracert traffic:

- Step 1.** Start a Wireshark capture.
- Step 2.** Open a command prompt.
- Step 3.** Type `tracert -d 8.8.8.8` and press Enter to trace the route to one of Google's public DNS servers. The `-d` option prevents DNS name resolution, which in this case will improve performance and reduce the amount of captured traffic.
- Step 4.** When the trace is complete, close the command prompt.
- Step 5.** Stop the Wireshark capture.

Activity 2 - Analyze Tracert Traffic

To analyze tracert traffic:

- Step 1.** Observe the traffic captured in the top Wireshark packet list pane. Look for traffic with ICMP listed as the protocol. To view only ICMP traffic, type `icmp` (lower case) in the Filter box and press Enter.
- Step 2.** Select the first ICMP packet, labeled Echo (ping) request.
- Step 3.** Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
- Step 4.** Expand Internet Protocol Version 4 to view IPv4 details.
- Step 5.** Observe the Time to live. Notice that the time to live is set to 1.
- Step 6.** Expand Internet Control Message Protocol to view ICMP details.
- Step 7.** Observe the Type. Notice that the type is 8 (Echo (ping) request). Tracert is performed through a series of ICMP Echo requests, varying the Time-To-Live (TTL) until the destination is found.
- Step 8.** In the top Wireshark packet list pane, select the second ICMP packet, labeled Time-to-live exceeded.
- Step 9.** Observe the packet details in the middle Wireshark packet details pane. Notice that it is an Ethernet II / Internet Protocol Version 4 / Internet Control Message Protocol frame.
- Step 10.** Expand Internet Protocol Version 4 to view IPv4 details.
- Step 11.** Observe the Source. This is the IP address of the router where the time was exceeded.
- Step 12.** Expand Internet Control Message Protocol to view ICMP details.
- Step 13.** Observe the Type. Notice that the type is 11 (Time-to-live exceeded).
- Step 14.** Observe the Code. Notice that the code is 0 (Time to live exceeded in transit).
- Step 15.** Observe the fields that follow. Notice that the contents of the request packet are returned with the time exceeded error.

Step 16. Continue selecting alternate ICMP Echo Request and ICMP Time-To-Live Exceeded packets. Notice that the request is repeated three times for each time-to-live count, and each reply indicates the IP address of the router where the time to live was exceeded.

Step 17. Close Wireshark to complete this activity. Quit without Saving to discard the captured traffic.

Trace Route in Command Prompt

```

C:\Users\Admin>tracert -d 8.8.8.8
Tracing route to 8.8.8.8 over a maximum of 30 hops
  0  5 ms    5 ms    5 ms  182.79.141.37
  1  4 ms    4 ms    4 ms  182.79.135.105
  2  4 ms    4 ms    4 ms  182.79.198.26
  3  72 ms   72 ms   72 ms  72.14.211.198
  4  108 ms  108 ms  108 ms  108.170.253.113
  5  216 ms  216 ms  216 ms  216.239.43.235
  6    6 ms    6 ms    6 ms  8.8.8.8
Trace complete.

C:\Users\Admin>tracert -d 8.8.8.8
Tracing route to 8.8.8.8 over a maximum of 30 hops
  0  5 ms    4 ms    4 ms  182.74.176.181
  1  4 ms    4 ms    4 ms  182.74.176.181
  2  7 ms    7 ms    7 ms  182.79.141.37
  3  1 ms    1 ms    1 ms  182.79.135.105
  4  1 ms    1 ms    1 ms  182.79.198.26
  5  72 ms   72 ms   72 ms  72.14.211.198
  6  108 ms  108 ms  108 ms  108.170.253.113
  7  216 ms  216 ms  216 ms  216.239.43.235
  8    6 ms    6 ms    6 ms  8.8.8.8
Trace complete.

C:\Users\Admin>tracert -d 8.8.8.8
Tracing route to 8.8.8.8 over a maximum of 30 hops
  0  5 ms    4 ms    4 ms  182.74.176.181
  1  4 ms    4 ms    4 ms  182.74.176.181
  2  7 ms    7 ms    7 ms  182.79.141.37
  3  1 ms    1 ms    1 ms  182.79.135.105
  4  1 ms    1 ms    1 ms  182.79.198.26
  5  72 ms   72 ms   72 ms  72.14.211.198
  6  108 ms  108 ms  108 ms  108.170.253.113
  7  216 ms  216 ms  216 ms  216.239.43.235
  8    6 ms    6 ms    6 ms  8.8.8.8
Trace complete.

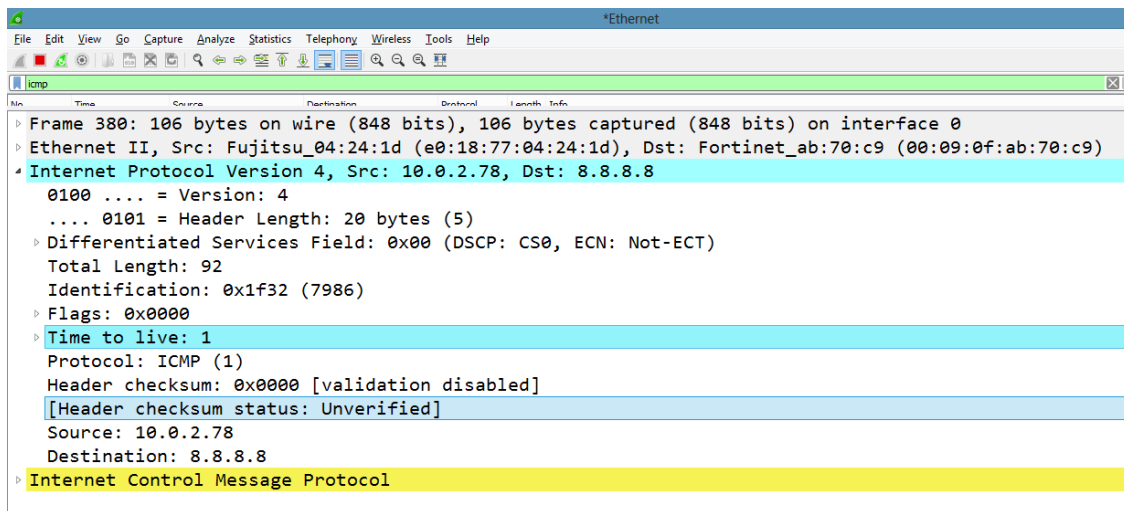
C:\Users\Admin>

```

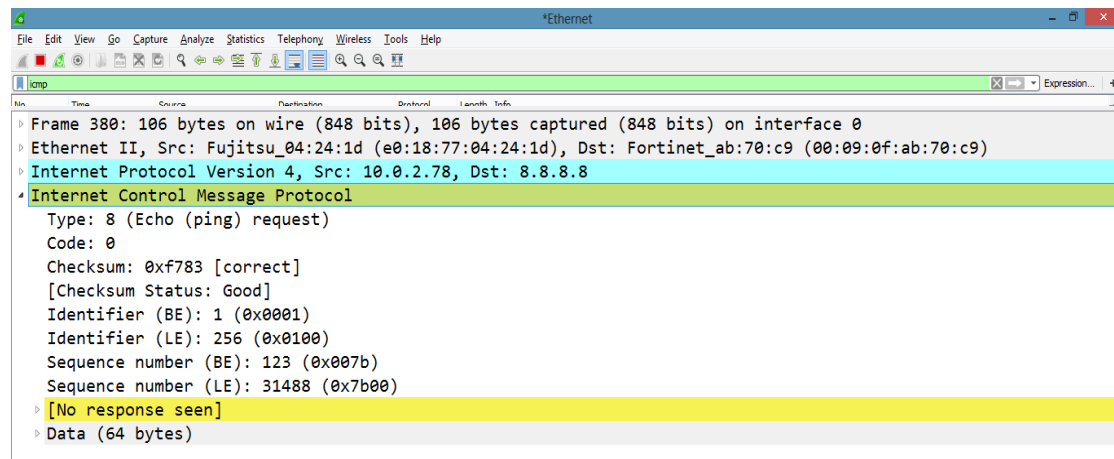
Wireshark – TraceRoute Analyzed

No.	Time	Source	Destination	Protocol	Length	Info
3...	1.794275	10.0.2.78	8.8.8.8	ICMP	106	Echo (ping) request id=0x0001, seq=123/31488, ttl=1 (no resp...
3...	1.799485	182.74.176...	10.0.2.78	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
3...	1.800983	10.0.2.78	8.8.8.8	ICMP	106	Echo (ping) request id=0x0001, seq=124/31744, ttl=1 (no resp...
3...	1.804775	182.74.176...	10.0.2.78	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
3...	1.806243	10.0.2.78	8.8.8.8	ICMP	106	Echo (ping) request id=0x0001, seq=125/32000, ttl=1 (no resp...
3...	1.810520	182.74.176...	10.0.2.78	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
5...	2.810249	10.0.2.78	8.8.8.8	ICMP	106	Echo (ping) request id=0x0001, seq=126/32256, ttl=2 (no resp...
5...	2.814591	182.74.176...	10.0.2.78	ICMP	70	Time-to-live exceeded (Time to live exceeded in transit)
6...	2.815737	10.0.2.78	8.8.8.8	ICMP	106	Echo (ping) request id=0x0001, seq=127/32512, ttl=2 (no resp...

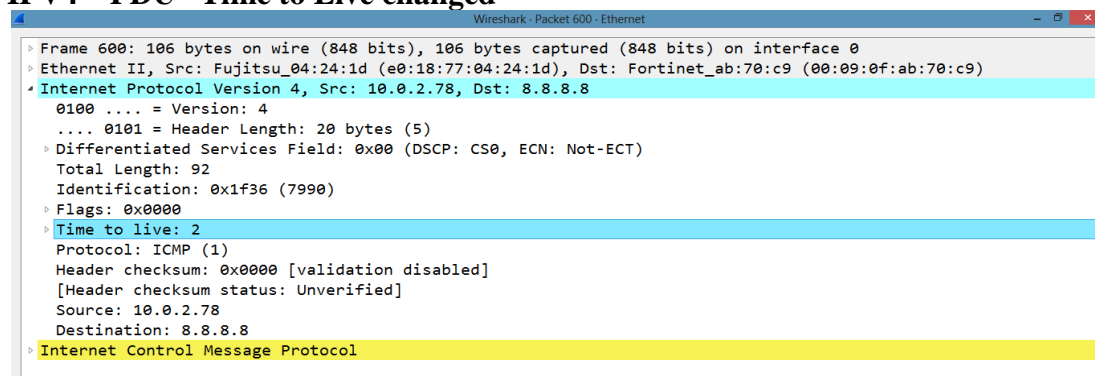
IPV4 – PDU



ICMP - PDU



IPV4 – PDU –Time to Live changed



Result Thus the trace route PDUS has been studied and examined using a Wireshark network protocol analyzer.

EX NO 2: A HTTP WEB CLIENT PROGRAM TO DOWNLOAD A WEB PAGE USING TCP SOCKETS

DATE:

Aim

To download a web page through a HTTP web client program using TCP sockets

Algorithm

Step 1. Create a URL object and pass url as string to download the webpage.

URL example = new URL(pass url of webpage you want to download)

Step 2. Create Buffered Reader object and pass openStream(). Method of URL in Input Stream object.

Step 3. Create a string object to read each line one by one from stream.

Step 4. Write each line in html file where webpage will be downloaded.

Step 5. Close all objects.

Step 6. Catch exceptions if url failed to download.

Program

```
import java.io.*;
import java.net.URL;
import java.net.MalformedURLException;
class Download {
    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("WebPage_VEC.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 = "http://www.velammal.edu.in/";
    DownloadWebPage(url);
}
}

```

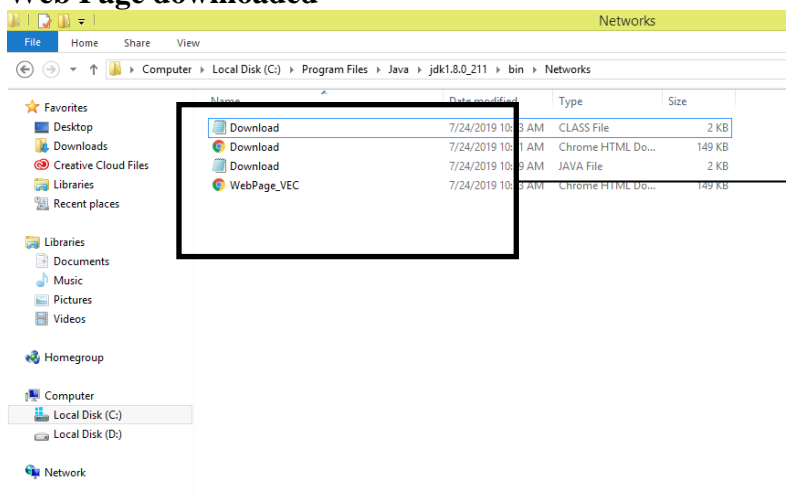
Output

```

Administrator: Command Prompt
C:\Program Files\Java\jdk1.8.0_211\bin\Networks>javac Download.java
C:\Program Files\Java\jdk1.8.0_211\bin\Networks>java Download
Successfully Downloaded.
C:\Program Files\Java\jdk1.8.0_211\bin\Networks>

```

Web Page downloaded



WebPage_VEC.HTML
downloaded and saved for
offline viewing

Result

Thus the java program to download a web page through a HTTP web client program using TCP sockets has been completed successfully.

EX NO 3A: Echo client and echo server using TCP sockets

DATE:

Aim

To implement echo client and server using TCP sockets in JAVA

Algorithm

CLIENT SIDE

- Step 1. Start the program.
- Step 2. Create a socket which binds the IP address of server and the port address to acquire service. After establishing connection send a data to server
- Step 3. Receive and print the same data from server.
- Step 4. Close the socket
- Step 5. End the program.

SERVER SIDE

- Step 1. Start the program.
- Step 2. Create a server socket to activate the port address.
- Step 3. Create a socket for the server socket which accepts the connection.
- Step 4. After establishing connection receive the data from client.
- Step 5. Print and send the same data to client.
- Step 6. Close the socket.
- Step 7. End the program.

Client to Server Program

Program

WishesServer. Java

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
import java.io.DataInputStream;
public class WishesServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket sersock = new ServerSocket(5000);
        System.out.println("server is ready"); // message to know the server is running
```



```

        Socket sock = sersock.accept();
        InputStream istream = sock.getInputStream();
        DataInputStream dstream = new DataInputStream(istream);
        String message2 = dstream.readLine();
        System.out.println(message2);
        dstream.close(); istream.close(); sock.close(); sersock.close();
    }
}

```

WishesClient. Java

```

import java.net.Socket;
import java.io.OutputStream;
import java.io.DataOutputStream;

public class WishesClient
{
    public static void main(String args[]) throws Exception
    {
        Socket sock = new Socket("127.0.0.1", 5000);
        String message1 = "Accept Message from Client";

        OutputStream ostream = sock.getOutputStream();
        DataOutputStream dos = new DataOutputStream(ostream);
        dos.writeBytes(message1);
        dos.close();
        ostream.close();
        sock.close();
    }
}

```

Output

Server Side

```

C:\Network>java WishesServer
server is ready
Accept Best Wishes, Serverji
C:\Network>

```

Client Side

```

C:\Network>java WishesClient
C:\Network>

```

Server to Client Program

To print Server's date in Client

DateInfoServer

```
import java.net.*;
import java.io.*;
public class DateInfoServer
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket sersock = new ServerSocket(7000);
        System.out.println("Server ready.....");
        Socket sock = sersock.accept( );

        OutputStream ostream = sock.getOutputStream();
        BufferedWriter bw1 = new BufferedWriter(new OutputStreamWriter(ostream));
        String s2 = "Thanks client for your calling on " + new java.util.Date();
        bw1.write(s2);

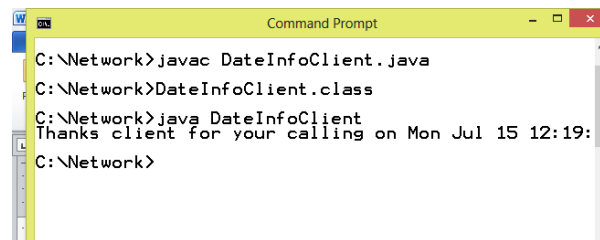
        bw1.close(); ostream.close(); sock.close(); sersock.close( );
    }
}
```

DateInfoClient

```
import java.io.*;
import java.net.*;
public class DateInfoClient
{
    public static void main(String args[]) throws Exception
    {
        Socket sock = new Socket("127.0.0.1", 7000);
        InputStream istream = sock.getInputStream();
        BufferedReader br1 = new BufferedReader(new InputStreamReader(istream));
        String s1 = br1.readLine();
        System.out.println(s1);
        br1.close(); istream.close(); sock.close( );
    }
}
```

Output

Client Side

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a yellow title bar. The text inside shows the compilation and execution of a Java client program.

```
C:\Network>javac DateInfoClient.java
C:\Network>DateInfoClient.class
C:\Network>java DateInfoClient
Thanks client for your calling on Mon Jul 15 12:19:
C:\Network>
```

Server Side

A screenshot of a Windows Command Prompt window titled "Command Prompt". The window has a grey title bar. The text inside shows the compilation and execution of a Java server program.

```
C:\Network>javac DateInfoServer.java
C:\Network>java DateInfoServer
Server ready.....
C:\Network>
```

Result

Thus the JAVA program to implement echo client and server using TCP sockets has been completed successfully

EX NO 3B: CHAT using TCP Sockets

DATE:

Aim

To implement chat using TCP sockets

Algorithm

CLIENT

- Step 1.** Start the program
- Step 2.** Include necessary package in java
- Step 3.** To create a socket in client to server.
- Step 4.** The client establishes a connection to the server.
- Step 5.** The client accept the connection and to send the data from client to server.
- Step 6.** The client communicates the server to send the end of the message
- Step 7.** Stop the program.

SERVER

- Step 1.** Start the program
- Step 2.** Include necessary package in java
- Step 3.** To create a socket in server to client
- Step 4.** The server establishes a connection to the client.
- Step 5.** The server accept the connection and to send the data from server to client
vice versa
- Step 6.** The server communicates with the client to send the end of the message.
- Step 7.** Stop the program.

Program

Chat Client

```
import java.io.*;
import java.net.*;
public class ChatClient
{
    public static void main(String[] args) throws Exception
    {
        Socket sock = new Socket("127.0.0.1", 3000);
        // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // sending to client (pwrite object)
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);

        // receiving from server ( receiveRead object)
```

```

InputStream istream = sock.getInputStream();
BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));

System.out.println("Start the chitchat, type and press Enter key");

String receiveMessage, sendMessage;
while(true)
{
    sendMessage = keyRead.readLine(); // keyboard reading
    pwrite.println(sendMessage);    // sending to server
    pwrite.flush();                // flush the data
    if((receiveMessage = receiveRead.readLine()) != null) //receive from server
    {
        System.out.println(receiveMessage); // displaying at DOS prompt
    }
}
}

```

Chat Server

```

import java.io.*;
import java.net.*;
public class ChatServer
{
    public static void main(String[] args) throws Exception
    {
        ServerSocket sersock = new ServerSocket(3000);
        System.out.println("Server ready for chatting");
        Socket sock = sersock.accept();
        // reading from keyboard (keyRead object)
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        // sending to client (pwrite object)
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream, true);

        // receiving from server ( receiveRead object)
        InputStream istream = sock.getInputStream();
        BufferedReader receiveRead = new BufferedReader(new InputStreamReader(istream));
        String receiveMessage, sendMessage;
        while(true)
        {
            if((receiveMessage = receiveRead.readLine()) != null)

```

```

    {
        System.out.println(receiveMessage);
    }
    sendMessage = keyRead.readLine();
    pwrite.println(sendMessage);
    pwrite.flush();
}
}
}

```

Output

Chat Client

```

Command Prompt - java ChatClient
C:\Network>javac ChatClient.java
C:\Network>java ChatClient
Start the chitchat, type and press Enter key
Hello Server
Hi Client
Java Client Program
JavaJaHJ
Java
Java Chat using TCP

```

Chat Server

```

Command Prompt - java ChatServer
C:\Network>javac ChatServer.java
C:\Network>java ChatServer
Server ready for chatting
Hello Server
Hi Client
Java Client Program
JavaJaHJ
Java
Java Chat using TCP

```

Result

Thus the JAVA program to implement chat using TCP sockets has been completed successfully.

EX NO 3C: File Transfer using TCP sockets

DATE:

Aim

To implement File Transfer using TCP/IP in a client and server

Algorithm

CLIENT SIDE

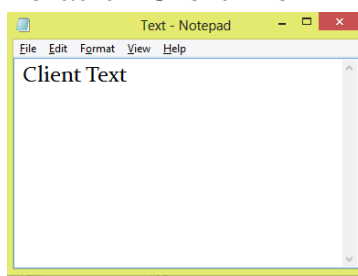
- Step 1.** Start.
- Step 2.** Establish a connection between the Client and Server.
- Step 3.** Implement a client that can send two requests
 - i. To get a file from the server.
 - ii. To put or send a file to the server.
- Step 4.** After getting approval from the server ,the client either get file from the server or send file to the server.

SERVER SIDE

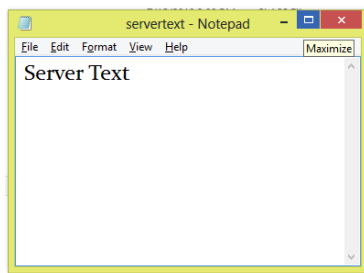
- Step 1.** Start.
- Step 2.** Implement a server socket that listens to a particular port number.
- Step 3.** Server reads the filename and sends the data stored in the file for the 'get' request.
- Step 4.** It reads the data from the input stream and writes it to a file in the server for the 'put' instruction.
- Step 5.** Exit upon client's request.
- Step 6.** Stop.

Output

Text.txt – Client File



Servertext.txt – Server file content to be written in Client File



File Transfer Server

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

```
public class FileTransferServer {

    public static void main(String[] args) throws Exception {
        //Initialize Sockets
        ServerSocket ssock = new ServerSocket(5000);
        Socket socket = ssock.accept();

        //The InetAddress specification
        InetAddress IA = InetAddress.getByName("localhost");

        //Specify the file
        File file = new File("d:\\serverfile.txt");
        FileInputStream fis = new FileInputStream(file);
        BufferedInputStream bis = new BufferedInputStream(fis);

        //Get socket's output stream
        OutputStream os = socket.getOutputStream();

        //Read File Contents into contents array
        byte[] contents;
        long fileLength = file.length();
        long current = 0;

        long start = System.nanoTime();
```



```

while(current!=fileLength){
    int size = 10000;
    if(fileLength - current >= size)
        current += size;
    else{
        size = (int)(fileLength - current);
        current = fileLength;
    }
    contents = new byte[size];
    bis.read(contents, 0, size);
    os.write(contents);
    System.out.print("Sending file ... "+(current*100)/fileLength+"% complete!");
}

os.flush();
//File transfer done. Close the socket connection!
socket.close();
sock.close();
System.out.println("File sent succesfully!");
}
}

```

```

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

```

```

public class FileTransferClient {

```

```

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

```

```

        //Initialize socket

```

```

        Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
        byte[] contents = new byte[10000];

```

```

        //Initialize the FileOutputStream to the output file's full path.

```

```

        FileOutputStream fos = new FileOutputStream("d:\\clientfile.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        InputStream is = socket.getInputStream();

```

```

//No of bytes read in one read() call
int bytesRead = 0;

while((bytesRead=is.read(contents))!=-1)
    bos.write(contents, 0, bytesRead);

bos.flush();
socket.close();

System.out.println("File saved successfully!");
}
}

```

File TransferClient

```

import java.io.BufferedOutputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.net.InetAddress;
import java.net.Socket;

public class FileTransferClient {

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

        //Initialize socket
        Socket socket = new Socket(InetAddress.getByName("localhost"), 5000);
        byte[] contents = new byte[10000];

        //Initialize the FileOutputStream to the output file's full path.
        FileOutputStream fos = new FileOutputStream("d:\\clientfile.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        InputStream is = socket.getInputStream();

        //No of bytes read in one read() call
        int bytesRead = 0;

        while((bytesRead=is.read(contents))!=-1)
            bos.write(contents, 0, bytesRead);

        bos.flush();
        socket.close();
    }
}

```

```

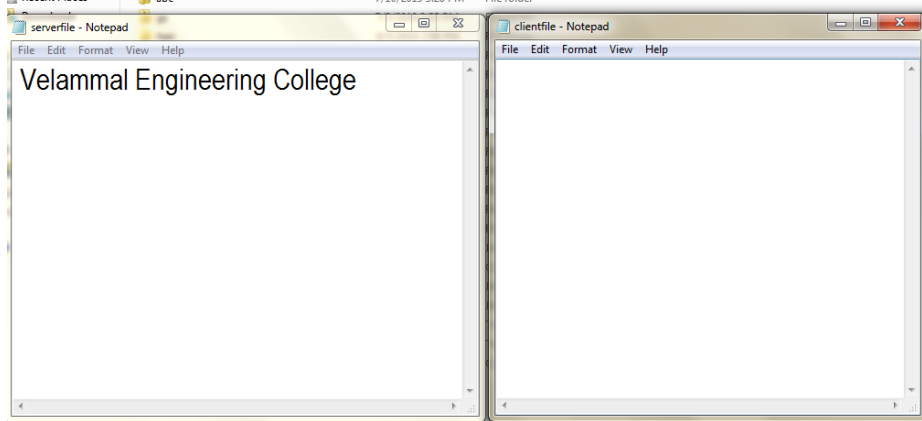
        System.out.println("File saved successfully!");
    }
}

```

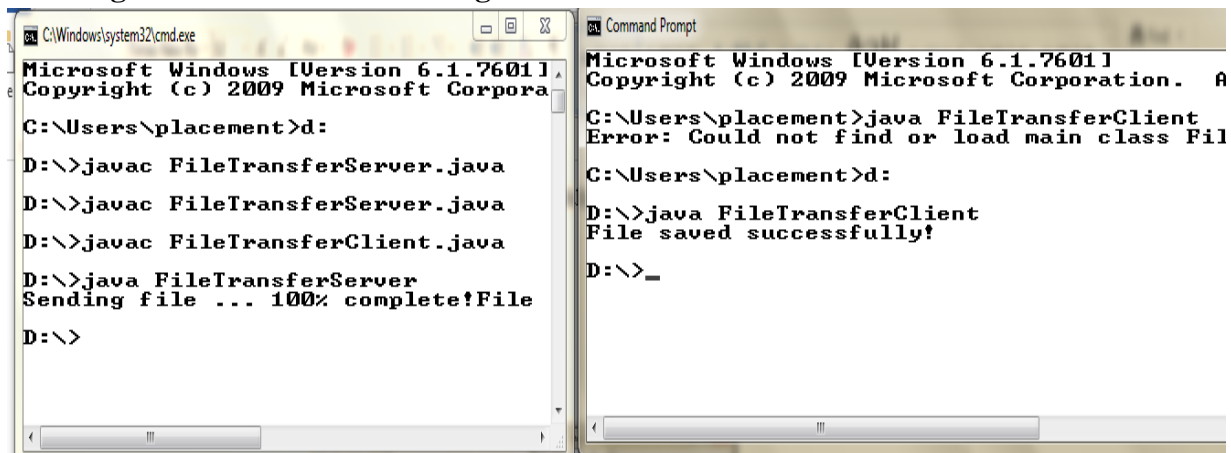
Output

ServerFile and ClientFile – Text files

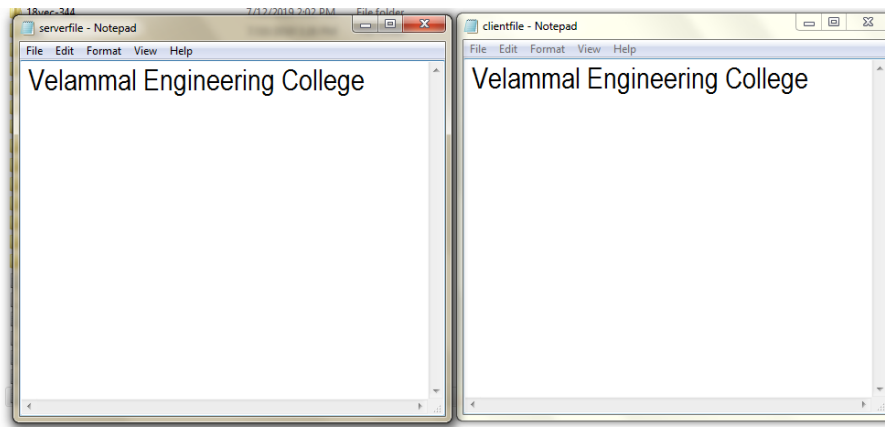
Client file is Blank



Running the file transfer Java Programs



Server File Content Written to Client File



Result

Thus the program to implement File Transfer using TCP/IP in a client and server has been completed successfully

EX NO 4: SIMULATION OF DNS USING UDP SOCKETS

DATE:

Aim

To implement a Java program to simulate DNS using UDP sockets in Java

Algorithm

1. Start the program.
2. Create a class for server.
3. Create an object for the Datagram class and declare an array which acts as buffer.
4. In server()
 1. 4.1 Initialise position.
 2. 4.2 Send character to client using packet if it is not a return or null
 3. 4.3 Increment position
5. In main()
 1. 5.1 Get the data.
 2. 5.2 Create object for Datagram socket.
 3. 5.3 Execute my server.
6. Create a class for client
7. Create object for Datagram socket and declare an array that acts as buffer.
8. In client()
 1. 8.1 Create an object for Datagram packet.
 2. 8.2 Receiver and print character from server.
9. In main()
 1. 9.1 create array for string address.
 2. 9.2 Setup a for loop to print address.
 3. 9.3 Create object for Datagram socket.
 4. 9.4 Execute myclient.
10. Stop the program.

// UDPServer.java

```
import java.io.*;
import java.net.*;
class UDPServer
{
public static DatagramSocket ds;
public static byte buffer[]=new byte[1024]; public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
ds=new DatagramSocket(clientport);
System.out.println("press ctrl+c to quit the program");
```

```

BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
InetAddress ia=InetAddress.getLocalHost();
while(true)
{
DatagramPacket p=new DatagramPacket(buffer,buffer.length);
ds.receive(p);
String psx=new String(p.getData(),0,p.getLength());
System.out.println("Client:" + psx);
InetAddress ib=InetAddress.getByName(psx);
System.out.println("Server output:" + ib);
String str=dis.readLine();
if(str.equals("end")) break;
buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,serverport));
}
}
}

```

// UDPclient.java

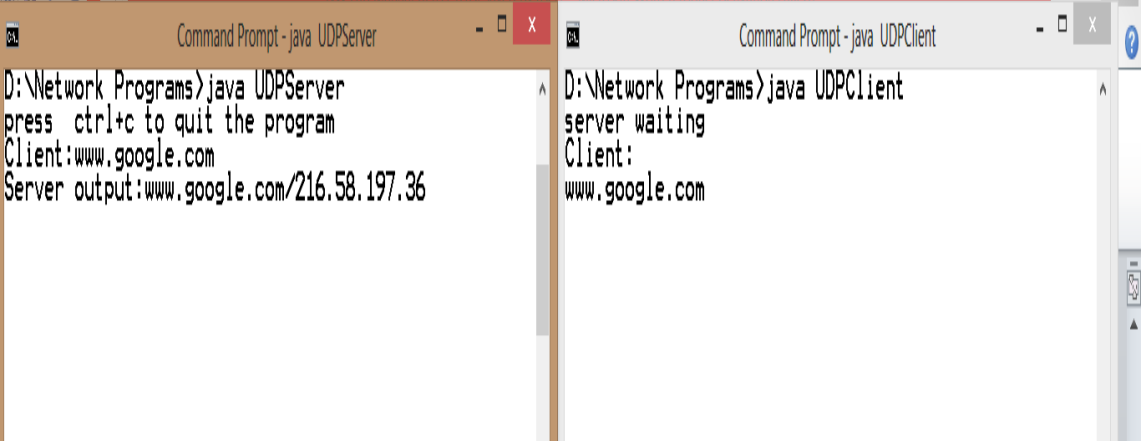
```

import java .io.*;
import java.net.*;
class UDPClient
{
public static DatagramSocket ds;
public static int clientport=789,serverport=790;
public static void main(String args[])throws Exception
{
byte buffer[]=new byte[1024]; ds=new DatagramSocket(serverport);
BufferedReader dis=new BufferedReader(new InputStreamReader(System.in));
System.out.println("server waiting");
InetAddress ia=InetAddress.getLocalHost(); while(true)
{
System.out.println("Client:"); String str=dis.readLine(); if(str.equals("end"))
break; buffer=str.getBytes();
ds.send(new DatagramPacket(buffer,str.length(),ia,clientport)); DatagramPacket p=new
DatagramPacket(buffer,buffer.length); ds.receive(p);

String psx=new String(p.getData(),0,p.getLength()); System.out.println("Server:" + psx);
}
}
}

```

Output



The image shows two side-by-side Windows Command Prompt windows. The left window, titled 'Command Prompt - java UDPServer', displays the following text: 'D:\Network Programs>java UDPServer', 'press ctrl+c to quit the program', 'Client:www.google.com', and 'Server output:www.google.com/216.58.197.36'. The right window, titled 'Command Prompt - java UDPClient', displays: 'D:\Network Programs>java UDPClient', 'server waiting', 'Client:', and 'www.google.com'.

```
Command Prompt - java UDPServer
D:\Network Programs>java UDPServer
press ctrl+c to quit the program
Client:www.google.com
Server output:www.google.com/216.58.197.36

Command Prompt - java UDPClient
D:\Network Programs>java UDPClient
server waiting
Client:
www.google.com
```

Result

Thus the JAVA program to simulate DNS using UDP sockets has been completed successfully

EX NO 5A: WRITE A CODE SIMULATING ARP PROTOCOL

DATE:

Aim

To implement a Java program to simulate ARP protocol

Algorithm

- Step 1: Start the program.
- Step 2: Include the necessary header files.
- Step 3: enter the system name
- Step 4: The IP address is mapped with the MAC address
- Step 5: Print the address details.
- Step 6: Stop the program.

Program

```
import java.net.InetAddress;
import java.io.*;
import java.net.*;
import java.lang.*;
import java.net.NetworkInterface;
import java.net.SocketException;
import java.util.Scanner;
import java.net.UnknownHostException;

public class Arps
{
    public static void main(String args[])
    {
        try
        {
            Scanner console=new Scanner(System.in);
            System.out.println("enter system name");
            String ipaddr=console.nextLine();
            InetAddress address=InetAddress.getByName(ipaddr);
            System.out.println("address="+address);
            NetworkInterface ni=NetworkInterface.getByInetAddress(address);
            if(ni!=null)
            {
                byte[] mac=ni.getHardwareAddress();
                if(mac!=null)
                {
                    System.out.println("MAC address:");
                    for(int i=0;i<mac.length;i++)
                    {
                        System.out.format("%02x%s", mac[i],(i<mac.length-1)? "- ":"");
                    }
                }
            }
        }
    }
}
```




```

    }
    }
else
{
System.out.println("address doesn't exist or is not accessible/");
}
}
else
{
System.out.println("Network Interface for the specified address is not found");
}
}
catch(Exception e)
{
}
}
}
}

```

Output



```

D:\Java\jdk1.6.0\bin\K>javac arp.java
D:\Java\jdk1.6.0\bin\K>java arp
enter system name:local host
address=localhost/165.165.0.1
MAC adress=6A:08:AB:C0

```

Result

Thus the JAVA program to implement ARP simulation is executed successfully and the output is verified.

EX NO 5B: WRITE A CODE SIMULATING RARP PROTOCOL

DATE:

Aim

To write a java program for simulating RARP protocols using UDP

Algorithm

Client

- Step 1. Start the program
- Step 2. Using datagram sockets UDP function is established.
- Step 3. Get the MAC address to be converted into IP address.
- Step 4. Send this MAC address to server.
- Step 5. Server returns the IP address to client.
- Step 6. Stop the program

Server

- Step 1. Start the program.
- Step 2. Server maintains the table in which IP and corresponding MAC addresses are stored.
- Step 3. Read the MAC address which is sent by the client.
- Step 4. Map the IP address with its MAC address and return the IP address to client.
- Step 5. Stop the program

Program

Server Side

```
import java.io.*; import java.net.*; import java.util.*;
class ServerRARP
{
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=new
DatagramPacket(sendbyte,sendbyte.length,addr,port); server.send(sender);
                break;
            }
        }
    }
}
catch(Exception e)
{
    System.out.println(e);
}
}
}

```

Client Side

```

import java.io.*; import java.net.*; import java.util.*;
class ClientRARP

{
    public static void main(String args[])
    {
        try
        {
            DatagramSocket client=new DatagramSocket(); InetAddress
            addr=InetAddress.getByName("127.0.0.1"); byte[] sendbyte=new

            byte[1024];
            byte[] receivebyte=new byte[1024];
            BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
            System.out.println("Enter the Physical address

            (MAC):");
            String str=in.readLine(); sendbyte=str.getBytes();
            DatagramPacket sender=new DatagramPacket(sendbyte,sendbyte.length,addr,1309);
            client.send(sender);

```

```

DatagramPacket receiver=new DatagramPacket(receivebyte,receivebyte.length);
client.receive(receiver);
String s=new String(receiver.getData()); System.out.println("The Logical Address is(IP):
"+s.trim()); client.close();

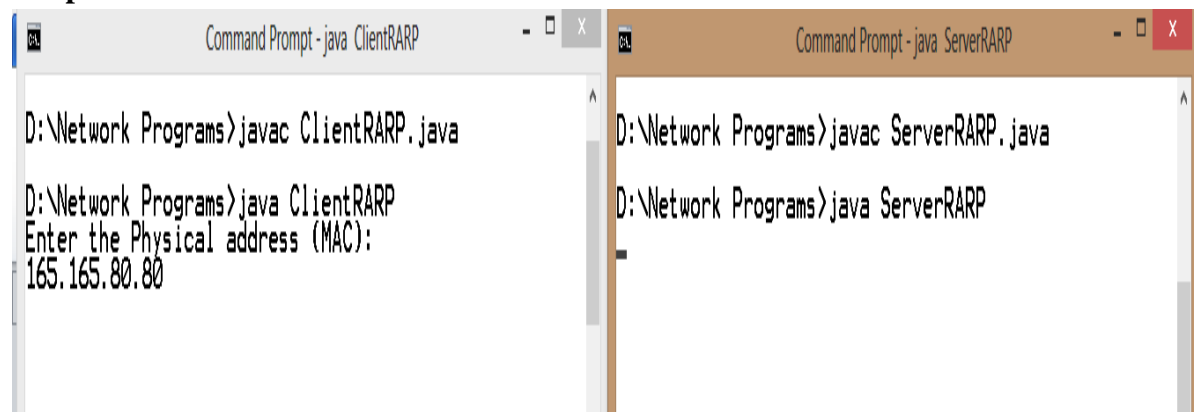
}

catch(Exception e)
{

System.out.println(e);
}
}
}

```

Output



Enter the Physical address (MAC): 6A:08:AA:C2
 The Logical Address is(IP): 165.165.80.80

Result

Thus a java program for simulating RARP protocols using UDP was written and executed successfully.

EX NO 6: SIMULATE STOP AND WAIT PROTOCOL USING SOCKET PROGRAMMING

DATE:

Aim

To implement stop and wait protocol using Socket Programming in JAVA

Algorithm

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

Sender Program

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

public class sender {
    public static void main(String args[])
    {
        int p=9000,i,q=8000;
        String h="localhost";
        try
        {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Enter number of frames : ");
            int number = scanner.nextInt();
            if(number==0)
            {
                System.out.println("No frame is sent");
            }
            else
            {
                Socket s2;
                s2= new Socket(h,q);
                DataOutputStream d1 = new DataOutputStream(s2.getOutputStream());
                d1.write(number);
            }
        }
    }
}
```

```

String str1;
    for (i=0;i<number;i++)
    {
System.out.print("Enter message : ");
String name = scanner.next();
System.out.println("Frame " + i+" is sent");
Socket s1;
    s1= new Socket(h,p+i);
    DataOutputStream d = new DataOutputStream(s1.getOutputStream());
    d.writeUTF(name);
    DataInputStream dd= new DataInputStream(s1.getInputStream());
    Integer sss1 = dd.read();
    System.out.println("Ack for :"+ sss1 + " is received");
    }
}
catch(Exception ex)
    {
        System.out.println("ERROR :"+ex);
    }
}
}

```

RECEIVER program:

```

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

public class receiver {
public static void main(String args[])
{
    String h="Serverhost";
    int q=5000;
    int i;
    try
    {
        ServerSocket ss2;
        ss2 = new ServerSocket(8000);
        Socket s1 =ss2.accept();
        DataInputStream dd1= new DataInputStream(s1.getInputStream());
        Integer i1 =dd1.read();
        for(i=0;i<i1;i++)
        {

```

```

        ServerSocket ss1;
        ss1 = new ServerSocket(9000+i);
        Socket s =ss1.accept();
        DataInputStream dd= new DataInputStream(s.getInputStream());
        String sss1 = dd.readUTF();
        System.out.println(sss1);
        System.out.println("Frame "+ i+" received");
        DataOutputStream d1 = new DataOutputStream(s.getOutputStream());
        d1.write(i);
        System.out.println("ACK sent for "+ i);
    }
}
catch(Exception ex)
{
    System.out.println("Error"+ex);
}
}
}

```

Output

//RECEIVER OUTPUT:

```

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

C:\Users\Placement>cd desktop
C:\Users\Placement\Desktop>javac receiver.java
C:\Users\Placement\Desktop>java receiver
subject
Frame 0 received
ACK sent for 0
computer
Frame 1 received
ACK sent for 1
networks
Frame 2 received
ACK sent for 2
C:\Users\Placement\Desktop>_

```

// SENDER OUTPUT:

```

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

C:\Users\Placement>cd desktop
C:\Users\Placement\Desktop>javac sender.java
C:\Users\Placement\Desktop>java sender
Enter number of frames : 3
Enter message : subject
Frame 0 is sent
Ack for :0 is received
Enter message : computer
Frame 1 is sent
Ack for :1 is received
Enter message : networks
Frame 2 is sent
Ack for :2 is received
C:\Users\Placement\Desktop>_

```

RESULT:

Thus a java program for implementing Stop and Wait protocol using socket programming was written and executed successfully.

EX NO 7: IMPLEMENTATION OF SUBNETTING

DATE:

AIM:

Write a program to implement subnetting and find the subnet masks.

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.util.Scanner;
class Subnet
{
public static void main(String args[])
{
Scanner sc = new Scanner(System.in);
System.out.print("Enter the ip address");
String ip = sc.nextLine();
String split_ip[]=ip.split("\\."); //Split the string after every '.'
String split_bip[] = new String[4]; //split binary ip
String bip = ""; for(int i=0;i<4;i++){
split_bip[i] = appendZeros(Integer.toBinaryString(Integer.parseInt(split_ip[i])));
// "18" => 18 => 10010=>00010010
bip += split_bip[i];
}
System.out.println("IP in binary is "+bip); System.out.print("Enter the number of addresses: ");
int n = sc.nextInt();
//Calculation of mask
int bits = (int)Math.ceil(Math.log(n)/Math.log(2));
/*eg if address = 120, log 120/log 2 gives log to the base 2 => 6.9068, ceil gives us upper integer
*/
System.out.println("Number of bits required for address = "+bits); int mask = 32-bits;
System.out.println("The subnet mask is = "+mask);
//Calculation of first address and last
address int fbip[] = new int[32];
for(int i=0; i<32;i++) fbip[i] = (int)bip.charAt(i)-48;
//convert character 0,1 to integer 0,1
for(int i=31;i>31-bits;i-)
//Get first address by ANDing last n bits with 0
fbip[i] &= 0;
String fip[] = {"", "", "", ""};
for(int i=0;i<32;i++)
fip[i/8] = new String(fip[i/8]+fbip[i]);
System.out.print("First address is = ");
```



```

for(int i=0;i<4;i++){ System.out.print(Integer.parseInt(fip[i],2)); if(i!=3) System.out.print(".");
}
System.out.println();
int lbip[] = new int[32];
for(int i=0; i<32;i++) lbip[i] = (int)bip.charAt(i)-48; //convert character 0,1 to integer 0,1 for(int
i=31;i>31-bits;i--) //Get last address by ORing last n bits with 1
lbip[i] |= 1;
String lip[] = {"", "", "", ""};
for(int i=0;i<32;i++)
lip[i/8] = new String(lip[i/8]+lbip[i]); System.out.print("Last address is =");
for(int i=0;i<4;i++){ System.out.print(Integer.parseInt(lip[i],2); if(i!=3) System.out.print(".");
}
System.out.println();
}
static String appendZeros(String s){
String temp = new String("00000000");
return temp.substring(s.length()+ s;
}
}

```

OUTPUT:

Enter the ip address: 100.110.150.10
IP in binary is 01100100011011101001011000001010
Enter the number of addresses: 7
Number of bits required for address = 3
The subnet mask is = 29
First address is = 100.110.150.8
Last address is = 100.110.150.15

RESULT:

Thus a program to implement subnetting and find the subnet masks was written and implemented successfully.

EX NO 8: SNMP - Simple Network Management Protocol

DATE:

AIM

To write a java program for SNMP application program

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.IOException;
import org.snmp4j.CommunityTarget;
import org.snmp4j.PDU;
import org.snmp4j.Snmp;
import org.snmp4j.Target;
import org.snmp4j.TransportMapping;
import org.snmp4j.event.ResponseEvent;
import org.snmp4j.mp.SnmpConstants;
import org.snmp4j.smi.Address;
import org.snmp4j.smi.GenericAddress;
import org.snmp4j.smi.OID;
import org.snmp4j.smi.OctetString;
import org.snmp4j.smi.VariableBinding;
import org.snmp4j.transport.DefaultUdpTransportMapping;

public class SNMPPManager
{
    Snmp snmp = null;
    String address = null;
    public SNMPPManager(String add){
        address = add;
    }
    public static void main(String[] args) throws IOException {
        SNMPPManager client = new SNMPPManager("udp:127.0.0.1/161");
        client.start();
        String sysDescr = client.getAsString(new OID(".1.3.6.1.2.1.1.1.0"));
        System.out.println(sysDescr);
    }
    private void start() throws IOException {
        TransportMapping transport = new DefaultUdpTransportMapping();
        snmp = new Snmp(transport);
    }
    public String getAsString(OID oid) throws IOException {
        ResponseEvent event = get(new OID[] { oid });
        return event.getResponse().get(0).getVariable().toString();
    }
}
```

```

public ResponseEvent get(OID oids[]) throws IOException {
    PDU pdu = new PDU();
    for (OID oid : oids)
    {
        pdu.add(new VariableBinding(oid));
    }
    pdu.setType(PDU.GET);
    ResponseEvent event = snmp.send(pdu, getTarget(), null);
    if(event != null)
    {
        return event;
    }
    throw new RuntimeException("GET timed out");
}

private Target getTarget() {
    Address targetAddress = GenericAddress.parse(address);
    CommunityTarget target = new CommunityTarget();
    target.setCommunity(new OctetString("public"));
    target.setAddress(targetAddress);
    target.setRetries(2);
    target.setTimeout(1500);
    target.setVersion(SnmpConstants.version2c);
    return target;
}
}

```

OUTPUT:

Hardware: x86 Family 6 Model 23 Stepping 10 AT/AT COMPATIBLE – Software: Windows 2000 Version 5.1 (Build 2600 Multiprocessor Free)

RESULT:

Thus a java program for SNMP application program was written and executed successfully

EX NO 9: FILE TRANSFER USING UDP

DATE:

AIM:

To write a java program for file transfer application using UDP Sockets

ALGORITHM:

1. Start the program.
2. Include the necessary header files.
3. Connection must be made between server and client
4. The server writes the necessary information in the file and that particular file name can be accessed by the client
5. The client then reads the file and sends an acknowledgement that it has received the file
6. The connection is then terminated.
7. Stop

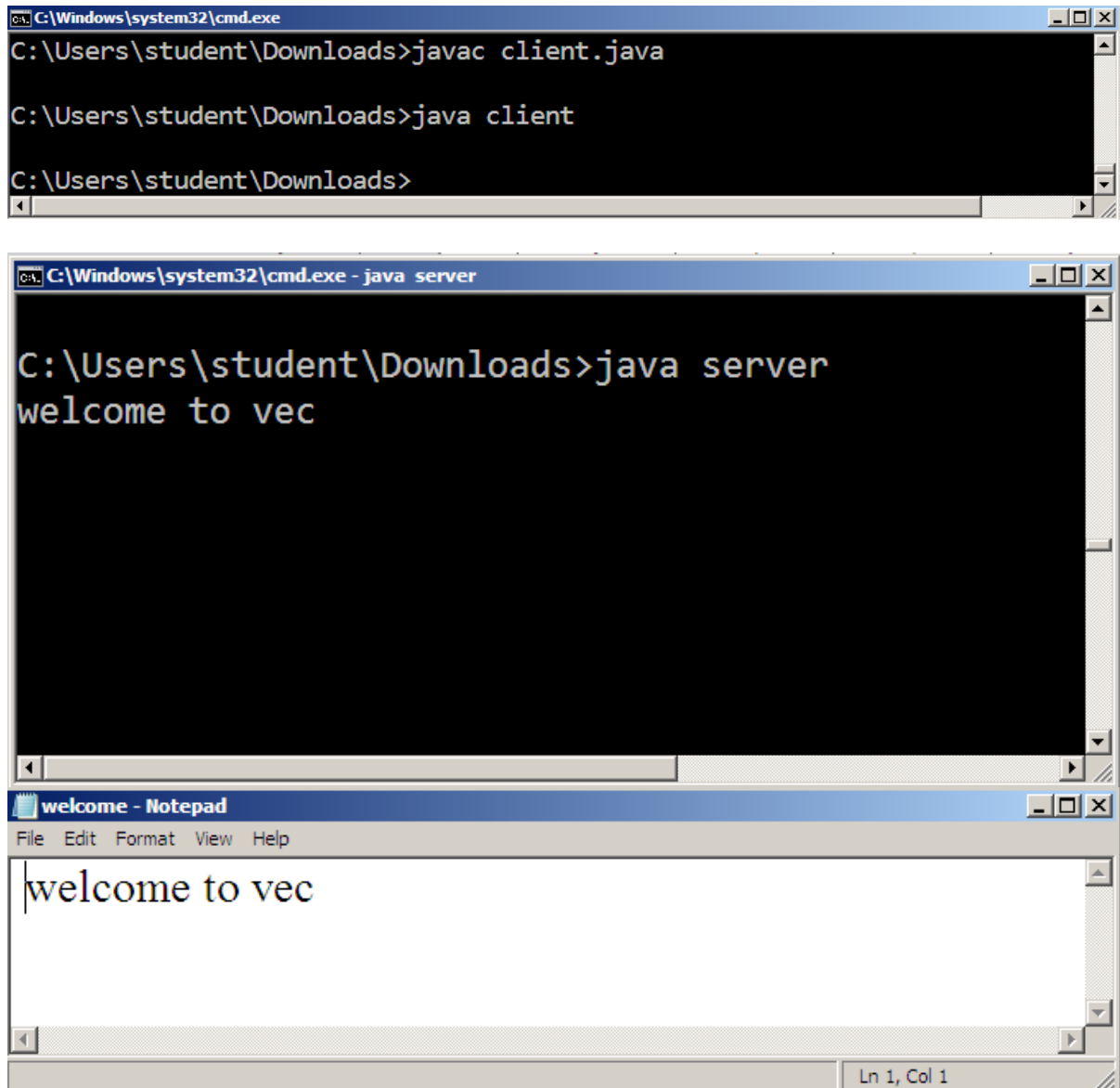
PROGRAM:

```
//server.java
import java.net.*;
import java.io.*;
public class server
{
    public static void main(String args[])throws IOException
    {
        byte b[]=new byte[3072];
        DatagramSocket dsoc=new DatagramSocket(1000);
        FileOutputStream f=new FileOutputStream("hello.txt");
        while(true)
        {
            DatagramPacket dp=new DatagramPacket(b,b.length);
            dsoc.receive(dp);
            System.out.println(new String(dp.getData(),0,dp.getLength()));
        }
    }
}

//client.java
import java.net.*;
import java.io.*;
public class client
{
    public static void main(String args[])throws Exception
    {
        byte b[]=new byte[1024];
        FileInputStream f=new FileInputStream("welcome.txt");
        DatagramSocket dsoc=new DatagramSocket(2000);
        int i=0;
        while(f.available()!=0)
        {
            b[i]=(byte)f.read();
            i++;
        }
    }
}
```

```
f.close();  
dsoc.send(new DatagramPacket(b,i,InetAddress.getLocalHost(),1000));  
}  
}
```

OUTPUT:



RESULT :

Thus a file is transferred successfully using UDP sockets successfully.

EX NO 9: Network Simulator (NS2).

DATE

AIM:

To Study the Network Simulator (NS2).\

STUDY:

The Network Simulator version 2 (NS-2) is a deterministic discrete event network simulator, initiated at the Lawrence Berkeley National Laboratory (LBNL) through the DARPA funded Virtual Internetwork Test bed (VINT) project. The VINT project is collaboration between the Information Sciences Institute (ISI) at the University of Southern California (USC), Xerox's Palo Alto Research Centre (Xerox PARC), University of California at Berkeley (UCB) and LBNL. NS-2 was initially created in 1989 as an alternative to the REAL Network Simulator. Since then there is significant growth in uses and width of NS project. Although there are several different network simulators available today, ns-2 is one of the most common. NS-2 differs from most of the others by being open source software, supplying the source code for free to anyone that wants it. Whereas most commercial network simulators will offer support and a guarantee but keeping the moneymaking source code for themselves.

The Structure of NS2

NS-2 is made up of hundreds of smaller programs, separated to help the user sort through and find what he or she is looking for. Every separate protocol, as well as variations of the same, sometimes has separate files. Though some are simple, but still dependent on the parental class.

C++

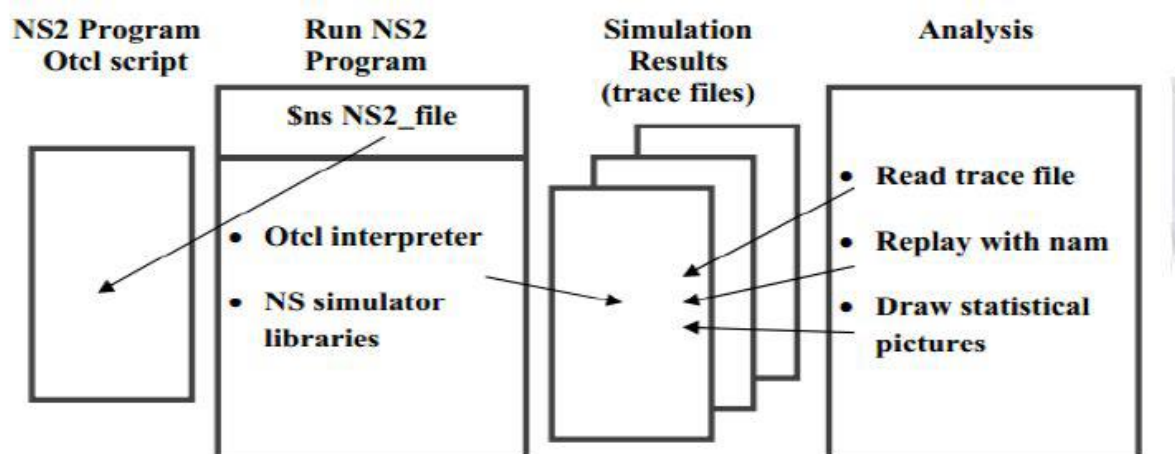


Figure A. The basic Structure of NS2

C++ is the predominant programming language in ns-2. It is the language used for all the small programs that make up the ns-2 hierarchy. C++, being one of the most common

programming languages and specially designed for object- oriented coding, was therefore a logical choice what language to be used. This helps when the user wants to either understand the code or do some alterations to the code. There are several books about C++ and hundreds, if not thousands, of pages on the Internet about C++ simplifying the search for help or answers concerning the ns-2 code.

OTcl

Object Tcl (OTcl) is object-oriented version of the command and syntax driven programming language Tool Command Language (Tcl). This is the second of the two programming languages that NS-2 uses. The front-end interpreter in NS-2 is OTcl which link the script type language of Tcl to the C++ backbone of NS-2. Together these two different languages create a script controlled C++ environment. This helps when creating a simulation, simply writing a script that will be carried out when running the simulation. These scripts will be the formula for a simulation and is needed for setting the specifications of the simulation itself. Without a script properly defining a network topology as well as the data-rows, both type and location, nothing will happen.

Nodes

A node is exactly what it sounds like, a node in the network. A node can be either an end connection or an intermediate point in the network. All agents and links must be connected to a node to work. There are also different kinds of nodes based on the kind of network that is to be simulated. The main types are node and mobile node, where node is used in most wired networks and the mobile node for wireless networks. There are several different commands for setting the node protocols to be used, for instance what kind of routing is to be used or if there is a desire to specify a route that differs from the shortest one. Most of the commands for node and mobile node can be easily found in the ns documentation. Nodes and the closely connected link creating commands, like simplex link and duplex link, could be considered to simulate the behavior of both the Link Layer.

Agents

An agent is the collective name for most of the protocols you can find in the transport layer. In the ns-2 documentation agents are defined as the endpoints where packets are created and consumed. All the agents defined in ns-2, like tcp, udp etc., are all connected to their parent class, simply called Agent. This is where their general behavior is set and the offspring classes are mostly based on some alterations to the inherent functions in the parent class. The modified functions will overwrite the old and thereby change the performance in order to simulate the wanted protocol.

Applications

The applications in ns-2 are related to the Application Layer in the TCP/IP suite. The hierarchy here works in the similar way as in the agent's case. To simulate some of the most important higher functions in network communication, the ns-2 applications are used. Since the purpose of ns-2 is not to simulate software, the applications only represent some different aspects of the higher functions. Only a few of the higher layer protocols has been implemented, since some are quite similar when it comes to using the lower functions of the TCP/IP stack. For instance there is no use adding both a SMTP and a HTTP application since they both use TCP to transfer small amounts of data in a similar way. The only applications incorporated in the release version of ns-2 are a number of different traffic generators for use with UDP and telnet and FTP for using TCP. All the applications are script controlled and when concerning the traffic generators, you set the interval and packet-size of the traffic. FTP can be requested to send a data packet whenever the user wants to, or to start a transfer of a file of arbitrary size. If starting an FTP transmission and not setting a file-size the transmission will go on until someone calls a stop.

NAM

The Network Animator NAM is a graphic tool to use with ns-2. It requires a namtracefile recorded during the simulation and will then show a visual representation of the simulation. This will give the user the possibility to view the traffic packet by packet as they move along the different links in the network. NAM offers the possibility of tracing a single packet during its travel and the possibility to move the nodes around for a user to draw up his network topology according to his own wishes.

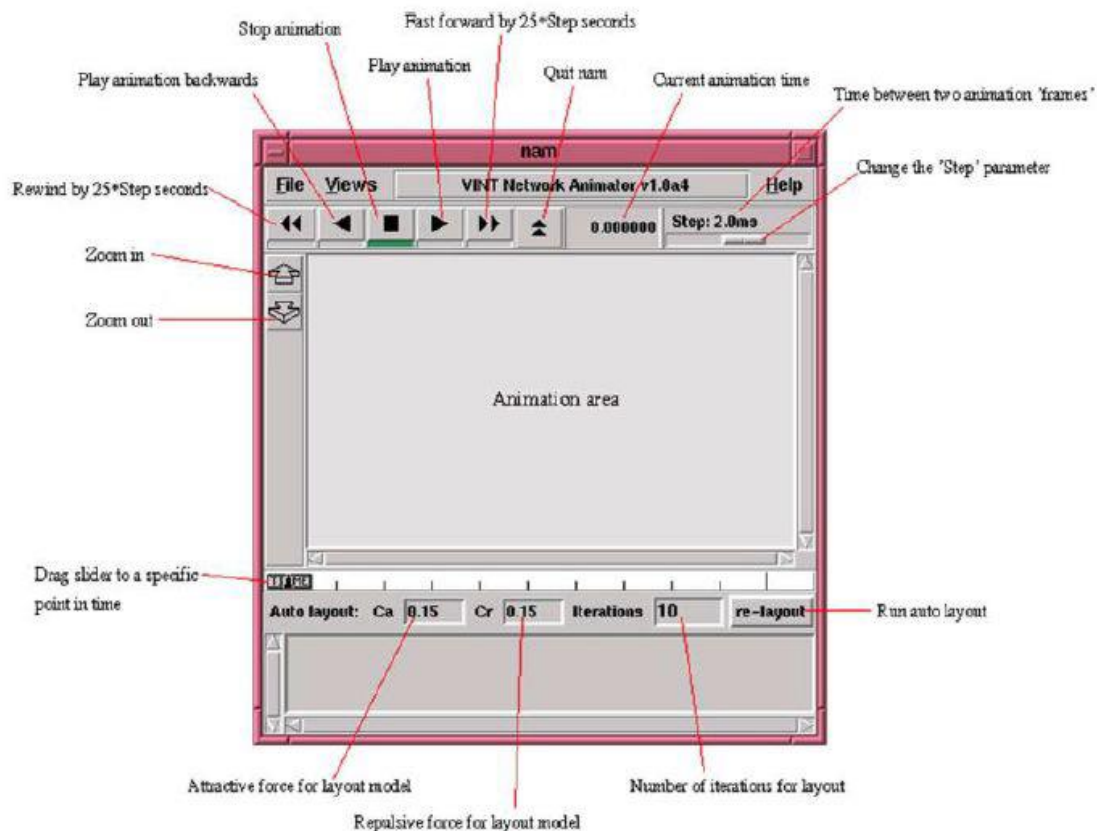


Figure of Ns2 visualization tool

Since the simulation has already been performed there is no possibility for the user to change the links or any other aspect of the simulation except the representation. The existence of an X-server allows NAM to be able to open a graphical window. Therefore if NAM is to work, there must be a version of X-server running

RESULT:

Thus the basics of NS2 simulator were studied.

EX NO 9: STUDY ON ROUTING PROTOCOL (RIP)

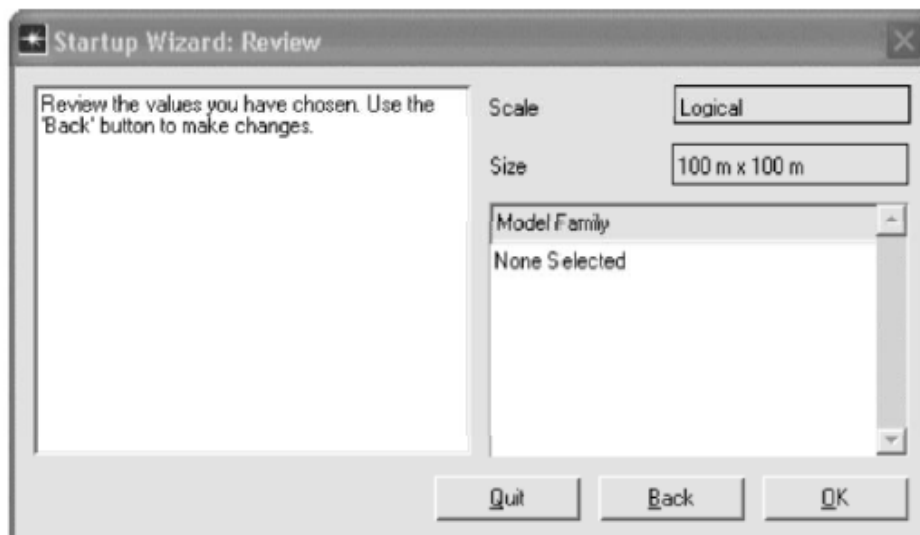
DATE

AIM:

To simulate the behavior of several routers running the RIP routing protocol and to learn how to use routing tables to find paths in a network.

EXPLANATION:

The Routing Information Protocol (RIP) is an example of a dynamic distance vector routing algorithm. This protocol chooses routes within a network that are of minimum distance. Routers adapt to changes in network topology (link or router failures) by exchanging information with their directly connected neighbors. Under normal conditions, routing table information is exchanged periodically (typically every 30 seconds). When changes in topology occur, however, the *triggered update* mechanism comes into play. If a router receives new routing information, it will wait 1-5 seconds (randomly determined) before passing that information on to its neighbors, rather than waiting the full 30 seconds. The triggered update mechanism causes topology information to propagate through the network much faster than it would using the standard timeout.

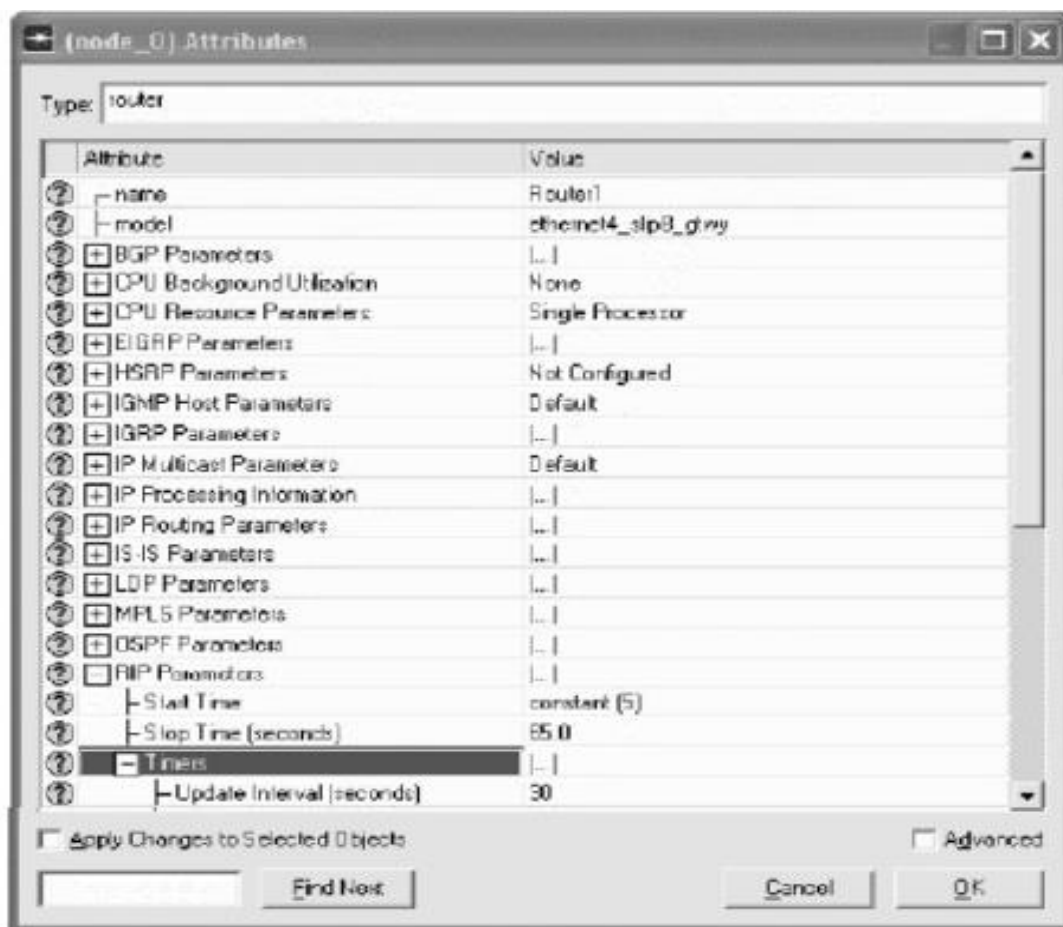


Steps in Building the Simulation

Step 1 – Set Parameters for Simulation Model

- Start up OPNET IT Guru Academic Edition.
- Select the **File** menu => **New...**

- Choose **Project** and click on **OK**.
- Change the **Project Name** to **xx_RIP_Network** (where **xx** are your initials) and click on **OK**.
- In the **Initial Topology** window, select **Create Empty Scenario** and click on **Next**.
- In the **Choose Network Scale** window, select **Logical** and click on **Next**.
- In the **Select Technologies** window, click on **Next**.
- In the **Review** window, click on **OK**.

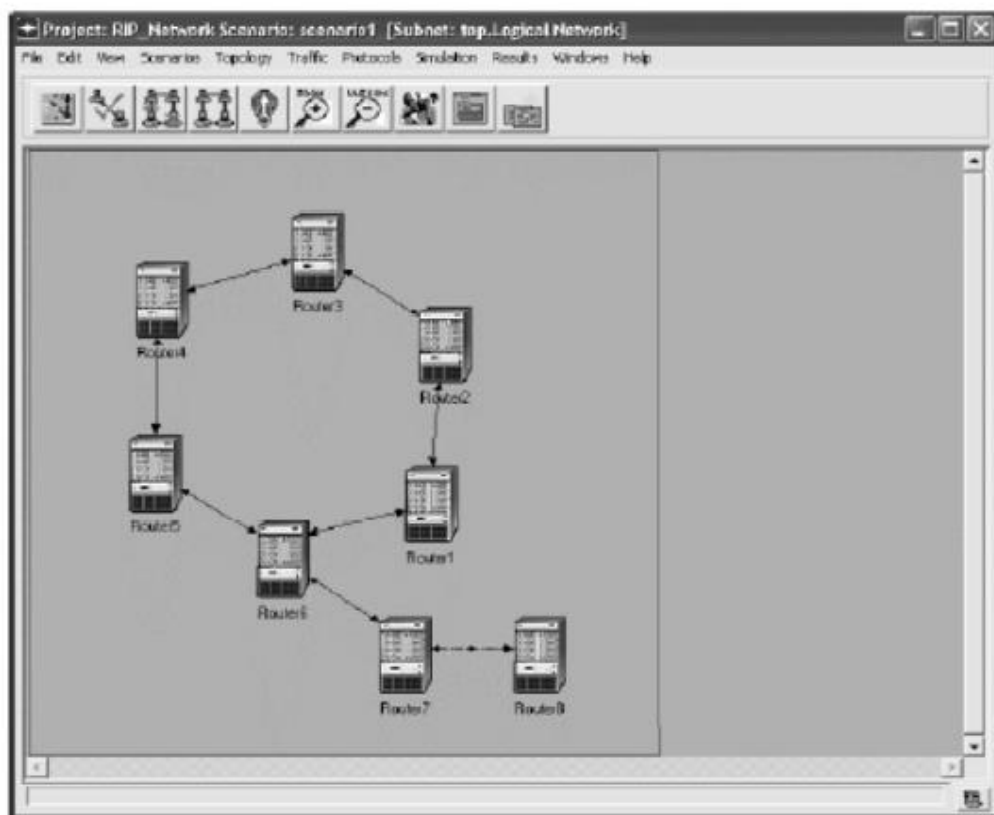


Step 2 - Building a small network of routers and setting up the RIP parameters

- Select an **ethernet4_slip8_gtwy** object from the Object Palette and place it in the project workspace. Right click on the object and choose **View Node Description**. Note that this gateway is equipped with four Ethernet interfaces and eight SLIP interfaces.
- Click on the close window icon to close the window.
- Right click on the router, choose **Edit Attributes**, and set the **name** attribute to **Router1**.
- Expand the **RIP Parameters** attribute and the **Timers** attribute. Note that the **Update**

Interval (seconds) is set to 30 seconds. This means that a router will exchange its routing table with each of its neighbors every 30 seconds, even if new information is not learned.

- Click on **OK** to close the window.
- Left click on the router and choose **Copy** from the **Edit** menu at the top of the window.
- Choose **Paste** from the **Edit** menu five times to generate five copies of the first router in the project workspace.
- Arrange them in a circle and then connect them together into a ring using six **PPP_DS1** lines from the Object Palette.
- Each router should be connected to two other routers.
- **Paste** two more routers on the map.
- Using 2 more **PPP_DS1** lines, make a tail off of Router6. Your final network should look like a ring of six routers with a tail of two routers:
- Select the **Protocols** menu => **IP** => **Routing** => **Configure Routing Protocols...**
Check the boxes next to **RIP**,



- Apply the above selection to subinterfaces, and **Visualize Routing Domains**.

- Select the radio button next to **All interfaces (including loopback)**.
- Click on **OK** to close the window. By doing so, you have ensured that the RIP routing protocol will be used to route packets across all the interfaces on all the routers. Note that the legend shows that RIP is used on all links.

Step 3 - Configuring one of the PPP links to fail part of the way through the simulation.

- Select **utilities** from the pull-down menu in the Object Palette in order to display all utility objects.
- Place a **Failure Recovery** object in the project workspace.
- Right click on the failure object and **Edit Attributes**.
- Set the **name** to **Link Failure**.
- Expand the **Link Failure/Recovery Specification** attribute and set **rows** to **1**.
- Expand **row 0**, and set the **Name** to **Logical Network.Router 1<->Router 6**.
- Set the **Time** to **300**. Note that the **Status** is set to **Fail**. This failure node will cause the link between router 1 and router 6 to fail 300 seconds into the simulation.
- Click on **OK** to close the window.

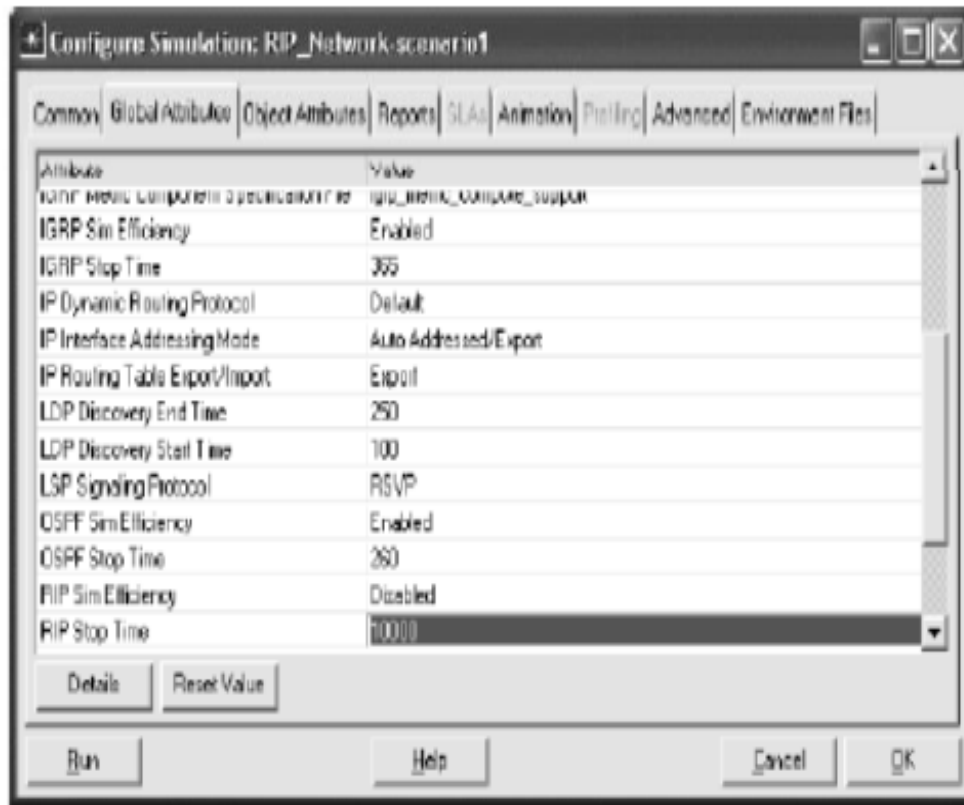
Step 4 - Set Simulation Statistics

- Select the **Simulation** tab => **Choose Individual Statistics...**
- Expand the **Global Statistics** item, and the **RIP** item and select the **Traffic Received (bits/sec)** statistic. Right click on the statistic and choose **Change Collection Mode**.
- Check the box next to **Advanced**, and change the **Capture Mode** to **all values**. This will ensure that OPNET will provide a more detailed graph.
- Click on **OK** twice to close the windows.

Step 5 – Configure Simulation

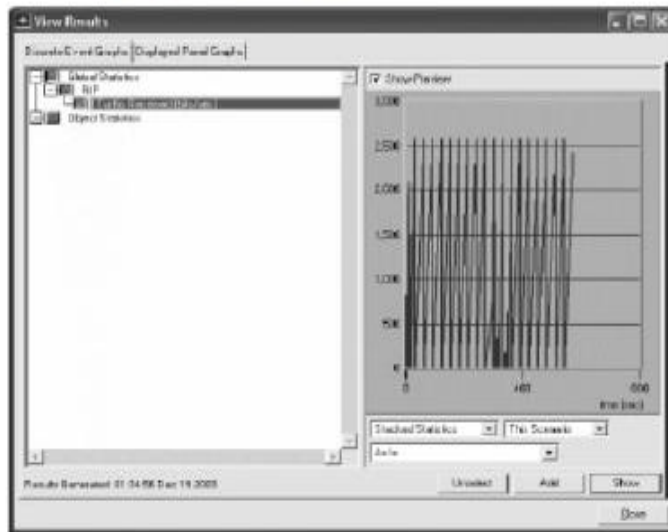
- Select **Simulation** => **Configure Discrete Event Simulation...**
- Under the **Common** tab, set the **Duration** to **10**, and the unit to **minute(s)**.
- Click on the **Global Attributes** tab.
- Set the **IP Interface Addressing Mode** to **Auto Address/Export**.
- Set the **IP Routing Table Export/Import** field to **Export**. This will cause the routing tables of all the routers to be written to a text file when the simulation completes.
- Set the **RIP Sim Efficiency** to **disabled**.
- Set the **RIP Stop Time** to **10000**. This will ensure that the routers keep exchanging routing information even when their routing tables have stabilized.

- Click on **Run** to run the simulation. When the simulation has completed, click on **Close** to close the window.



Step 6 - Inspect and Analyze Results

- Select **Results => View Results...**
- Select and expand the **Global Statistics** item and the **RIP** item, and select the **Traffic Received (bits/sec)** statistic. This statistic shows how much traffic was generated by the routing algorithm.
- Click on **Show** to generate a separate window for the statistics graph. You should see nice, regular peaks in routing protocol traffic every 30 seconds except for two periods, one at the beginning of the simulation and one half way through.
- Select the first 2 minutes of the graph to zoom in. You should now be able to see that many irregularly timed updates were done very quickly at the beginning of the simulation. These updates are due to the triggered update mechanism in RIP. The other period of irregular updates is due to the link failure that occurred after 300 seconds.
- Click on the close window icon and **Delete** the panel.
- Click on **Close** to close the **View Results** window.



Step 7 – Duplicating Scenarios

- Select **Scenarios => Duplicate Scenario...** Accept the default scenario name of **scenario2** by clicking on the **OK** button.
- Left click on the Failure node to select it and choose **Edit => Cut** to remove the Failure node.
- Choose **Simulation => Run Discrete Event Simulation** to generate results for this scenario in which no link failures occur. When the simulation has completed, click on **Close** to close the window.
- Select **File => Model Files => Refresh Model Directories**. Select **File => Open...**
- Choose to open a **Generic Data File**, and from the list displayed, choose **xx_RIP_Network-scenario2-ip_addresses**. This file shows the IP addresses and subnetwork masks which have been assigned to each interface by OPNET. Note that each router has one interface called the *loopback* interface that is not attached to any link. These logical interfaces are commonly used for testing or to allow two socket programs to communicate even if they are running on the same machine. Note that the figure below shows only part of the entire file.
- Click on the close window icon to close the addresses window.

Now we will inspect the routing tables of the nodes.

- Select **File => Open...**
- Choose to open a **Generic Data File** again, and from the list displayed, choose **xx_RIP_Network-scenario2-ip_routes**. This file is a dump of the routing tables maintained at each router in the state they were in at the end of the simulation run. There

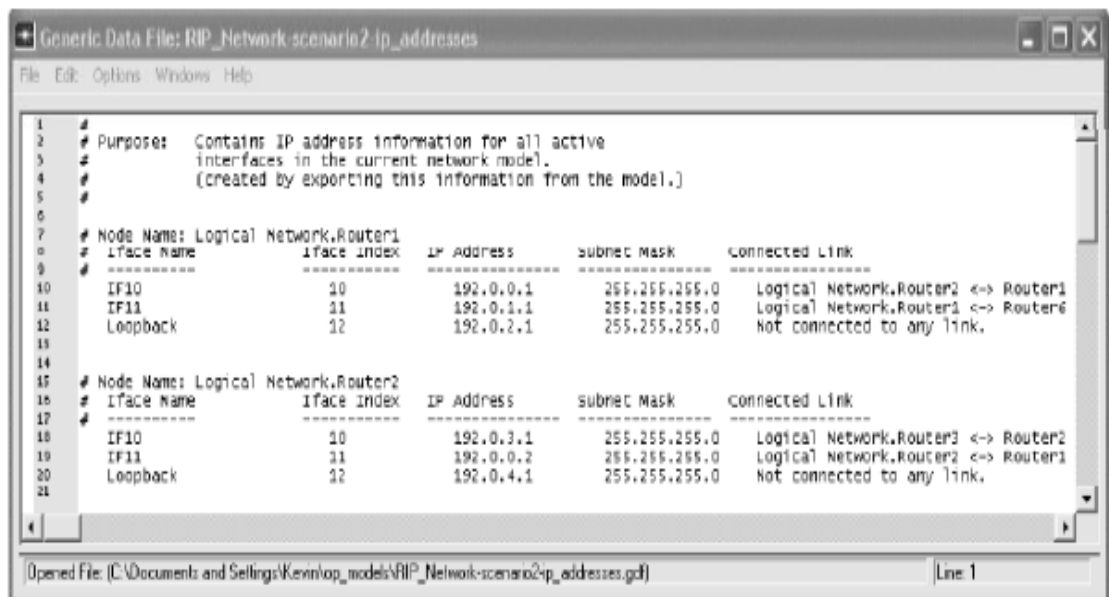
are two parts to the file. In the first part, the directly connected networks for each router are listed. A network is directly connected to a router if the router has an interface which matches that network address up to the length of the subnetwork mask. For example, Router1 has an interface of 192.0.2.1 with a subnetwork mask of 255.255.255.0. A subnetwork mask of 255.255.255.0 requires that the first 24 bits of the IP address match the network address. So the 192.0.2.0 network matches the interface address of 192.0.2.1 and Router1 is directly connected to the 192.0.2.0 network.

- The second part of the `ip_routes` file shows routes to networks which are *not* directly connected. Note that the next figure below shows just the part of the routing table file which applies to Router1. Your file will include one routing table for each router. Note that your address assignments may be different, depending on the order in which you placed the routers in the workspace.
- Using the `ip_routes` file, we can determine the path that a packet would take through our network. Let's say that we would like to find the path from **Router1** to **Router8**. Using the `xx_RIP_Network-scenario1-ip_addresses` file, let's pick an interface on each router (other than the loopback interfaces).

Router1: 192.0.1.1

Router8: 192.0.13.2

- 192.0.13.2 is on the 192.0.13.0 network so that is what we'll be looking for in the routing tables. Note that, in order to reduce the size of the routing tables, routing is done based on network addresses rather than individual IP addresses.
- Starting with Router1's routing table, we look for the destination network (**Dest Network** field in the table), 192.0.13.0, and pull out the **Next Hop Addr** field. In this case, the **Next Hop Addr** is 192.0.1.2. We now look at the **Interface Information** portion of this file or in the `xx_RIP_Network-scenario1-ip_addresses` file to see which router has an interface with this address. In this case, it is Router6.
- We then repeat our steps, first looking for the destination network, 192.0.13.0, in Router6's routing table, pulling out the next hop address (192.0.11.2), and finding the router that owns this address (Router7). We continue until we reach a router that is directly connected to the destination network. Router 7 has an interface (192.0.13.1) on the destination network, meaning that it has a direct connection to Router8 (192.0.13.2).
- You can see that Router7 and Router8 are on the same network because their IP addresses are the same up to the length of the subnetwork mask (24 bits).



- In our example, the path is Router1 (192.0.1.1) => Router6 (192.0.1.2) => Router7 (192.0.11.2) => Router8 (192.0.13.2) Success!
- Click on the close window icon to close the **ip_routes** window. You may also access the routing table dump file directly. Using Windows Explorer, look in your home directory for a directory called **op_models**.
- Change to this directory and look for the generic data file called **xx_RIP_Networkscenario1-ip_routes.gdf**. This is the file you just inspected from within OPNET.
- If you have trouble finding the directory, select **Edit => Preferences**, and look for the **mod_dirs** attribute. The associated value is the path to your **op_models** directory.
- Save your model and close all windows.

```

Generic Data File: RIP_Network-scenario2-ip_addresses
File Edit Options Windows Help

1 #
2 # Purpose: Contains IP address information for all active
3 #           interfaces in the current network model.
4 #           (created by exporting this information from the model.)
5 #
6 #
7 # Node Name: Logical Network.Router1
8 # Iface Name      Iface Index  IP Address      Subnet Mask      Connected Link
9 # -----
10 # IF10            10          192.0.0.1        255.255.255.0    Logical Network.Router2 <-> Router1
11 # IF11            11          192.0.1.1        255.255.255.0    Logical Network.Router1 <-> Router6
12 # Loopback        12          192.0.2.1        255.255.255.0    Not connected to any link.
13 #
14 #
15 # Node Name: Logical Network.Router2
16 # Iface Name      Iface Index  IP Address      Subnet Mask      Connected Link
17 # -----
18 # IF10            10          192.0.3.1        255.255.255.0    Logical Network.Router3 <-> Router2
19 # IF11            11          192.0.0.2        255.255.255.0    Logical Network.Router2 <-> Router1
20 # Loopback        12          192.0.4.1        255.255.255.0    Not connected to any link.
21 #

Opened File: [C:\Documents and Settings\Kevin\op_models\RIP_Network-scenario2-ip_addresses.gdl] Line: 1

```

```

Generic Data File: RIP_Network-scenario2-ip_routes
File Edit Options Windows Help

134 START_ROUTING_TABLE
135 #Module OBJECT 10, Table SIZE, NUMBER OF INTERFACES, IS STATIC
136 128,13,3,0
137 #Module Hierarchical Name:
138 Logical Network.Router1.rip
139 #Interface Information: Interface, IP Address, Mask
140 10,-1,192.0.0.0,255.255.255.0
141 11,-1,192.0.1.0,255.255.255.0
142 12,-1,192.0.2.0,255.255.255.0
143 #RIP ROUTING TABLE CONTENTS:
144 #
145 # Int. Addr., Dest. Network, Dest. Net Mask, Metric, Next Hop Addr., Route Tag, Permanent
146 # -----
147 3,192.0.3.0,Invalid,1,192.0.0.2,0,0
148 4,192.0.4.0,Invalid,1,192.0.0.2,0,0
149 5,192.0.5.0,Invalid,1,192.0.1.2,0,0
150 11,192.0.11.0,Invalid,1,192.0.1.2,0,0
151 12,192.0.12.0,Invalid,1,192.0.1.2,0,0
152 7,192.0.7.0,Invalid,2,192.0.1.2,0,0
153 8,192.0.8.0,Invalid,3,192.0.1.2,0,0
154 10,192.0.10.0,Invalid,2,192.0.1.2,0,0
155 13,192.0.13.0,Invalid,2,192.0.1.2,0,0
156 14,192.0.14.0,Invalid,2,192.0.1.2,0,0
157 15,192.0.15.0,Invalid,3,192.0.1.2,0,0
158 5,192.0.5.0,Invalid,2,192.0.0.2,0,0
159 6,192.0.6.0,Invalid,2,192.0.0.2,0,0
160 END_ROUTING_TABLE

```

RESULT:

A study on Routing Information Protocol (RIP) has been made.

STUDY OF SOCKET PROGRAMMING IN JAVA

DATE:

AIM:

To study about the basics of Socket in Network Programming

NETWORK PROGRAMMING

Network programming involves writing programs that communicate with other programs across a computer N/W. One program is normally called the client, and the other the server. Common examples in the TCP/IP are web clients (browsers) & Web Servers, FTP clients & server and Telnet clients & servers. To facilitate communication between unrelated processes,

and to standardize network programming, an API is needed. There are two such APIs: 1.

Sockets, sometimes called „Berkeley Sockets“ 2. XTI (X/open transport interface)

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols:

TCP: TCP stands for Transmission Control Protocol, which allows for reliable communication

between two applications. TCP is typically used over the Internet Protocol, which is referred to

as TCP/IP.

UDP: UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications

Socket

In TCP/IP, an addressable point that consists of an IP address and a TCP or UDP port member

that provides application with access to TCP/IP protocol is called Socket. A socket is an abstraction that represents an endpoint of communication.

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to

a server.

When the connection is made, the server creates a socket object on its end of the communication.

The client and server can now communicate by writing to and reading from the socket.

The `java.net.Socket` class represents a socket, and the `java.net.ServerSocket` class provides a mechanism for the server program to listen for clients and establish connections with them.

The following steps occur when establishing a TCP connection between two computers using sockets:

- The server instantiates a `ServerSocket` object, denoting which port number communication is to occur on.
- The server invokes the `accept()` method of the `ServerSocket` class. This method waits until a client connects to the server on the given port.
- After the server is waiting, a client instantiates a `Socket` object, specifying the server name and port number to connect to.
- The constructor of the `Socket` class attempts to connect the client to the specified server and port number. If communication is established, the client now has a `Socket` object capable of communicating with the server.
- On the server side, the `accept()` method returns a reference to a new socket on the server that is connected to the client's socket.

After the connections are established, communication can occur using I/O streams. Each socket

has both an `OutputStream` and an `InputStream`. The client's `OutputStream` is connected to the server's `InputStream`, and the client's `InputStream` is connected to the server's `OutputStream`.

TCP is a twoway communication protocol, so data can be sent across both streams at the same

time. There are following usefull classes providing complete set of methods to implement sockets.

METHODS:

1: `void connect(SocketAddress endpoint)`

connects this socket to the server

2: void getchannel();

Returns the unique socket channel object

3: getInputStream()

returns an input stream for this object

4: getLocalAddress()

Gets the local address to which the socket is bound

5: close()

Close the socket

SERVERSOCKET

This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

METHODS

1: accept()

Listens for a connection to be made to this socket and accepts it.

2: close()

Closes this socket.

3: getchannel()

Returns the unique ServerSocketChannel object associated with this socket, if any.

4: getlocalHost()

Returns the port on which this socket is listening.

System:

The System class contains several useful class fields and methods. It cannot be instantiated.

Among the facilities provided by the System class are standard input, standard output, and error

output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

FIELD DETAIL

IN

The "standard" input stream. This stream is already open and ready to supply input data. Typically this stream corresponds to keyboard input or another input source specified by the host environment or user.

OUT

The "standard" output stream. This stream is already open and ready to accept output data. Typically this stream corresponds to display output or another output destination specified by the host environment or user.

DATAINPUTSTREAM

A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way. An application uses a data output stream to write data that can later be read by a data input stream.

METHODS:

1.readInt()

read 4 input bytes

2.readFloat()

Read 4 input bytes

3.readChar()

Read 2 input bytes

4.readLine()

Reads the next line of text

5.readUTF()

Reads the string encoded in UTF-8 format

DATAOUTPUTSTREAM

A data output stream lets an application write primitive Java data types to an output stream in a portable way. An application can then use a data input stream to read the data back in.

METHODS

1.flush()

Flushes the data output stream

2.writeByte(int v)

write out the byte

3.writechar(int v)

Write the char

4.writeUTF()

Write a string to the underlying outputstream using modified UTF-8

5.writeInt()

Write an int to the output stream

BUFFERED READER

Reads text from a character-input stream, buffering characters so as to provide for the efficient

reading of characters, arrays, and lines.

The buffer size may be specified, or the default size may be used. The default is large enough for

most purposes.

In general, each read request made of a Reader causes a corresponding read request to be made

of the underlying character or byte stream. It is therefore advisable to wrap a `BufferedReader` around any Reader whose `read()` operations may be costly, such as `FileReaders` and

`InputStreamReaders`. For example,

`BufferedReader in`

```
= new BufferedReader(new FileReader("foo.in"));
```

will buffer the input from the specified file. Without buffering, each invocation of `read()` or `readLine()` could cause bytes to be read from the file, converted into characters, and then returned, which can be very inefficient.

BUFFERED WRITER

Writes text to a character-output stream, buffering characters so as to provide for the efficient writing of single characters, arrays, and strings.

The buffer size may be specified, or the default size may be accepted. The default is large enough for most purposes.

A `newLine()` method is provided, which uses the platform's own notion of line separator as

defined by the system property `line.separator`. Not all platforms use the newline character (`\n`) to terminate lines. Calling this method to terminate each output line is therefore preferred to writing a newline character directly.

In general, a `Writer` sends its output immediately to the underlying character or byte stream. Unless prompt output is required, it is advisable to wrap a `BufferedWriter` around any `Writer` whose `write()` operations may be costly, such as `FileWriters` and `OutputStreamWriters`. For example,

```
PrintWriter out  
= new PrintWriter(new BufferedWriter(new FileWriter("foo.out")));
```

PRINTSTREAM

A `PrintStream` adds functionality to another output stream, namely the ability to print representations of various data values conveniently. Two other features are provided as well. Unlike other output streams, a `PrintStream` never throws an `IOException`; instead, exceptional situations merely set an internal flag that can be tested via the `checkError` method. Optionally, a `PrintStream` can be created so as to flush automatically; this means that the `flush` method is automatically invoked after a byte array is written, one of the `println` methods is invoked, or a newline character or byte (`\n`) is written.

METHODS

1.`append(char c)`

appends the specified character to the output stream

2.`close()`

Close the stream

3.`flush()`

Flush the stream

4.`print(char c)`

prints a char

5.`write(int b)`

Writes the specified byte to the stream

`InetAddress` Class Methods:

This class represents an Internet Protocol (IP) address. Here are following usefull methods which

you would need while doing socket programming:

SN Methods with Description

1 static InetAddress getByAddress(byte[] addr)

Returns an InetAddress object given the raw IP address .

2 static InetAddress getByAddress(String host, byte[] addr)

Create an InetAddress based on the provided host name and IP address.

3 static InetAddress getByName(String host)

Determines the IP address of a host, given the host's name.

4 String getHostAddress()

Returns the IP address string in textual presentation.

5 String getHostName()

Gets the host name for this IP address.

6 static InetAddress InetAddress getLocalHost()

Returns the local host.

7 String toString()

Converts this IP address to a String.

RESULT:

The basics of socket programming is studied.

CONTENT BEYOND SYLLABUS

STUDY OF NETWORK PROGRAMMING IN PYTHON

Python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

This chapter gives you understanding on most famous concept in Networking - Socket Programming.

What is Sockets?

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.

Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The *socket* library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Sockets have their own vocabulary –

Sr.No.	Term & Description
1	Domain The family of protocols that is used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
2	type The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
3	protocol Typically zero, this may be used to identify a variant of a protocol within a domain and type.
4	hostname The identifier of a network interface – <ul style="list-style-type: none">• A string, which can be a host name, a dotted-quad address, or an IPV6

	<p>address in colon (and possibly dot) notation</p> <ul style="list-style-type: none"> • A string "<broadcast>", which specifies an INADDR_BROADCAST address. • A zero-length string, which specifies INADDR_ANY, or • An Integer, interpreted as a binary address in host byte order.
5	<p>port</p> <p>Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.</p>

The *socket* Module

To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax –

```
s = socket.socket (socket_family, socket_type, protocol=0)
```

Here is the description of the parameters –

- **socket_family** – This is either AF_UNIX or AF_INET, as explained earlier.
- **socket_type** – This is either SOCK_STREAM or SOCK_DGRAM.
- **protocol** – This is usually left out, defaulting to 0.

Once you have *socket* object, then you can use required functions to create your client or server program. Following is the list of functions required –

Server Socket Methods

Sr.No.	Method & Description
1	<p>s.bind()</p> <p>This method binds address (hostname, port number pair) to socket.</p>
2	<p>s.listen()</p> <p>This method sets up and start TCP listener.</p>
3	<p>s.accept()</p> <p>This passively accept TCP client connection, waiting until connection arrives (blocking).</p>

Client Socket Methods

Sr.No.	Method & Description
1	s.connect() This method actively initiates TCP server connection.

General Socket Methods

Sr.No.	Method & Description
1	s.recv() This method receives TCP message
2	s.send() This method transmits TCP message
3	s.recvfrom() This method receives UDP message
4	s.sendto() This method transmits UDP message
5	s.close() This method closes socket
6	socket.gethostname() Returns the hostname.

A Simple Server

To write Internet servers, we use the **socket** function available in socket module to create a socket object. A socket object is then used to call other functions to setup a socket server.

Now call **bind(hostname, port)** function to specify a *port* for your service on the given host.

Next, call the *accept* method of the returned object. This method waits until a client connects to the port you specified, and then returns a *connection* object that represents the connection to that client.

```
#!/usr/bin/python      # This is server.py file

import socket          # Import socket module

s = socket.socket()     # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345           # Reserve a port for your service.
s.bind((host, port))    # Bind to the port

s.listen(5)            # Now wait for client connection.
while True:
    c, addr = s.accept() # Establish connection with client.
    print 'Got connection from', addr
    c.send('Thank you for connecting')
    c.close()           # Close the connection
```

A Simple Client

Let us write a very simple client program which opens a connection to a given port 12345 and given host. This is very simple to create a socket client using Python's *socket* module function.

The **socket.connect(hostname, port)** opens a TCP connection to *hostname* on the *port*. Once you have a socket open, you can read from it like any IO object. When done, remember to close it, as you would close a file.

The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits –

```
#!/usr/bin/python      # This is client.py file

import socket          # Import socket module

s = socket.socket()     # Create a socket object
host = socket.gethostname() # Get local machine name
port = 12345           # Reserve a port for your service.

s.connect((host, port))
print s.recv(1024)
s.close()              # Close the socket when done
```

Now run this server.py in background and then run above client.py to see the result.

```
# Following would start a server in background.
$ python server.py &
```

Once server is started run client as follows:

```
$ python client.py
```

This would produce following result –

```
Got connection from ('127.0.0.1', 48437)
```

```
Thank you for connecting
```

Python Internet modules

A list of some important modules in Python Network/Internet programming.

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib