

CSL 332

NETWORKING LAB MANUAL

Prepared By

Mrs. Sruthy Manmadhan
Assistant Professor

Department of Computer Science and Engineering
N.S.S College of Engineering, Palakkad

CONTENTS

Sl. No.	Program	Page No.
1	Familiarization of Networking Commands in Linux	2
2	To familiarize and understand the use and functioning of system calls used for network programming in Linux	11
3	Implement client-server communication using socket programming and TCP as transport layer protocol.....	14
4	Implement client-server communication using socket programming and UDP as transport layer protocol.....	17
5	Implement congestion control using a leaky bucket algorithm	22
6	Simulate sliding window flow control protocols	27
7	Implement and simulate algorithm for Distance Vector Routing protocol or Link State Routing protocol.....	32
8	Implement Simple Mail Transfer Protocol	38
9	Implement File Transfer Protocol	41
10	Familiarization of Wireshark packet capturing.....	64
11	Study of NS2 simulator.....	71

1. Familiarization of Networking Commands in Linux

1.1 Aim

Familiarize the following networking commands available in Linux

Sl.No	Command	Sl.No	Command
1	Netstat	10	host
2	ss	11	hostname
3	ping	12	Telnet
4	tracert	13	w
5	tracert	14	wget
6	ifconfig	15	mtr
7	ip	16	ifplugstatus
8	route	17	scp
9	arp		

1.2 Commands

1.Netstat

The netstat command stands for **Network Statistic**. It displays information about different interface statistics including open sockets, routing tables and connection information.

Syntax - netstat[address option]

Example - netstat -r

This command displays routing table information

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ netstat -r  
Kernel IP routing table  
Destination      Gateway         Genmask         Flags         MSS Window  irtt Iface  
default          10.0.0.1        0.0.0.0         UG            0 0        0 wlan0  
10.0.0.0          *               255.255.255.0   U             0 0        0 wlan0  
link-local        *               255.255.0.0     U             0 0        0 wlan0  
sssit@JavaTpoint:~$
```

2.ss

The ss command is a replacement for netstat command. This command gives more information in comparison to the netstat. It is also faster than netstat as it gets all information from kernel userspace.

Syntax - ss

Example -

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ss  
State      Recv-Q Send-Q           Local Address:Port           Peer Address:Port  
CLOSE-WAIT 1        0           10.0.0.11:47137             91.189.89.144:http  
ESTAB      0        0           10.0.0.11:45681            198.252.206.25:https  
sssit@JavaTpoint:~$
```

3.ping

The ping command stands for **(Packet INternet Groper)**. It checks the connectivity between two nodes that is whether a server is reachable or not. ping command keep executing and sends the packet until you interrupt. To stop from execution press **ctrl + c**

Syntax – ping <destination>

Example–

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ping 2.2.2.2  
PING 2.2.2.2 (2.2.2.2) 56(84) bytes of data.  
^C
```

4.traceroute

The traceroute command is a network troubleshooting utility which helps us to determine number of hops and packets travelling path required to reach a destination.

Syntax –traceroute <destination>

Example -

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ traceroute javatpoint.com  
traceroute to javatpoint.com (144.76.11.18), 64 hops max  
 1  10.0.0.1 (10.0.0.1) 0.788ms 0.632ms 0.696ms  
 2  192.168.1.1 (192.168.1.1) 2.661ms 2.614ms 2.578ms  
 3  122.176.127.68 (122.176.127.68) 57.322ms 76.686ms *  
 4  122.176.127.33 (122.176.127.33) 129.848ms 26.913ms 26.151ms  
 5  125.19.38.109 (125.19.38.109) 22.831ms 25.645ms 29.178ms  
 6  182.79.234.61 (182.79.234.61) 360.588ms 370.449ms 348.176ms  
 7  182.79.211.102 (182.79.211.102) 304.926ms 308.559ms 306.309ms  
 8  182.79.217.182 (182.79.217.182) 307.927ms 305.935ms 203.724ms  
 9  182.79.211.25 (182.79.211.25) 75.589ms 72.094ms 69.982ms  
10  182.79.198.129 (182.79.198.129) 302.147ms * 267.738ms  
11  80.249.209.55 (80.249.209.55) 510.565ms 409.538ms 409.573ms  
12  213.239.203.157 (213.239.203.157) 409.183ms 409.937ms 409.286ms  
13  213.239.229.78 (213.239.229.78) 513.178ms 510.590ms 409.560ms  
14  213.239.229.54 (213.239.229.54) 409.936ms 410.547ms 408.193ms  
15  144.76.11.18 (144.76.11.18) 409.678ms 528.521ms 494.972ms  
sssit@JavaTpoint:~$
```

5.tracepath

It is similar to traceroute command, but it doesn't require root privileges. By default, it is

installed in Ubuntu but you may have to download traceroute on Ubuntu. It traces the network path of the specified destination and reports each hop along the path. If you have a slow network then tracepath will show you where your network is weak.

Syntax – tracepath <destination>

Example –

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ tracepath javatpoint.com  
1:  JavaTpoint.local                0.077ms pmtu 1500  
1:  10.0.0.1                        21.342ms  
1:  10.0.0.1                        4.281ms  
2:  █
```

6.ifconfig

The command ifconfig stands for interface configurator. This command enables us to initialize an interface, assign IP address, enable or disable an interface. It display route and network interface. You can view IP address, MAC address and MTU (Maximum Transmission Unit) with ifconfig command.

Syntax – ifconfig

Example–

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ifconfig  
eth0      Link encap:Ethernet  HWaddr 94:de:80:87:7c:3c  
          UP BROADCAST MULTICAST  MTU:1500  Metric:1  
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)  
          Interrupt:40 Base address:0xc000  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          inet6 addr: ::1/128 Scope:Host  
          UP LOOPBACK RUNNING  MTU:16436  Metric:1  
          RX packets:2466 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:2466 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:228069 (228.0 KB)  TX bytes:228069 (228.0 KB)  
  
wlan0     Link encap:Ethernet  HWaddr c8:3a:35:c2:a4:cd  
          inet addr:10.0.0.11  Bcast:10.0.0.255  Mask:255.255.255.0  
          inet6 addr: fe80::ca3a:35ff:fec2:a4cd/64 Scope:Link  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:36408 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:18520 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:15698000 (15.6 MB)  TX bytes:4489507 (4.4 MB)  
  
sssit@JavaTpoint:~$ █
```

7.ip

This is the newer version of ifconfig command.

Syntax - ip a or ip addr

Example –

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast state DOWN qlen 1000  
    link/ether 94:de:80:87:7c:3c brd ff:ff:ff:ff:ff:ff  
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000  
    link/ether c8:3a:35:c2:a4:cd brd ff:ff:ff:ff:ff:ff  
    inet 10.0.0.11/24 brd 10.0.0.255 scope global wlan0  
    inet6 fe80::ca3a:35ff:fec2:a4cd/64 scope link  
        valid_lft forever preferred_lft forever  
sssit@JavaTpoint:~$
```

8.route

The route command displays and manipulate IP routing table for your system. A router is a device which is basically used to determine the best way to route packets to a destination.

Syntax -route

Example –

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ route  
Kernel IP routing table  
Destination      Gateway         Genmask         Flags Metric Ref    Use Iface  
default          10.0.0.1        0.0.0.0         UG      0      0        0 wlan0  
10.0.0.0         *               255.255.255.0   U        2      0        0 wlan0  
link-local       *               255.255.0.0     U       1000    0        0 wlan0  
sssit@JavaTpoint:~$
```

9.arp

The command arp stands for **A**ddress **R**esolution **P**rotocol. It allows us to view or add content into kernel's ARP table.

Syntax – arp

Example –

```
sssit@JavaTpoint: ~  
sssit@JavaTpoint:~$ arp  
Address          HWtype  HWaddress      Flags Mask    Iface  
10.0.0.1         ether   c0:ff:d4:91:49:df C           wlan0  
sssit@JavaTpoint:~$
```


10.host

This command displays domain name for given IP address and displays IP address for given domain name. It also performs DNS lookups related to DNS query.

Syntax – host <address>

Example –

A terminal window with a dark purple background and a title bar that reads 'sssit@JavaTpoint: ~'. The terminal shows the following commands and output:

```
sssit@JavaTpoint:~$ host javatpoint.com
javatpoint.com has address 144.76.11.18
javatpoint.com mail is handled by 0 javatpoint.com.
sssit@JavaTpoint:~$
sssit@JavaTpoint:~$ host 144.76.11.18
18.11.76.144.in-addr.arpa domain name pointer www.javatpoint.com.
sssit@JavaTpoint:~$
```

11.hostname

With the help of hostname command you can set and view hostname of the system.

Syntax – hostname

Example -

A terminal window with a dark purple background and a title bar that reads 'sssit@JavaTpoint: ~'. The terminal shows the following command and output:

```
sssit@JavaTpoint:~$ hostname
JavaTpoint
sssit@JavaTpoint:~$
```

12. Telnet

The **telnet** command is used for interactive communication with another host using the TELNET protocol. It begins in command mode, where it prints a telnet command prompt

Syntax – telnet

Example –

```

guru99@VirtualBox:~$ telnet localhost
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Ubuntu 11.10
VirtualBox login: guru99
Password:
Welcome to Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic i686)

 * Documentation:  https://help.ubuntu.com/

New release '12.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
guru99@VirtualBox:~$

```

13.w

The command **w** on many Unix-like operating systems provides a quick summary of every user logged into a computer, what each user is currently doing, and what load all the activity is imposing on the computer itself. The command is a one-command combination of several other Unix programs: who, uptime, and ps -a.

Syntax – w

Example –

```

w
 11:12am up 608 day(s), 19:56, 6 users, load average: 0.36, 0.36, 0.37
User  tty  login@ idle what
smithj pts/5   8:52am   w
jonesm pts/23  20Apr06  28 -bash
harry  pts/18  9:01am   9 pine
peterb pts/19  21Apr06   emacs -nw html/index.html
janetmcq pts/8   10:12am  3days -csh
singh  pts/12  16Apr06  5:29 /usr/bin/perl -w perl/test/program.pl

```

14.wget

This command is used to download a file from the internet using CLI. wget command will be used without any option. The file will be saved in the current directory.

Syntax – wget <filelink>

Example

—


```

sssit@JavaTpoint: ~
sssit@JavaTpoint:~$ wget google.com/doodles/new-years-day-2012
--2016-07-31 11:59:08-- http://google.com/doodles/new-years-day-2012
Resolving google.com (google.com)... 216.58.199.142, 2404:6800:4009:806::200e
Connecting to google.com (google.com)|216.58.199.142|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.google.com/doodles/new-years-day-2012 [following]
--2016-07-31 11:59:09-- https://www.google.com/doodles/new-years-day-2012
Resolving www.google.com (www.google.com)... 74.125.68.99, 74.125.68.103, 74.125.68.104, ...
Connecting to www.google.com (www.google.com)|74.125.68.99|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 448662 (438K) [text/html]
Saving to: 'new-years-day-2012'

100%[=====] 4,48,662 495K/s in 0.9s

2016-07-31 11:59:17 (495 KB/s) - 'new-years-day-2012' saved [448662/448662]

sssit@JavaTpoint:~$

```

15.mtr

The mtr command is a combination of ping and traceroute command. It continuously sends packet showing ping time for each hop. It also shows network problems.

Syntax – mtr <path>

Example –

```

sssit@JavaTpoint: ~
My traceroute [v0.80]
javaTpoint (0.0.0.0) Sun Jul 31 12:30:51 2016
Resolver error: Answered class does not match queried class.fields quit
Packets
Host Loss% Snt Last Avg Best Wrst StDev
1. 10.0.0.1 0.0% 3 0.8 13.0 0.8 21.3 10.8
2. 192.168.1.1 0.0% 2 2.0 2.1 2.0 2.3 0.2
3. 122.176.127.68 0.0% 2 21.1 21.4 21.1 21.6 0.3
4. abts-north-static-038.127.176.12 50.0% 2 1379. 1379. 1379. 1379. 0.0
5. 125.18.48.45 0.0% 2 20.0 30.3 20.0 40.5 14.5
6. 182.79.245.238 0.0% 2 170.8 180.2 170.8 189.5 13.2
7. amsix-gw.hetzner.de 0.0% 2 306.8 309.8 306.8 312.8 4.3
8. 213.239.203.157 0.0% 2 297.8 299.5 297.8 301.2 2.3
9. 213.239.203.154 0.0% 2 322.8 322.8 322.8 322.8 0.0
10. 213.239.229.50 0.0% 2 318.5 318.5 318.5 318.5 0.0
11. www.javatpoint.com 0.0% 2 292.4 292.4 292.4 292.4 0.0

```

16.ifplugstatus

This command tells us whether a cable is plugged into our network interface or not. By default, it is not installed in Ubuntu, to install it use command **sudo apt-get install ifplugd**.

Syntax – ifplugstatus

Example –

```

sssit@JavaTpoint: ~
sssit@JavaTpoint:~$ ifplugstatus
lo: link beat detected
wlan0: link beat detected
eth0: unplugged
sssit@JavaTpoint:~$

```

17.scp

Is the command itself and tells the operating system to copy one or more files over a secure shell connection, better known as ssh connection.

Syntax – scp <source file><destination file>

Example –

```
scp /path/to/source-file user@host:/path/to/destination-folder/
```

2. Familiarization of System Calls

2.1 Aim

Familiarize and understand the use and functioning of system calls used for network programming in Linux

2.2 Theoretical Background

A system call is a procedure that provides the interface between a process and the operating system. It is the way by which a computer program requests a service from the kernel of the operating system.

System calls are divided into 5 categories mainly:

- 1 Process Control
- 2 File Management
- 3 Device Management
- 4 Information Maintenance
- 5 Communication

I. Process Control

This system calls perform the task of process creation, process termination, etc.

- a) **fork()** :- A new process is created by the fork() system call. A new process may be created with fork() without a new program being run-the new sub-process simply continues to execute exactly the same program that the first (parent) process was running.
- b) **exec()** :- A new program will start executing after a call to exec(). Running a new program does not require that a new process be created first: any process may call exec() at any time. The currently running program is immediately terminated, and the new program starts executing in the context of the existing process.

II. File Management

This system call handles file manipulation jobs like creating a file, reading, writing, etc.

- a) **open() :-** It is the system call to open a file. This system call just opens the file, to perform operations such as read and write, we need to execute different system call to perform the operations.
- b) **read() :-** This system call opens the file in reading mode. We can not edit the files with this system call. Multiple processes can execute the read() system call on the same file simultaneously.

III. Device Management

Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc.

- a) **ioctl() :-** ioctl() is referred to as Input and Output Control. ioctl is a system call for device-specific input/output operations and other operations which cannot be expressed by regular system calls.
- b) **write() :-** It is used to write data from a user buffer to a device like a file. This system call is one way for a program to generate data. It takes three arguments in general:
 - A file descriptor.
 - A pointer to the buffer in which data is saved.
 - The number of bytes to be written from the buffer.

IV. Information Maintenance

It handles information and its transfer between the OS and the user program. In addition, OS keeps the information about all its processes and system calls are used to access this information.

- a) **getpid() :-** getpid stands for Get the Process ID. The getpid() function shall return the process ID of the calling process. The getpid() function shall always be successful and no return value is reserved to indicate an error.
- b) **sleep() :-** This System call suspends the execution of the currently running process for some interval of time. Meanwhile, during this interval, another process is given chance to execute

V. Communication

These types of system calls are specially used for inter-process communications through Message Passing and Shared memory

- a) **pipe() :-** The pipe() system call is used to communicate between different Linux processes. It is mainly used for inter-process communication. The pipe() system function is used to open file descriptors.
- b) **shmget() :-** shmget stands for shared memory segment. It is mainly used for Shared memory communication. This system call is used to access the shared memory and access the messages in order to communicate with the process.

2.3. Programming Assignment:

1 Write a Java program that does the following:

1.a Takes an integer argument (say, N1) from the command line.

1.b Forks two children processes

b.i First child computes $1+2+\dots+N1$ (sum of positive integers up to N1) and prints out the result and its own identifier.

b.ii Second child computes $1*2*\dots*N1$ (the factorial of N1) and prints out the result and its own identifier.

1.c Parent waits until both children are finished, then prints out the message "Done"

2 Write a Java program to copy contents of a file "F1.txt" to another file "F2.txt". [Use system calls open(), read() and write()]

Algorithm:

1)

STEP 1:- Start

STEP 2:- Get n from user as a command line argument

STEP 3:- Create an object of ForkJoinClass and initialize the fork STEP 4:- In the class which extended RecursiveTask Class

Initialize variables using constructor Override compute method

If $c==0$ then compute and return sum of 1st n numbers else compute and return factorial of n

STEP 5:- Create separate objects of the class that extended RecursiveTask class for computing factorial and sum of n integers respectively.

STEP 6:- Submitting above created tasks to ForkJoinTool to run fork using invoke() STEP 7:- Display sum of 1st n numbers

STEP 8:- Display factorial of n STEP 9:- Stop

2)

STEP 1:- Start

STEP 2:- Open files "F1.txt" and "F2.txt"

STEP 3:- Using FileInputStream Class create a variable named input1 to read from "F1.txt"

STEP 4:- Using FileOutputStream Class create a variable named output1 to write to file "F2.txt"

STEP 5:- Repeat until not end of "F1.txt"

read a byte from "F1.txt" using input1 variable and write it to "F2.txt" using output1 variable

STEP 6:- Close opened files

STEP 7:- Display file has been successfully copied STEP 8:- Stop

2.4 Program

1)

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

```
import java.util.concurrent.RecursiveTask; import java.util.concurrent.ForkJoinPool;
```

```
class comp extends RecursiveTask<Integer>
```

```
{
```

```
int n; int c;
```

```
public comp(int n,int c)
```

```
{
```

```
this.n = n; this.c = c;
```

```
}
```

```
@Override
```

```
protected Integer compute()
```

```
{
```

```
if(c == 0)
```

```
return sum();
```

```
else
```

```
return fact();
```

```
}
```

```
private int fact()
```

```
{
```

```
int fact = 1;
```

```
for(int i = 1; i<=n; i++)
```

```
{
```

```
fact*=i;
```

```
}
```

```
return fact;
```

```
}
```

```

private int sum()
{
return (n*(n+1)/2);
}
}
class ForkJoinExample
{
public static void main(String argsv[])
{
int n = Integer.parseInt(argsv[0]); System.out.printf("Entered integer : %d\n", n);
// Creating an object of the ForkJoinPool class ForkJoinPool pool = ForkJoinPool.commonPool();
// Now creating an object of the SearchWork class comp sm = new comp(n,0);
comp fct = new comp(n,1);
// submitting the task to the ForkJoinPool int x = pool.invoke(sm);
int k = pool.invoke(fct); System.out.println("Factorial of "+n+" = "+ k);
System.out.println("Sum of integers upto "+n+" = "+ x); System.out.println("Done");
}
}

```

2)

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

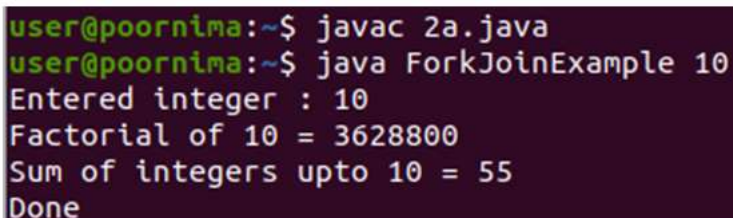
```

class CopyFromFileaToFileb {
public static void copyContent(File a, File b) throws Exception { FileInputStream in = new
FileInputStream(a); FileOutputStream out = new FileOutputStream(b);
int n;
while ((n = in.read()) != -1) { out.write(n);
}
in.close();
out.close(); System.out.println("File Copied");
}
public static void main(String[] args) throws Exception
{
File x = new File("F1.txt"); File y = new File("F2.txt"); copyContent(x, y);
}
}

```

Output

1)



```

user@poornima:~$ javac 2a.java
user@poornima:~$ java ForkJoinExample 10
Entered integer : 10
Factorial of 10 = 3628800
Sum of integers upto 10 = 55
Done

```


2)



```
Open [icon] F1.txt Save [icon] [icon] [icon] [icon]
1 program to copy contents of a file

user@poornima:~$ javac 2b.java
user@poornima:~$ java CopyFromFileaToFileb
File Copied

Open [icon] F2.txt Save [icon] [icon] [icon] [icon]
1 program to copy contents of a file
```

3. Socket programming using TCP

3.1 Aim

Implement Client-Server communication using Socket Programming and TCP as transport layer protocol.

3.2 Theoretical Background

Network-based systems consist of a server, client, and a media for communication. A computer running a program that makes a request for services is called client machine. A computer running a program that offers requested services from one or more clients is called server machine. The media for communication can be wired or wireless network. Generally, programs running on client machines make requests to a program (often called as server program) running on a server machine. They involve networking services provided by the transport layer, which is part of the Internet software stack, often called TCP/IP (Transport Control Protocol/Internet Protocol) stack. The transport layer comprises two types of protocols, TCP (Transport Control Protocol) and UDP (User Datagram Protocol). The most widely used programming interfaces for these protocols are sockets. TCP is a connection-oriented protocol that provides a reliable flow of data between two computers. Example applications that use such services are HTTP, FTP, and Telnet.

A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Sockets provide an interface for programming networks at the transport layer. Network communication using Sockets is very much similar to performing file I/O. In fact, socket handle is treated like file handle. The streams used in file I/O operation are also applicable to socket-based I/O. Socket-based communication is independent of a programming language used for implementing it. That means, a socket program written in Java language can communicate to a program written in non-Java (say C or C++) socket program. A server (program) runs on a specific computer and has a socket that is bound to a specific port. The server listens to the socket for a client to make a connection request. If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to a different port. It needs a new socket (consequently a different port number) so that it can continue to listen to the original socket for connection requests while serving the connected client.

The two key classes from the java.net package used in creation of server and client programs are:

- ServerSocket
- Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket). In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept: developers have to open a socket, perform I/O, and close it.

3.3 Algorithm

- TCP Server:

1. Create server socket
2. Wait for client request
3. If client request received then
 - a. Get message from client's input stream
 - b. Write acknowledgement to client's output stream
 - c. Close client

- **TCP Client:**

1. Create client socket
2. Read input from user and send it to server
3. Print acknowledgement

3.4 Program

- **TCP Server:**

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

public class Tcpserver extends Thread
{
    Thread t1,t2;
    BufferedReader br,buf;
    PrintWriter pout;
    String str,a;

    ServerSocket server;
    Socket client;

    Tcpserver()
    {
        try
        {
            server=new ServerSocket(5555);
            client=server.accept();
            br=new BufferedReader(new InputStreamReader(System.in));
            buf=new BufferedReader(new
InputStreamReader(client.getInputStream()));
            pout=new PrintWriter(client.getOutputStream(),true);
        }
        catch(Exception e)
        {
            System.out.println("error :"+e);
        }
    }
}
```

```

public void run()
{
    while(true)
    {
        if(Thread.currentThread()==t1)
        {
            try
            {
                str=br.readLine();
                pout.println(str);
            }
            catch(Exception e)
            {
                System.out.println("error :"+e);
            }
        }
        else
        {
            try
            {
                a=buf.readLine();
                System.out.println("from client :"+a);
            }
            catch(Exception e)
            {
                System.out.println("error :"+e);
            }
        }
    }
}

public static void main(String args[])
{
    Tcpserver s=new Tcpserver();
    s.t1=new Thread(s);
    s.t2=new Thread(s);
    s.t1.start();
    s.t2.start();
}
}

```

- **TCP Client:**

```

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

public class Tcpclient extends Thread
{
    Thread t1,t2;
    BufferedReader br,buf;
}

```

```

        PrintWriter pout;
        String str,a;

        Socket client;

        Tcpclient()
        {
            try
            {
                client=new Socket("127.0.0.1",5555);
                br=new BufferedReader(new InputStreamReader(System.in));
                buf=new BufferedReader(new
InputStreamReader(client.getInputStream()));
                pout=new PrintWriter(client.getOutputStream(),true);
            }
            catch(Exception e)
            {
                System.out.println("error :"+e);
            }
        }

        public void run()
        {
            while(true)
            {
                if(Thread.currentThread()==t1)
                {
                    try
                    {
                        str=br.readLine();
                        pout.println(str);
                    }
                    catch(Exception e)
                    {
                        System.out.println("error :"+e);
                    }
                }
                else
                {
                    try
                    {
                        a=buf.readLine();
                        System.out.println("from server :"+a);
                    }
                    catch(Exception e)
                    {
                        System.out.println("error :"+e);
                    }
                }
            }
        }
    }
}

```

```

    }
}
public static void main(String args[])
{
    Tcpclient s=new Tcpclient();
    s.t1=new Thread(s);
    s.t2=new Thread(s);
    s.t1.start();
    s.t2.start();} }

```

3.5. Output

Tcpserver	Tcpclient
from client :hii from client :hloo from client :how are you? am fine	from server :hii hloo how are you? from server :am fine

4. Socket programming using UDP

4.1 Aim

Implement Client-Server communication using Socket Programming and UDP as transport layer protocol.

4.2 Theoretical Background

TCP guarantees the delivery of packets and preserves their order on destination. Sometimes these features are not required and since they do not come without performance costs, it would be better to use a lighter transport protocol. This kind of service is accomplished by the UDP protocol which conveys datagram packets. Datagram packets are used to implement a connectionless packet delivery service supported by the UDP protocol. Each message is transferred from source machine to destination based on information contained within that packet. That means, each packet needs to have destination address and each packet might be routed differently, and might arrive in any order. Packet delivery is not guaranteed.

Java supports datagram communication through the following classes:

- DatagramPacket
- DatagramSocket

The class DatagramPacket contains several constructors that can be used for creating packet object. One of them is: DatagramPacket(byte[] buf, int length, InetAddress address, int port); This constructor is used for creating a datagram packet for sending packets of length length to the specified port number on the specified host. The message to be transmitted is indicated in the first argument. The key methods of DatagramPacket class are: byte[] getData() Returns the data buffer. int getLength() Returns the length of the data to be sent or the length of the data received. void setData(byte[] buf) Sets the data buffer for this packet. void setLength(int length) Sets the length for this packet. The class DatagramSocket supports various methods that can be used for transmitting or receiving data a datagram over the network. The two key methods are: void send(DatagramPacket p) Sends a datagram packet from this socket. void receive(DatagramPacket p) Receives a datagram packet from this socket. A simple UDP server program that waits for client's requests and then accepts the message (datagram) and sends back the same message is given below. Of course, an extended server program can manipulate client's messages/request and send a new message as a response.

4.3 Algorithm

- **UDP Server:**
 1. Start
 2. Create Datagram Socket
 3. Now listen to a port

4. Create empty byte array
5. While (true)
 - a. Create a Datagram Packet
 - b. Receive Datagram Packet from client
 - c. Now convert byte to string
 - d. Print message
6. End While
7. Stop

- **UDP Client:**

1. Start
2. Create Datagram Socket
3. While (true)
 - a. Get input from keyboard
 - b. Convert string to byte
 - c. Create a packet with destination, IP, port, address
 - d. Attach packet with byte array (data)
 - e. Send the packet
4. End While
5. Stop

4.4 Program

- **UDP Server:**

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

```
public class Serverudp extends Thread
```

```
{
    Thread t1,t2;
    BufferedReader br;
    byte[] rec;
    byte[] send=new byte[20];
    DatagramSocket server;
    DatagramPacket dp=null;
    public static InetAddress ip;
    public static int port;
    String str,a;
```

```
Serverudp()
```

```
{
    try
    {
        server=new DatagramSocket(5555);
        br=new BufferedReader(new InputStreamReader(System.in));

    }
    catch(Exception e)
```

```

        {
            System.out.println("error :"+e);
        }
    }
    public void run()
    {
        while(true)
        {
            if(Thread.currentThread()==t1)
            {
                try
                {
                    rec=new byte[20];
                    dp=new DatagramPacket(rec,rec.length);
                    server.receive(dp);
                    ip=dp.getAddress();
                    port=dp.getPort();
                    str=new String(rec);

                    System.out.println("from client:"+str);
                }
                catch(Exception e)
                {
                    System.out.println("error :"+e);
                }
            }
            else
            {
                try
                {
                    a=br.readLine();
                    send=a.getBytes();
                    dp=new
DatagramPacket(send,send.length,ip,port);
                    server.send(dp);
                }
                catch(Exception e)
                {
                    System.out.println("error :"+e);
                }
            }
        }
    }
    public static void main(String args[])
    {
        Serverudp s=new Serverudp();
        s.t1=new Thread(s);
        s.t2=new Thread(s);
        s.t1.start();
        s.t2.start();
    }
}

```

UDP Client:

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

```

import java.net.*;

public class Clientudp extends Thread
{
    Thread t1,t2;
    DatagramSocket client;
    DatagramPacket dp;

    BufferedReader br;
    byte[] send=new byte[20];
    byte[] rec=new byte[20];
    InetAddress ip;
    String str,a;

    Clientudp()
    {
        try
        {
            client=new DatagramSocket();
            ip=InetAddress.getLocalHost();
            br=new BufferedReader(new InputStreamReader(System.in));
        }
        catch(Exception e)
        {
            System.out.println("error :"+e);
        }
    }

    public void run()
    {
        while(true)
        {
            if(Thread.currentThread()==t1)
            {
                try
                {
                    dp=new DatagramPacket(rec,rec.length);
                    client.receive(dp);
                    str=new String(rec);
                    System.out.println("from server:"+str);
                }
                catch(Exception e)
                {
                    System.out.println("error :"+e);
                }
            }
            else
            {
                try
                {
                    a=br.readLine();
                    send=a.getBytes();
                    dp=new DatagramPacket(send,send.length,ip,5555);
                    client.send(dp);
                }
                catch(Exception e)
                {
                    System.out.println("error :"+e);
                }
            }
        }
    }
}

```

4.5 Output

UdpServer	Udpclient
from client:hii hllo from client:how are youu? am fine..thankyou from client:ok bye	hii from server:hllo how are youu? from server:am fine..thankyou ok bye

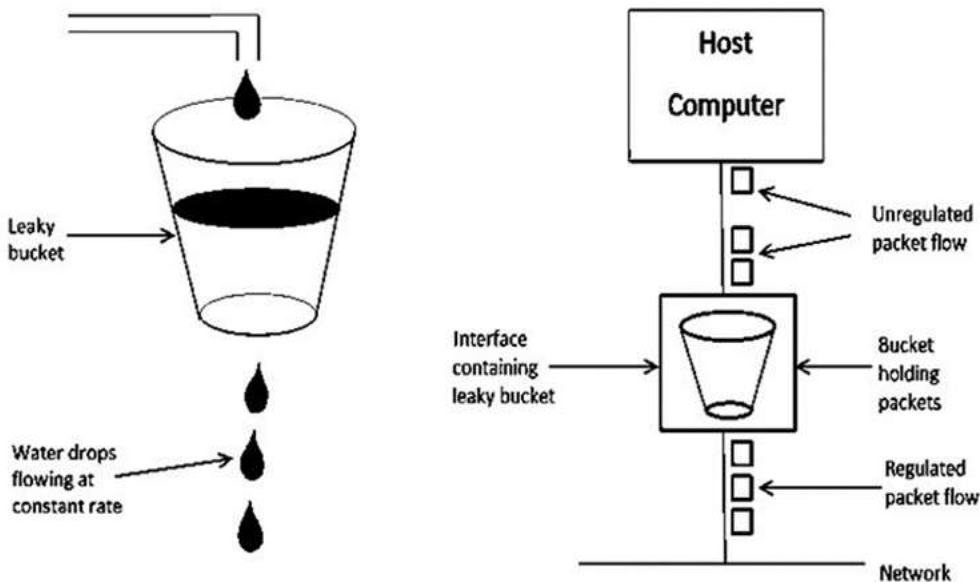
5. Leaky Bucket Algorithm

5.1 Aim

Implement congestion control using a leaky bucket algorithm in Java

5.2 Theoretical Background

There are 2 types of traffic shaping algorithms :- leaky bucket and token bucket. Suppose we have a bucket in which we are pouring water in a bursty rate but we have to get water in a fixed rate, for this we will make a hole at the bottom of the bucket. The input rate can vary but the output rate remains the same. Similarly in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in bucket and sent out at an average rate.



3.3 Algorithm:

STEP 1:- Start

STEP 2:- Get bucket size, outgoing rate, incoming rate and number of inputs from user as input

STEP 3:- Initialize store=0

STEP 4:- Repeat until number of inputs > 0

Display incoming size

if incoming size \leq (bucket size – store) store =store + incoming size Display remaining size
else

packet loss = incoming size - (bucket size – store) store = bucket size
Display packet loss and buffer size

store = store – outgoing Display remaining size

STEP 5:- Stop

3.4 Program

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

```
public class Main {
public static void main(String[] args) throws InterruptedException
{
int n, incoming, outgoing, store=0, bucketSize; Scanner sc = new Scanner(System.in);
System.out.print("Enter the bucket size:- "); bucketSize = sc.nextInt();
System.out.print("Enter the outgoing rate:- "); outgoing = sc.nextInt();
System.out.print("Enter the number of inputs:- "); n = sc.nextInt();
System.out.print("Enter the incoming size:- "); incoming = sc.nextInt();
while(n!=0)
{
System.out.println("\nIncoming size is " + incoming); if(incoming <=(bucketSize-store))
{
store +=incoming;
System.out.println("Bucket buffer size is " + store + " out of " + bucketSize);
}
else
{
System.out.println("packet loss = " + (incoming-( bucketSize-store))); store= bucketSize;
System.out.println("Buffer is full");
}
store-= outgoing;
System.out.println("After outgoing " + store + " packets are left out of "+bucketSize+
" in the buffer");
n--;
Thread.sleep(3000);
}
}
}
```

3.5 Output

```
user@poornima:~$ javac Main.java
user@poornima:~$ java Main
Enter the bucket size:- 300
Enter the outgoing rate:- 50
Enter the number of inputs:- 3
Enter the incoming size:- 200

Incoming size is 200
Bucket buffer size is 200 out of 300
After outgoing 150 packets are left out of 300 in the buffer

Incoming size is 200
packet loss = 50
Buffer is full
After outgoing 250 packets are left out of 300 in the buffer

Incoming size is 200
packet loss = 150
Buffer is full
After outgoing 250 packets are left out of 300 in the buffer
```

6. Sliding Window Protocols

A) Aim

To implement the stop and wait protocol using java programming language.

Theoretical Background

Flow control in computer networks is defined as the process of managing the rate of data transmission between two systems(nodes), this mechanism ensures that the rate of data (transmitted by the sender) is within the receiving capacity of the receiver node. The sliding window is a technique for sending multiple frames at a time. It controls the data packets between the two devices where reliable and gradual delivery of data frames is needed. It is also used in TCP (Transmission Control Protocol). In this technique, each frame has sent from the sequence number. The sequence numbers are used to find the missing data in the receiver end. The purpose of the sliding window technique is to avoid duplicate data, so it uses the sequence number. Stop and Wait, Go back N, Selective Repeat ARQ are types of sliding window protocols.

Stop and wait means, whatever the data that sender wants to send, he sends the data to the receiver. After sending the data, he stops and waits until he receives the acknowledgment from the receiver. The stop and wait protocol is a flow control protocol where flow control is one of the services of the data link layer. It is a data-link layer protocol which is used for transmitting the data over the noiseless channels. It provides unidirectional data transmission which means that either sending or receiving of data will take place at a time. It provides flow-control mechanism but does not provide any error control mechanism. The idea behind the usage of this frame is that when the sender sends the frame then he waits for the acknowledgment before sending the next frame.

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 sendNACK signal to client.
- 6.Stop the program.

Program

- **Sender:**

```
//SENDER//
import java.io.*;
import java.net.*;
import java.util.Scanner;
class stopwaitsender
{
    public static void main(String args[]) throws Exception
    {
        stopwaitsender sws = new stopwaitsender();
        sws.run();
    }
    public void run() throws Exception
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter no of frames to be sent:");
        int n=sc.nextInt();
        Socket myskt=new Socket("localhost",9999);
        PrintStream myps=new PrintStream(myskt.getOutputStream());
        for(int i=0;i<=n;)
        {
            if(i==n)
            {
                myps.println("exit");
                break;
            }
            System.out.println("Frame no "+i+" is sent");
            myps.println(i);
            BufferedReader bf=new BufferedReader(new
InputStreamReader(myskt.getInputStream()));
            String ack=bf.readLine();
            if(ack!=null)
            {
```

```
        System.out.println("Acknowledgement was Received from receiver");
        i++;
        Thread.sleep(4000);
    }
    else
    {
        myps.println(i);
    }
}
}
```

- **Receiver:**

```
//RECEIVER//
import java.io.*;
import java.net.*;
class stopwaitreceiver
{
    public static void main(String args[])throws Exception
    {
        stopwaitreceiver swr = new stopwaitreceiver();
        swr.run();
    }
    public void run() throws Exception
    {
        String temp="any message",str="exit";
        ServerSocket myss=new ServerSocket(9999);
        Socket ss_accept=myss.accept();
        BufferedReader ss_bf=new BufferedReader(new
        InputStreamReader(ss_accept.getInputStream()));
        PrintStream myps=new PrintStream(ss_accept.getOutputStream());
        while(temp.compareTo(str)!=0)
        {
            Thread.sleep(1000);
            temp=ss_bf.readLine();
            if(temp.compareTo(str)==0)
```

```

        { break;}
        System.out.println("Frame "+temp+" was received");
        Thread.sleep(500);
        myps.println("Received");
    }
    System.out.println("ALL FRAMES WERE RECEIVED SUCCESSFULLY");
}
}

```

Output

SENDER	RECEIVER
Enter no of frames to be sent:4 Frame no 0 is sent Acknowledgment was Received from receiver Frame no 1 is sent Acknowledgment was Received from receiver Frame no 2 is sent Acknowledgment was Received from receiver Frame no 3 is sent Acknowledgment was Received from receiver	Frame 0 was received Frame 1 was received Frame 2 was received Frame 3 was received ALL FRAMES WERE RECEIVED SUCCESSFULLY

B) Aim

A) To implement the Go Back N protocol using java programming language.

Theoretical Background

Go-Back-N protocol is a data link layer protocol that uses a sliding window method. In this, if any frame is corrupted or lost, all subsequent frames have to be sent again. The size of the sender window is N in this protocol. For example, Go-Back-8, the size of the sender window, will be 8. The receiver window size is always 1. If the receiver receives a corrupted frame, it cancels it. The receiver does not accept a corrupted frame. When the timer expires, the sender sends the correct frame again.

Algorithm

Go-Back-N Sender:

- Function Sender is
- Send_base \leftarrow 0;
- Nextseqnum \leftarrow 0;
- While True do
- If nextseqnum < send_base+N then
- Send packet nextseqnum;
- Nextseqnum \leftarrow nextseqnum+1;
- End
- If receive ACK n then
- Send_base \leftarrow n+1;
- If send_base == nextseqnum then
- Stop timer;
- Else
- Start timer;
- End
- End
- If timeout then
- Start timer;
- Send packet send_base;
- Send packet send_base+1;
-
- Send packet nextseqnum-1;
- End
- End
- End

Go-Back-N Receiver:

- Function Receiver is
- Nextseqnum \leftarrow 0;
- While True do

- If A packet is received then
- If the received packet is not corrupted and
sequence_number==nextseqnum then deliver the data to the upper layer;
- Send ACK nextseqnum;
- Nextseqnum \leftarrow nextseqnum+1;
- Else
- /* if the packet is corrupted or out of order, simply drop it*/
- Send ACK nextseqnum-1;
- End
- Else
- End
- End

Program

- **Client:**

```
import java.util.*;
import java.net.*;
import java.io.*;
public class Client
{
    public static void main(String args[]) throws Exception
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Enter the value of m : ");
        int m=Integer.parseInt(br.readLine());
        int x=(int)((Math.pow(2,m))-1);
        System.out.print("Enter no. of frames to be sent:");
        int count=Integer.parseInt(br.readLine());
        int data[]=new int[count];
        int h=0;
        for(int i=0;i<count;i++)
        {
            System.out.print("Enter data for frame no " +h+ " => ");
            data[i]=Integer.parseInt(br.readLine());
            h=(h+1)%x;
        }
    }
}
```

```
Socket client=new Socket("localhost",6262);
ObjectInputStream ois=new ObjectInputStream(client.getInputStream());
ObjectOutputStream oos=new ObjectOutputStream(client.getOutputStream());
System.out.println("Connected with server.");
boolean flag=false;
GoBackNListener listener=new GoBackNListener(ois,x);
listener=new GoBackNListener(ois,x);
listener.t.start();
int strt=0;
h=0;
oos.writeObject(x);
do
{
    int c=h;
    for(int i=h;i<count;i++)
    {
        System.out.print("|"+c+"|");
        c=(c+1)%x;
    }
    System.out.println();
    System.out.println();
    h=strt;
    for(int i=strt;i<x;i++)
    {
        System.out.println("Sending frame:"+h);
        h=(h+1)%x;
        System.out.println();
        oos.writeObject(i);
        oos.writeObject(data[i]);
        Thread.sleep(10);
    }
    listener.t.join(3500);
    if(listener.reply!=x-1)
    {
        System.out.println("No reply from server in 3.5 seconds.
Resending data from frame no " + (listener.reply+1));
        System.out.println();
        strt=listener.reply+1;
        flag=false;
    }
    else
    {
```

```

        System.out.println("All elements sent successfully. Exiting");
        flag=true;
    }
    }while(!flag);
    oos.writeObject(-1);
}
}
class GoBackNListener implements Runnable
{
    Thread t;
    ObjectInputStream ois;
    int reply,x;
    GoBackNListener(ObjectInputStream o,int i)
    {
        t=new Thread(this);
        ois=o;
        reply=-2;
        x=i;
    }
    @Override
    public void run()
    {
        try
        {
            int temp=0;
            while(reply!=-1)
            {
                reply=(Integer)ois.readObject();
                if(reply!=-1 && reply!=temp+1)
                    reply=temp;
                if(reply!=-1)
                {
                    temp=reply;
                    System.out.println("Acknowledgement of frame no " + (reply%x)
                    + " recieved.");
                    System.out.println();
                }
            }
            reply=temp;
        }
        catch(Exception e)
        {

```

```

        System.out.println("Exception => " + e);
    }
}

```

- **Server:**

```

import java.net.*;
import java.io.*;
import java.util.*;
public class Server
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket server=new ServerSocket(6262);
        System.out.println("Server established.");
        Socket client=server.accept();
        ObjectOutputStream oos=new ObjectOutputStream(client.getOutputStream());
        ObjectInputStream ois=new ObjectInputStream(client.getInputStream());
        System.out.println("Client is now connected.");
        int x=(Integer)ois.readObject();
        int k=(Integer)ois.readObject();
        int j=0;
        int i=(Integer)ois.readObject();
        boolean flag=true;
        Random r=new Random(6);
        int mod=r.nextInt(6);
        while(mod==1||mod==0)
        {
            mod=r.nextInt(6);
        }
        while(true)
        {
            int c=k;
            for(int h=0;h<=x;h++)
            {
                System.out.print("|"+c+"|");
                c=(c+1)%x;
            }
            System.out.println();
            System.out.println();
            if(k==j)
            {
                System.out.println("Frame "+k+" recieved"+"\\n"+"Data:"+j);
                j++;
                System.out.println();
            }
        }
    }
}

```

```

    }
    else
        System.out.println("Frames recieved not in correct order"+"\\n"+"
Expected farne:" + j +"\\n"+ " Recieved frame no :"+ k);
    System.out.println();
    if(j%mod==0 && flag)
    {
        System.out.println("Error found. Acknowledgement not sent. ");
        flag=!flag;
        j--;
    }
    else if(k==j-1)
    {
        oos.writeObject(k);
        System.out.println("Acknowledgement sent");
    }
    System.out.println();
    if(j%mod==0)
        flag=!flag;
        k=(Integer)ois.readObject();
    if(k==-1)
        break;
    i=(Integer)ois.readObject();
}
System.out.println("Client finished sending data. Exiting");
oos.writeObject(-1);
}
}

```

Output

Client:

Enter the value of m : 7
 Enter no. of frames to be sent:5
 Enter data for frame no 0 => 1
 Enter data for frame no 1 => 2
 Enter data for frame no 2 => 3
 Enter data for frame no 3 => 4
 Enter data for frame no 4 => 5
 Connected with server.
 |0||1||2||3||4|

Sending frame:0

Sending frame:1

Acknowledgement of frame no 0 recieved.

Sending frame:2

Sending frame:3

Sending frame:4

Sending frame:5

Server:

Server established.

Client is now connected.

0||1||2||3||4||5||6||7||8||9||10||11||12||13||14||15||16||17||18||19||20||21||22||23||24||25||26||27||28||29||30||
31||32||33||34||35||36||37||38||39||40||41||42||43||44||45||46||47||48||49||50||51||52||53||54||55||56||57||58||
59||60||61||62||63||64||65||66||67||68||69||70||71||72||73||74||75||76||77||78||79||80||81||82||83||84||85||8
6||87||88||89||90||91||92||93||94||95||96||97||98||99||100||101||102||103||104||105||106||107||108||109||1
10||111||112||113||114||115||116||117||118||119||120||121||122||123||124||125||126||0|

Frame 0 recieved

Data:0

Acknowledgement sent

1||2||3||4||5||6||7||8||9||10||11||12||13||14||15||16||17||18||19||20||21||22||23||24||25||26||27||28||29||30||31||
32||33||34||35||36||37||38||39||40||41||42||43||44||45||46||47||48||49||50||51||52||53||54||55||56||57||58||5
9||60||61||62||63||64||65||66||67||68||69||70||71||72||73||74||75||76||77||78||79||80||81||82||83||84||85||86||
87||88||89||90||91||92||93||94||95||96||97||98||99||100||101||102||103||104||105||106||107||108||109||110||
111||112||113||114||115||116||117||118||119||120||121||122||123||124||125||126||0||1|

Frame 1 recieved

Data:1

Error found. Acknowledgement not sent.

2||3||4||5||6||7||8||9||10||11||12||13||14||15||16||17||18||19||20||21||22||23||24||25||26||27||28||29||30||31||32||33||34||35||36||37||38||39||40||41||42||43||44||45||46||47||48||49||50||51||52||53||54||55||56||57||58||59||60||61||62||63||64||65||66||67||68||69||70||71||72||73||74||75||76||77||78||79||80||81||82||83||84||85||86||87||88||89||90||91||92||93||94||95||96||97||98||99||100||101||102||103||104||105||106||107||108||109||110||111||112||113||114||115||116||117||118||119||120||121||122||123||124||125||126||0||1||2|

Frames recieved not in correct order

Expected farme:1

Recieved frame no :2

3||4||5||6||7||8||9||10||11||12||13||14||15||16||17||18||19||20||21||22||23||24||25||26||27||28||29||30||31||32||33||34||35||36||37||38||39||40||41||42||43||44||45||46||47||48||49||50||51||52||53||54||55||56||57||58||59||60||61||62||63||64||65||66||67||68||69||70||71||72||73||74||75||76||77||78||79||80||81||82||83||84||85||86||87||88||89||90||91||92||93||94||95||96||97||98||99||100||101||102||103||104||105||106||107||108||109||110||111||112||113||114||115||116||117||118||119||120||121||122||123||124||125||126||0||1||2||3|

Frames recieved not in correct order

Expected farme:1

Recieved frame no :3

4||5||6||7||8||9||10||11||12||13||14||15||16||17||18||19||20||21||22||23||24||25||26||27||28||29||30||31||32||33||34||35||36||37||38||39||40||41||42||43||44||45||46||47||48||49||50||51||52||53||54||55||56||57||58||59||60||61||62||63||64||65||66||67||68||69||70||71||72||73||74||75||76||77||78||79||80||81||82||83||84||85||86||87||88||89||90||91||92||93||94||95||96||97||98||99||100||101||102||103||104||105||106||107||108||109||110||111||112||113||114||115||116||117||118||119||120||121||122||123||124||125||126||0||1||2||3||4|

Frames recieved not in correct order

Expected farme:1

Recieved frame no :4

C) Aim

To implement the Selective Repeat ARQ protocol using java programming language.

Theoretical Background

Selective repeat protocol, also called Selective Repeat ARQ (Automatic Repeat reQuest), is a data link layer protocol that uses sliding window method for reliable delivery of data frames.

Here, only the erroneous or lost frames are retransmitted, while the good frames are received and buffered. Selective Repeat ARQ is a data link layer protocol that uses a sliding window method. The Go-back-N ARQ protocol works well if it has fewer errors. But if there is a lot of error in the frame, lots of bandwidth loss in sending the frames again. So, we use the Selective Repeat ARQ protocol. In this protocol, the size of the sender window is always equal to the size of the receiver window. The size of the sliding window is always greater than 1. If the receiver receives a corrupt frame, it does not directly discard it. It sends a negative acknowledgment to the sender. The sender sends that frame again as soon as on the receiving negative acknowledgment. There is no waiting for any time-out to send that frame.

Algorithm

1. Start.
2. Establish connection (recommended UDP)
3. Accept the window size from the client(should be ≤ 40)
4. Accept the packets from the network layer.
5. Calculate the total frames/windows required.
6. Send the details to the client(total packets, total frames.)
7. Initialize the transmit buffer.
8. Built the frame/window depending on the window size.
9. Transmit the frame.
10. Wait for the acknowledgement frame.
11. Check for the acknowledgement of each packet and repeat the process for the packet for which the negative acknowledgement is received.
Else continue as usual.
12. Increment the frame count and repeat steps 7 to 12 until all packets are transmitted.
13. Close the connection.
14. Stop.

Program

Client:

```
import java.lang.System;
import java.net.*;
import java.io.*;
import java.text.*;
import java.util.Random;
```

```
import java.util.*;

public class cli
{
    static Socket connection;
    public static void main(String a[]) throws SocketException
    {
        try {
            int v[] = new int[10];
            int n = 0;
            Random rand = new Random();
            int rand = 0;
            InetAddress addr = InetAddress.getByName("Localhost");
            System.out.println(addr);
            connection = new Socket(addr, 8011);
            DataOutputStream out = new DataOutputStream(
                connection.getOutputStream());
            DataInputStream in = new DataInputStream(
                connection.getInputStream());
            int p = in.read();
            System.out.println("No of frame is:" + p);
            for (int i = 0; i < p; i++) {
                v[i] = in.read();
                System.out.println(v[i]);
                //g[i] = v[i];
            }
            rand = rand.nextInt(p); //FRAME NO. IS RANDOMLY GENERATED
            v[rand] = -1;
            for (int i = 0; i < p; i++)
            {
                System.out.println("Received frame is: " + v[i]);
            }
            for (int i = 0; i < p; i++)
                if (v[i] == -1)
                {
                    System.out.println("Request to retransmit from packet no "
                        + (i+1) + " again!!");
                    n = i;
                    out.write(n);
                    out.flush();
                }
            }
        }
    }
}
```

```

        System.out.println();
        v[n] = in.read();
        System.out.println("Received frame is: " + v[n]);
        System.out.println("quiting");
    } catch (Exception e) {
        System.out.println(e);
    }
}
}

```

Server:

```

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

public class ser
{
    static ServerSocket Serversocket;
    static DataInputStream dis;
    static DataOutputStream dos;
    public static void main(String[] args) throws SocketException
    {
        try
        {
            int a[] = { 30, 40, 50, 60, 70, 80, 90, 100 };
            Serversocket = new ServerSocket(8011);
            System.out.println("waiting for connection");
            Socket client = Serversocket.accept();
            dis = new DataInputStream(client.getInputStream());
            dos = new DataOutputStream(client.getOutputStream());
            System.out.println("The number of packets sent is:" + a.length);
            int y = a.length;
            dos.write(y);
            dos.flush();
            for (int i = 0; i < a.length; i++)
            {
                dos.write(a[i]);
                dos.flush();
            }
        }
    }
}

```

```
        }
        int k = dis.read();
        dos.write(a[k]);
        dos.flush();
    }
    catch (IOException e)
    {
        System.out.println(e);
    }
    finally
    {
        try
        {
            dis.close();
            dos.close();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

Output:**Client:**

Localhost/127.0.0.1

No of frame is:8

30

40

50

60

70

80

90

100

Received frame is: 30

Received frame is: -1

Received frame is: 50

Received frame is: 60
Received frame is: 70
Received frame is: 80
Received frame is: 90
Received frame is: 100
Request to retransmit from packet no 2 again!!

Received frame is: 40
quitting

Server:

waiting for connection
The number of packets sent is:8

7. Distance Vector Routing Algorithm

7.1 Aim:

To implement and simulate algorithm for Distance Vector Routing protocol.

7.2 Theoretical Background

Distance Vector Routing protocol is a ‘dynamic routing’ protocol. With this protocol, every router in the network creates a routing table which helps them in determining the **shortest path** through the network. All the routers in the network are aware of every other router in the network and they keep on updating their routing table **periodically**. This protocol uses the principle of **Bellman-Ford’s** algorithm. The Bellman Ford algorithm means each router maintains a Distance Vector table containing the distance between itself and all possible destination nodes. Distances, based on a chosen metric, are computed using information from the neighbors’ distance vectors.

In DVR, each router maintains a routing table. It contains only one entry for each router. It contains two parts – a preferred outgoing line to use for that destination and an estimate of time (delay). Tables are updated by exchanging the information with the neighbor’s nodes. Each router knows the delay in reaching its neighbors (Ex – send echo request). Routers periodically exchange routing tables with each of their neighbors. It compares the delay in its local table with the delay in the neighbor’s table and the cost of reaching that neighbor. If the path via the neighbor has a lower cost, then the router updates its local table to forward packets to the neighbor.

7.3 Algorithm

- 1.A router transmits its distance vector to each of its neighbors in a routing packet.
- 2.Each router receives and saves the most recently received distance vector from each of its neighbors.
- 3.A router recalculates its distance vector when:
 - It receives a distance vector from a neighbor containing different information than before.
 - It discovers that a link to a neighbor has gone down.

7.4 Program

```
import java.io.*;
public class DVR
{
    static int graph[][];
    static int via[][];
    static int rt[][];
    static int v;
    static int e;

    public static void main(String args[]) throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Please enter the number of Vertices: ");
        v = Integer.parseInt(br.readLine());
        System.out.println("Please enter the number of Edges: ");
        e = Integer.parseInt(br.readLine());
        graph = new int[v][v];
        via = new int[v][v];
        rt = new int[v][v];
        for(int i = 0; i < v; i++)
            for(int j = 0; j < v; j++)
            {
                if(i == j)
                    graph[i][j] = 0;
                else
                    graph[i][j] = 9999;
            }
        for(int i = 0; i < e; i++)
        {
            System.out.println("Please enter data for Edge " + (i + 1) + ":");
            System.out.print("Source: ");
            int s = Integer.parseInt(br.readLine());
            s--;
            System.out.print("Destination: ");
            int d = Integer.parseInt(br.readLine());
        }
    }
}
```

```

        d--;
        System.out.print("Cost: ");
        int c = Integer.parseInt(br.readLine());
        graph[s][d] = c;
        graph[d][s] = c;
    }

    dvr_calc_disp("The initial Routing Tables are: ");
    System.out.print("Please enter the Source Node for the edge whose cost
has changed: ");
    int s = Integer.parseInt(br.readLine());
    s--;
    System.out.print("Please enter the Destination Node for the edge whose
cost has changed: ");
    int d = Integer.parseInt(br.readLine());
    d--;
    System.out.print("Please enter the new cost: ");
    int c = Integer.parseInt(br.readLine());
    graph[s][d] = c;
    graph[d][s] = c;
    dvr_calc_disp("The new Routing Tables are: ");
}
static void dvr_calc_disp(String message)
{
    System.out.println();
    init_tables();
    update_tables();
    System.out.println(message);
    print_tables();
    System.out.println();
}
static void update_table(int source)
{
    for(int i = 0; i < v; i++)
    {
        if(graph[source][i] != 9999)

```



```
        {
            int dist = graph[source][i];
            for(int j = 0; j < v; j++)
            {
                int inter_dist = rt[i][j];
                if(via[i][j] == source)
                    inter_dist = 9999;
                if(dist + inter_dist < rt[source][j])
                {
                    rt[source][j] = dist + inter_dist;
                    via[source][j] = i;
                }
            }
        }
    }
}

static void update_tables()
{
    int k = 0;
    for(int i = 0; i < 4*v; i++)
    {
        update_table(k);
        k++;
        if(k == v)
            k = 0;
    }
}

static void init_tables()
{
    for(int i = 0; i < v; i++)
    {
        for(int j = 0; j < v; j++)
        {
            if(i == j)
            {
                rt[i][j] = 0;
            }
        }
    }
}
```

```
                via[i][j] = i;
            }
            else
            {
                rt[i][j] = 9999;
                via[i][j] = 100;
            }
        }
    }
}

static void print_tables()
{
    for(int i = 0; i < v; i++)
    {
        for(int j = 0; j < v; j++)
        {
            System.out.print("Dist: " + rt[i][j] + "   ");
        }
        System.out.println();
    }
}
}
```

7.5 Output

Please enter the number of Vertices:

4

Please enter the number of Edges:

5

Please enter data for Edge 1:

Source: 1

Destination: 2

Cost: 1

Please enter data for Edge 2:

Source: 1

Destination: 3

Cost: 3

Please enter data for Edge 3:

Source: 2

Destination: 3

Cost: 1

Please enter data for Edge 4:

Source: 2

Destination: 4

Cost: 1

Please enter data for Edge 5:

Source: 3

Destination: 4 Cost: 4

The initial Routing Tables are:

Dist: 0	Dist: 1	Dist: 2	Dist: 2
---------	---------	---------	---------

Dist: 1	Dist: 0	Dist: 1	Dist: 1
---------	---------	---------	---------

Dist: 2	Dist: 1	Dist: 0	Dist: 2
---------	---------	---------	---------

Dist: 2	Dist: 1	Dist: 2	Dist: 0
---------	---------	---------	---------

Please enter the Source Node for the edge whose cost has changed: 2

Please enter the Destination Node for the edge whose cost has changed: 4 Please enter the new cost: 10

The new Routing Tables are:

Dist: 0	Dist: 1	Dist: 2	Dist: 6
---------	---------	---------	---------

Dist: 1	Dist: 0	Dist: 1	Dist: 5
---------	---------	---------	---------

Dist: 2	Dist: 1	Dist: 0	Dist: 4
---------	---------	---------	---------

8. SMTP using TCP

9.1 Aim:

To implement SMTP.

9.2 Theoretical Background:

Most of the internet systems use SMTP as a method to transfer mail from one user to another. SMTP is a push protocol and is used to send the mail whereas POP (post office protocol) or IMAP (internet message access protocol) are used to retrieve those mails at the receiver's side. SMTP is an application layer protocol. The client who wants to send the mail opens a TCP connection to the SMTP server and then sends the mail across the connection. The SMTP server is always on listening mode. As soon as it listens for a TCP connection from any client, the SMTP process initiates a connection on that port (25). After successfully establishing the TCP connection the client process sends the mail instantly.

9.3 Algorithm:

- **Client**

1. Create client socket.
2. Accept username and password from user and send to client's output stream.
3. Get response from server to 'response'.
4. If response= "success" then
 5. start readmessage thread.
 6. start createmessage thread.
7. else
8. Print "Login failed".

Thread_write_message

1. Read message from user.
 2. Write mail to server.

Thread_read_message

1. Read mail from user
2. Print mail.

- **Server**

1. Create server socket.
2. Get username and password.
3. If valid then
4. send response "success" to client.

5. accept mail from client mail.
6. create client socket for server2.
7. send mail to server2
8. If mail received from server2 then
9. send mail to client

9.4 Program:

- SMTP Server1

```
import java.io.*;
import java.net.*;
public class Server1 extends Thread
{
    int flag;
    Thread t;
    ServerSocket svr;
    Socket client;
    public static Clientthreads[] ths=new Clientthreads[15];
    public static ServerSocket mm;
    public static Socket cmm;
    public Server1(int portno) //constructor
    {
        try
        {
            mm=new ServerSocket(4444);
            svr=new ServerSocket(portno);
            for(int i=0;i<15;i++)
                ths[i]=null;
        }
        catch(Exception e)
        {
            System.out.println("error :"+e);
        }
    }
    public void run()
    {
        try
        {
            int i=0;
            cmm=mm.accept();
            System.out.println("Connected to yahoo server");
            while(true)
```

```

        {
            client=svr.accept();
            BufferedReader br=new BufferedReader(new
InputStreamReader(client.getInputStream()));
            PrintWriter pout=new PrintWriter(client.getOutputStream(),true);
            String a=br.readLine();
            BufferedReader fin=new BufferedReader(new
FileReader("login1.txt"));
            String line;
            while((line=fin.readLine())!=null)
            {
                if(a.equals(line))
                {
                    pout.println("1");    //login successful
                    flag=1;
                    break;
                }
            }
            if(flag!=1)
            {
                pout.println("0");
                client.close();
                continue;
            }
            if(th[s[i]]==null)
            {
                th[s[i]]=new Clientthreads(client,th,s[i],cmm);
                th[s[i]].t1=new Thread(th[s[i]]);
                th[s[i]].t2=new Thread(th[s[i]]);
                th[s[i]].t1.start();
                th[s[i]].t2.start();
                i++;
            }
        }
    }
    catch(Exception e)
    {
        System.out.println("Error !:"+e);
    }
}

```

```
    }
    public static void main(String args[])
    {
        Server1 s=new Server1(2222);
        s.t=new Thread(s);
        s.t.start();
    }
}

class Clientthreads extends Thread
{
    BufferedReader buf;
    PrintWriter out;
    String str,a;
    String sendmsg[];
    Socket client;
    Clientthreads[] ths;
    int id;
    Thread t1,t2;
    public static String[] names=new String[15];
    public static Socket cmm;
    public static BufferedReader bmm;
    public static PrintWriter pmm;
    public Clientthreads(Socket client,Clientthreads[] threads,int i,Socket cmm)
    {
        try
        {
            this.client=client;
            ths=threads;
            id=i;
            this.cmm=cmm;
            bmm=new BufferedReader(new InputStreamReader(cmm.getInputStream()));
            pmm=new PrintWriter(cmm.getOutputStream(),true);
            buf=new BufferedReader(new InputStreamReader(client.getInputStream()));
            out=new PrintWriter(client.getOutputStream(),true);
        }
    }
}
```

```
        catch(Exception e)
        {
            System.out.println("Error :"+e);
        }
    }
    public void run()
    {
        if(Thread.currentThread()==t1)
        {
            while(true)
            {
                try
                {
                    str=buf.readLine();
                    if(str.equals("logout"))
                    {
                        ths[id]=null;
                        break;
                    }
                    pmm.println(str);
                }
                catch(Exception e)
                {
                    System.out.println("error! :"+e);
                }
            }
        }
        else
        {
            while(true)
            {
                try
                {
                    a=bmm.readLine();
                    for(int i=0;ths[i]!=null;i++)
                    {
                        ths[i].out.println(a);
                    }
                }
            }
        }
    }
}
```



```

    }
    catch(Exception e)
    {
        System.out.println("error!! :"+e);
    }
}
}
}
}
}

```

- SMTP Client1

```

import java.io.*;
import java.net.*;
public class Client1 extends Thread
{
    public static int i;
    Socket sock;
    Thread t1,t2;
    BufferedReader br;
    String line,usr,pwd;
    BufferedReader buff;
    PrintWriter pout;
    public Client1()
    {
        try
        {
            sock=new Socket("127.0.0.1",2222);
            System.out.println("server connected");
            br=new BufferedReader(new InputStreamReader(sock.getInputStream()));
            buff=new BufferedReader(new InputStreamReader(System.in));
            pout=new PrintWriter(sock.getOutputStream(),true);
            System.out.println("enter username : ");
            usr=buff.readLine();
            System.out.println("enter password : ");
            pwd=buff.readLine();
            pout.println(usr+"//"+pwd);
            line=br.readLine();

```

```
        i=Integer.valueOf(line);
        if(i==1)
            System.out.println("Login Successful!!\n");
        else
        {
            System.out.println("Login failed");
            sock.close();
        }
    }
    catch(IOException e)
    {
        System.err.println("error"+e);
    }
}
public void run()
{
    if(Thread.currentThread()==t1)
    {
        while(true)
        {
            try
            {
                line=br.readLine();    //read from socket
                String rcmsg[]=line.split("/");
                System.out.println("\nIncoming message!!\n");
                System.out.println("To :"+rcmsg[0]);
                System.out.println("From :"+rcmsg[1]);
                System.out.println("Message :"+rcmsg[2]);
            }
            catch(Exception e)
            {
                System.err.println("error"+e);
            }
        }
    }
    else
```

```

        {
            while(true)
            {
                try
                {
                    String to,from,msg;
                    System.out.println("\nenter mail to send a message");
                    String str=buff.readLine(); //to read from terminal
                    if(str.equals("mail"))
                    {
                        System.out.println("To :");
                        to=buff.readLine();
                        System.out.println("From :");
                        from=buff.readLine();

                        System.out.println("Message :");
                        msg=buff.readLine();
                        pout.println(to+"//"+from+"//"+msg); //to write to
socket
                    }
                    else if(str.equals("logout"))
                    {
                        pout.println(str);
                        t1=null;
                        t2=null;
                        break;
                    }
                }
                catch(Exception e)
                {
                    System.err.println("error"+e);
                }
            }
        }
    }

    public static void main(String args[])
    {

```

```

        Client1 ss=new Client1();
        if(i==1)          //if logged in
        {
            ss.t1=new Thread(ss);
            ss.t2=new Thread(ss);
            ss.t1.start();
            ss.t2.start();
        }
    }
}

```

- login1.txt

```

dolly@gmail.com//dolly
rohan@gmail.com//rohan
nikki@gmail.com//nikki
sunny@gmail.com//sunny

```

- login2.txt

```

chandler@yahoo.com//chandler
monica@yahoo.com//monica
phoebe@yahoo.com//phoebe
rachel@yahoo.com//rachel
ross@yahoo.com//ross
joey@yahoo.com//joey

```

- SMTP Server2

```

import java.io.*;
import java.net.*;
public class Server2 extends Thread
{
    int flag;
    Thread t;
    ServerSocket svr;
    Socket client;

```

```
public static Clientthreads[] ths=new Clientthreads[15];
public static Socket cmm;
public Server2(int portno) //constructor
{
    try
    {
        svr=new ServerSocket(portno);
        cmm=new Socket("127.0.0.1",4444);
        System.out.println("Connected to gmail server");
        for(int i=0;i<15;i++)
            ths[i]=null;
    }
    catch(Exception e)
    {
        System.out.println("error :"+e);
    }
}
public void run()
{
    try
    {
        int i=0;
        while(true)
        {
            client=svr.accept();
            BufferedReader br=new BufferedReader(new
InputStreamReader(client.getInputStream()));
            PrintWriter pout=new PrintWriter(client.getOutputStream(),true);
            String a=br.readLine();
            BufferedReader fin=new BufferedReader(new FileReader("login2.txt"));
            String line;
            while((line=fin.readLine())!=null)
            {
                if(a.equals(line))
                {
                    pout.println("1");    //login successful
                    flag=1;
                    break;
                }
            }
        }
    }
}
```

```

        }
        if(flag!=1)
        {
            pout.println("0");
            client.close();
            continue;
        }

        if(th[s[i]]==null)
        {
            th[s[i]]=new Clientthreads(client,th,s,i,cmm);
            th[s[i]].t1=new Thread(th[s[i]]);
            th[s[i]].t2=new Thread(th[s[i]]);
            th[s[i]].t1.start();
            th[s[i]].t2.start();
            i++;
        }
    }
}
catch(Exception e)
{
    System.out.println("Error !!" + e);
}
}
public static void main(String args[])
{
    Server2 s=new Server2(6666);
    s.t=new Thread(s);
    s.t.start();
}
}

```

```

class Clientthreads extends Thread
{

```

```

    BufferedReader buf;
    PrintWriter out;
    String str,a;
    String sendmsg[];

```

```
Socket client;
Clientthreads[] ths;
int id;
Thread t1,t2;
public static String[] names=new String[15];
public static Socket cmm;
public static BufferedReader bmm;
public static PrintWriter pmm;
public Clientthreads(Socket client,Clientthreads[] threads,int i,Socket cmm)
{
    try
    {
        this.client=client;
        ths=threads;
        id=i;
        this.cmm=cmm;
        bmm=new BufferedReader(new InputStreamReader(cmm.getInputStream()));
        pmm=new PrintWriter(cmm.getOutputStream(),true);
        buf=new BufferedReader(new InputStreamReader(client.getInputStream()));
        out=new PrintWriter(client.getOutputStream(),true);
    }
    catch(Exception e)
    {
        System.out.println("Error :"+e);
    }
}
public void run()
{
    if(Thread.currentThread()==t1)
    {
        while(true)
        {
            try
            {
                str=buf.readLine();
                if(str.equals("logout"))
                {
```

```

                ths[id]=null;
                break;
            }
            pmm.println(str);
        }
        catch(Exception e)
        {
            System.out.println("error :"+e);
        }
    }
}
else
{
    while(true)
    {
        try
        {
            a=bmm.readLine();
            for(int i=0;ths[i]!=null;i++)
            {
                ths[i].out.println(a);
            }
        }
        catch(Exception e)
        {
            System.out.println("error :"+e);
        }
    }
}
}
}

```

- SMTP Client2

```

import java.io.*;
import java.net.*;
public class Client2 extends Thread
{
    public static int i;

```



```
Socket sock;
Thread t1,t2;
BufferedReader br;
String line,usr,pwd;
BufferedReader buff;
PrintWriter pout;
public Client2()
{
    try
    {
        sock=new Socket("127.0.0.1",6666);
        System.out.println("server connected");
        br=new BufferedReader(new InputStreamReader(sock.getInputStream()));
        buff=new BufferedReader(new InputStreamReader(System.in));
        pout=new PrintWriter(sock.getOutputStream(),true);
        System.out.println("enter username : ");
        usr=buff.readLine();
        System.out.println("enter password : ");
        pwd=buff.readLine();
        pout.println(usr+"/"+pwd);
        line=br.readLine();
        i=Integer.valueOf(line);
        if(i==1)
            System.out.println("Login Successful!!\n");
        else
        {
            System.out.println("Login failed");
        }
    }
    catch(IOException e)
    {
        System.err.println("error"+e);
    }
}
public void run()
{
    if(Thread.currentThread()==t1)
```

```
{
    while(true)
    {
        try
        {
            line=br.readLine();    //read from socket
            String rcmsg[];
            rcmsg=line.split("/");
            System.out.println("\nIncoming message!!\n");
            System.out.println("To :"+rcmsg[0]);
            System.out.println("From :"+rcmsg[1]);
            System.out.println("Message :"+rcmsg[2]);
        }
        catch(Exception e)
        {
            System.err.println("error"+e);
        }
    }
}
else
{
    while(true)
    {
        try
        {
            String to,from,msg;
            System.out.println("\nenter mail to send a message");
            String str=buff.readLine(); //to read from terminal
            if(str.equals("mail"))
            {
                System.out.println("To :");
                to=buff.readLine();
                System.out.println("From :");
                from=buff.readLine();

                System.out.println("Message :");
                msg=buff.readLine();
            }
        }
        catch(Exception e)
        {
            System.err.println("error"+e);
        }
    }
}
```

```

                                pout.println(to+"/"+from+"/"+msg); //to write to
socket
                                }
                                else if(str.equals("logout"))
                                {
                                    pout.println(str);
                                    t1=null;
                                    t2=null;
                                    break;
                                }
                                }
                                catch(Exception e)
                                {
                                    System.err.println("error"+e);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    public static void main(String args[])
    {
        Client2 ss=new Client2();
        if(i==1) //if logged in
        {
            ss.t1=new Thread(ss);
            ss.t2=new Thread(ss);
            ss.t1.start();
            ss.t2.start();
        }
    }
}

```

9.5 Output

Server 1 mtechstudents@admin1-System- Product-Name:~/smtp\$ java Server1 Connected to yahoo server	Server 2 mtechstudents@admin1-System- Product-Name:~/smtp\$ java Server1 Connected to gmail server
Client 1 mtechstudents@admin1-System- Product-Name:~/smtp\$ java Client1 server connected enter username : dolly@gmail.com enter password : dolly Login Successful!! enter mail to send a message mail To : ross@yahoo.com From : dolly@gmail.com Message : How are u?	Client 2 mtechstudents@admin1-System- Product-Name:~/smtp\$ java Client2 server connected enter username : ross@yahoo.com enter password : ross Login Successful!! Incoming message!! To :ross@yahoo.com From :dolly@gmail.com Message :How are u? enter mail to send a message

9. FTP- File Transfer Protocol

9.1 Aim:

Develop concurrent file server which will provide the file requested by client if it exists. If not server sends appropriate message to the client. Server should also send its thread ID to clients for display along with file or the message.

9.2 Theoretical Background:

FTP is a protocol for transferring files between local hosts and remote hosts. The user at the local host provides the host name of the remote host to the FTP client process on the local host. This creates a connection between local host client process and remote host server process. The user then provides identification and password to the server which authorizes the user. Now files can be transferred between hosts. FTP uses two parallel connections to transfer a file. One connection is called control connection and other connection is called the data connection. The commands get and put are sent over FTP control connection. The data connection is used for sending the files. When the user starts FTP session, the client establishes the control connection with the remote server. When the server gets a command to transfer a file, the server establishes a data connection. After the file transfer, the data connection is closed. To send another file, a new data connection is established. So control connection is persistent during entire FTP session. However the data connection breaks after each data transfer. When a local machine is connected with remote server through FTP, it will receive a terminal of server. Then the communication is based on set of FTP commands. The following are a sublist of popular FTP commands.

9.3 Algorithm:

- **Client Connection**

1. Read input from user
2. If input == '1'.
3. call receivefile()
4. Else
5. If input == '2'
6. call sendfile(filename)
7. Search file name.
8. If exists, convert it into bytes.
9. send to output stream.
10. sendfile()

11. open the file with filename.
12. convert into byte array
13. Else
14. Error

- **File Server**

1. Create server socket.
2. Listen for client.
3. If new client received, create a new thread.
4. Start the thread.

- **File Client**

1. Create client socket.
2. Read integer from the console
3. If input == '1'
4. sendfile()
5. If input == '2'
6. readfilename()
7. receive file.

9.4 Program

- **Client Connection**

```
import java.net.*;
import java.io.*;
import java.util.*;
public class ClientConnection implements Runnable
{
    private Socket clientSocket;
    private int id;
    private BufferedReader in = null;
    public ClientConnection(Socket client, int id)
    {
        this.clientSocket = client;
        this.id = id;
    }
}
```

```
    }

    @Override
    public void run()
    {
        try
        {
            in = new BufferedReader(new InputStreamReader(
clientSocket.getInputStream()));
            String clientSelection;
            while ((clientSelection = in.readLine()) != null)
            {
                switch (clientSelection)
                {
                    case "1": receiveFile();
                    break;
                    case "2": String outGoingFileName;
                        while ((outGoingFileName = in.readLine()) != null)
                        {
                            sendFile(outGoingFileName);
                        } break;
                    default: System.out.println("Incorrect command
received.");

                        break;
                }
                in.close();
                break;
            }
        }
        catch (IOException ex)
        {

            //Logger.getLogger(ClientConnection.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    public void receiveFile()
```

```
{
    try
    {
        int bytesRead;
        DataInputStream clientData = new
DataInputStream(clientSocket.getInputStream());
        String fileName = clientData.readUTF();
        OutputStream output = new FileOutputStream(("received_from_client_" +
fileName));

        long size = clientData.readLong();
        byte[] buffer = new byte[1024];
        while (size > 0 && (bytesRead = clientData.read(buffer, 0, (int)
Math.min(buffer.length, size))) != -1)
        {
            output.write(buffer, 0, bytesRead);
            size -= bytesRead;
        }
        output.close();
        clientData.close();
        System.out.println("File "+fileName+" received from client.");
    }
    catch (IOException ex)
    {
        System.err.println("Client error. Connection closed.");
    }
}

public void sendFile(String fileName)
{
    try
    {
        OutputStream os = clientSocket.getOutputStream();
        //Sending file name and file size to the server
        DataOutputStream dos = new DataOutputStream(os);
        //handle file read
        try {
            File myFile = new File(fileName);
```



```
byte[] mybytearray = new byte[(int) myFile.length()];
FileInputStream fis = new FileInputStream(myFile);
BufferedInputStream bis = new BufferedInputStream(fis);
//bis.read(mybytearray, 0, mybytearray.length);
DataInputStream dis = new DataInputStream(bis);
dis.readFully(mybytearray, 0, mybytearray.length);
//handle file send over socket
dos.writeInt(200);
dos.writeUTF(myFile.getName());
dos.writeLong(mybytearray.length);
dos.write(mybytearray, 0, mybytearray.length);
dos.flush();
System.out.println("File "+fileName+" sent to client.");
}
catch(FileNotFoundException e) {
    dos.writeInt(404);
    dos.writeInt(id);
    System.err.print("File not found!");
}
catch(IOException e) {
    e.printStackTrace();
}
}
catch (IOException e)
{
    e.printStackTrace();
}
}
```

- File Server

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

public class FileServer
```

```

{
    private static ServerSocket serverSocket;
    private static Socket clientSocket = null;
    public static void main(String[] args) throws IOException
    {
        Random r = new Random();
        try
        {
            serverSocket = new ServerSocket(4444);
            System.out.println("Server started.");
        }
        catch (Exception e)
        {
            System.err.println("Port already in use.");
            System.exit(1);
        }
        while (true)
        {
            try
            {
                clientSocket = serverSocket.accept();
                System.out.println("Accepted connection : " + clientSocket);
                Thread t = new Thread(new ClientConnection(clientSocket,
r.nextInt()));
                t.start();
            }
            catch (Exception e)
            {
                System.err.println("Error in connection attempt.");
            }
        }
    }
}

```

- File Client

```
import java.net.*;
import java.util.*;
import java.io.*;
public class FileClient {
    private static Socket sock;
    private static String fileName;
    private static BufferedReader stdin;
    private static PrintStream os;
    public static void main(String[] args) throws IOException
    {
        try
        {
            sock = new Socket("localhost", 4444);
            stdin = new BufferedReader(new InputStreamReader(System.in));
        }
        catch (Exception e) {
            System.err.println("Cannot connect to the server, try again later.");
            System.exit(1);
        }
        os = new PrintStream(sock.getOutputStream());
        try
        {
            switch (Integer.parseInt(selectAction())) {
                case 1: os.println("1");
                    sendFile();
                    break;
                case 2: os.println("2");
                    System.err.print("Enter file name: ");
                    fileName = stdin.readLine();
                    os.println(fileName);
                    receiveFile(fileName);
                    break;
            }
        }
        catch (Exception e)
        {

```

```
        System.err.println("not valid input");
    }
    sock.close();
}
public static String selectAction() throws IOException {
    System.out.println("1. Send file.");
    System.out.println("2. Recieve file.");
    System.out.print("\nMake selection: ");
    return stdin.readLine();
}
public static void sendFile()
{
    try
    {
        System.out.println("Enter file name: ");
        fileName = stdin.readLine();
        File myFile = new File(fileName);
        byte[] mybytearray = new byte[(int) myFile.length()];
        FileInputStream fis = new FileInputStream(myFile);
        BufferedInputStream bis = new BufferedInputStream(fis);
        //bis.read(mybytearray, 0, mybytearray.length);
        DataInputStream dis = new DataInputStream(bis);
        dis.readFully(mybytearray, 0, mybytearray.length);
        OutputStream os = sock.getOutputStream();
        //Sending file name and file size to the server
        DataOutputStream dos = new DataOutputStream(os);
        dos.writeUTF(myFile.getName());
        dos.writeLong(mybytearray.length);
        dos.write(mybytearray, 0, mybytearray.length);
        dos.flush();
        System.out.println("File "+fileName+" sent to Server.");
    }
    catch (Exception e) {
        System.err.println("File does not exist!");
    }
}
```

```
public static void receiveFile(String fileName)
{
    try
    {
        int bytesRead;
        InputStream in = sock.getInputStream();
        DataInputStream clientData = new DataInputStream(in);
        int resp = clientData.readInt();
        if(resp==200) {

            fileName = clientData.readUTF();
            OutputStream output = new FileOutputStream(("received_from_server_"
+ fileName));

            long size = clientData.readLong();
            byte[] buffer = new byte[1024];
            while (size > 0 && (bytesRead = clientData.read(buffer, 0, (int)
Math.min(buffer.length, size))) != -1) {
                output.write(buffer, 0, bytesRead);
                size -= bytesRead;
            }
            output.close();
            in.close();
            System.out.println("File "+fileName+" received from Server.");
        }else {
            int id = clientData.readInt();
            System.out.println("Sorry Client "+id+" File not found!");
        }
        catch (IOException ex) {
//Logger.getLogger(CLIENTConnection.class.getName()).log(Level.SEVERE, null, ex);}}}
}
```

9.5 Output

<u>Server</u>	<u>Client</u>
Server started	1.send file
Accepted connection:	2.receive file
Socket[addr=/127.0.0.1, port=38040, local port=4444]	Make selection : 1
File h.txt received from client	Enter file name : h.txt
Accepted connection:	File h.txt sent to server
Socket[addr=/127.0.0.1, port=38041, local port=4444]	1.send file
File b.txt sent to client	2.receive file
	Make selection : 1
	Enter file name : b.txt
	File b.txt sent from server

10. Familiarization of Wireshark packet capturing

10.1 Introduction

Wireshark is a free and open-source packet analyzer that captures data packets flowing over the network (wire) and presents them in an understandable form through its GUI. It is used for network troubleshooting, analysis, software and communications protocol development, and education. It runs on Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License. Wireshark supports a wide range of protocols like TCP, UDP, HTTP and even advanced protocols such as AppleTalk. It has several advance options such as filtering the packets, exporting packets, and name resolution. Wireshark can capture live data flowing through the network.

In computer networking, promiscuous mode or promisc mode is a mode for a wired network interface controller (NIC) or wireless network interface controller (WNIC) that causes the controller to pass all traffic it receives to the central processing unit (CPU) rather than passing only the frames that the controller is intended to receive. This mode is normally used for packet sniffing that takes place on a router or on a computer connected to a hub (instead of a switch) or one being part of a WLAN. Interfaces are placed into promiscuous mode by software bridges often used with hardware virtualization.

The Wireshark network sniffing make use of the promiscuous mode. First, Wireshark transfers the network interface into promiscuous mode where it can capture raw binary data flowing through the network. Then the chunks of binary data collected are then converted into a readable form. The packets are also re-assembled based on their sequence. Finally the captured and re-assembled data is analyzed. The initial analysis involves identifying the protocol type, the

communication channel, port numbers, and so on. At an advanced level, the different protocol headers can also be analyzed for a deeper understanding.

People use Wireshark for:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals
- Beside these examples, Wireshark can be helpful in many other situations too

10.2 Features :

- Available for UNIX and Windows.
- Capture live packet data from a network interface.
- Display packets with very detailed protocol information.
- Open and Save packet data captured.
- Import and Export packet data from and to a lot of other capture programs.
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.
- Create various statistics.
- Live capture from many different network media: Wireshark can capture traffic from many different network media types - and despite its name - including wireless LAN as well. Which media types are supported, depends on many things like the operating system you are using.
- Import files from many other capture programs: Wireshark can open packets captured from a large number of other capture programs.

- Export files for many other capture programs: Wireshark can save packets captured in a large number of formats of other capture programs.
- Open Source Software: Wireshark is an open source software project, and is released under the GNU. You can freely use Wireshark on any number of computers you like, without worrying about license keys or fees or such. In addition, all source code is freely available under the GPL. Because of that, it is very easy for people to add new protocols to Wireshark, either as plugins, or built into the source.

Getting Wireshark

Wireshark can be started on the PCs by executing the following steps:

Step 1 – Log on to the Linux PC

Step 2 - Open a the terminal window

Step 3 – Enter the command “sudo wireshark”.

Step 4 - Enter your account password

Running Wireshark

When you run the Wireshark program, the Wireshark graphical user interface shown in Figure 11.1 will be displayed. Initially, no data will be displayed in the various windows.

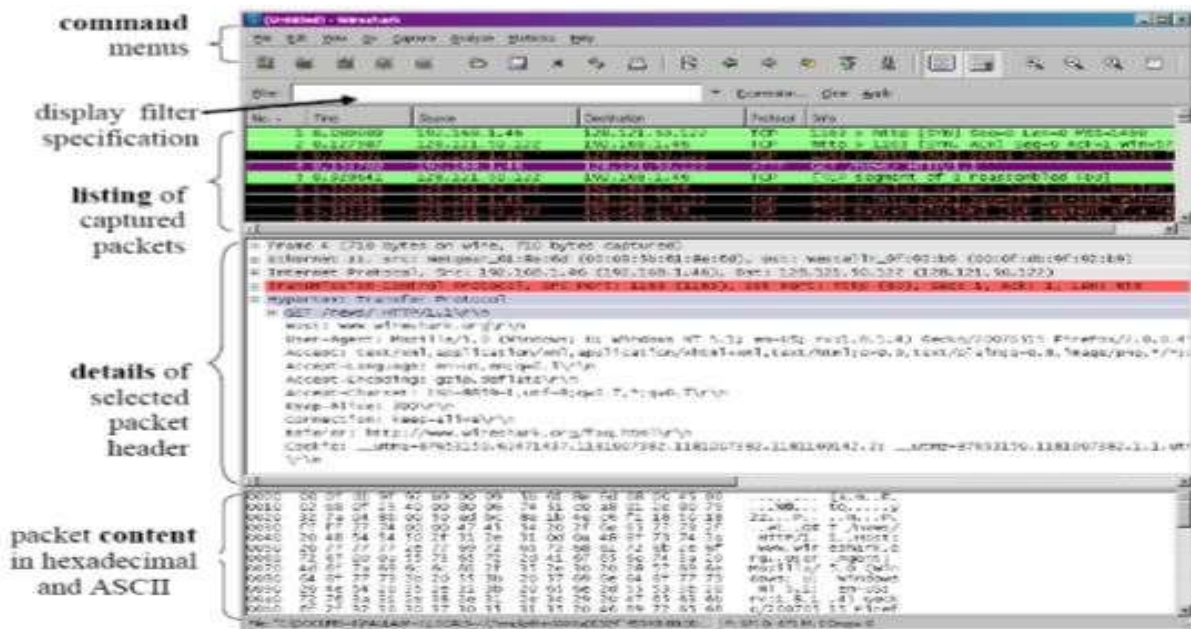


Figure 10.1. Wireshark Graphical User Interface

10.3 Wireshark interface components

- The **command menus** are standard pulldown menus located at the top of the window. Of interest to us now are the File and Capture menus. The File menu allows you to save captured packet data or open a file containing previously captured packet data, and exit the Wireshark application. The Capture menu allows you to begin packet capture.
- The **packet-listing window** displays a one-line summary for each packet captured, including the packet number (assigned by Wireshark; this is not a packet number contained in any protocol's header), the time at which the packet was captured, the packet's source and destination addresses, the protocol type, and protocol-specific information contained in the packet. The packet listing can be sorted according to any of these categories by clicking on a column name. The protocol type field lists the highest level protocol that sent or received this packet, i.e., the protocol that is the source or ultimate sink for this packet.

- The **packet-header details window** provides details about the packet selected (highlighted) in the packet listing window. (To select a packet in the packet listing window, place the cursor over the packet's one-line summary in the packet listing window and click with the left mouse button.). These details include information about the Ethernet frame and IP datagram that contains this packet. The amount of Ethernet and IP-layer detail displayed can be expanded or minimized by clicking on the right-pointing or down-pointing arrowhead to the left of the Ethernet frame or IP datagram line in the packet details window. If the packet has been carried over TCP or UDP, TCP or UDP details will also be displayed, which can similarly be expanded or minimized. Finally, details about the highest level protocol that sent or received this packet are also provided.
- The **packet-contents window** displays the entire contents of the captured frame, in both ASCII and hexadecimal format.
- Towards the top of the Wireshark graphical user interface, is the **packet display filter field**, into which a protocol name or other information can be entered in order to filter the information displayed in the packet-listing window (and hence the packet-header and packet-contents windows). In the example below, we'll use the packet-display filter field to have Wireshark hide (not display) packets except those that correspond to HTTP messages.

Test Run

To perform the following steps, identify the two PCs for your test run.

1. Start up your favorite web browser.
2. Start up the Wireshark software. You will initially see a window similar to that shown in Figure 11.2, except that no packet data will be displayed in the packet listing, packet-header, or packet-contents window, since Wireshark has not yet begun capturing packets. Make sure you check "Don't show this message again" and press "ok" on the small dialog box that pops up.
3. To begin packet capture, select the Capture pull down menu and select Interfaces. This will cause the "Wireshark: Capture Interfaces" window to be displayed, as shown in Figure 11.3.

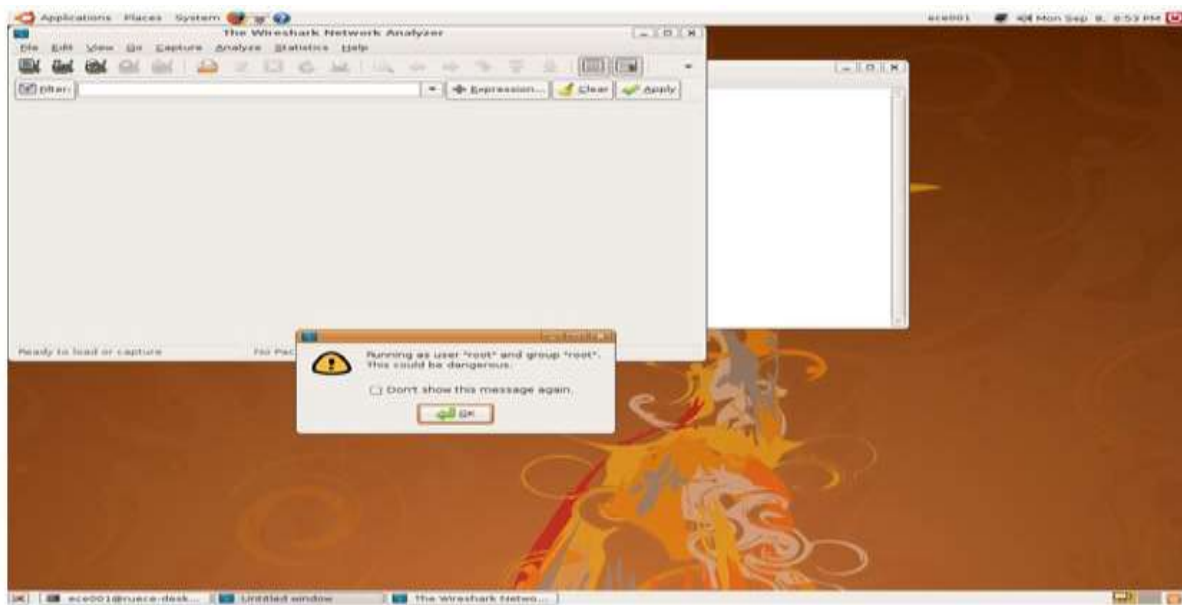


Figure. 10.2. Wireshark GUI

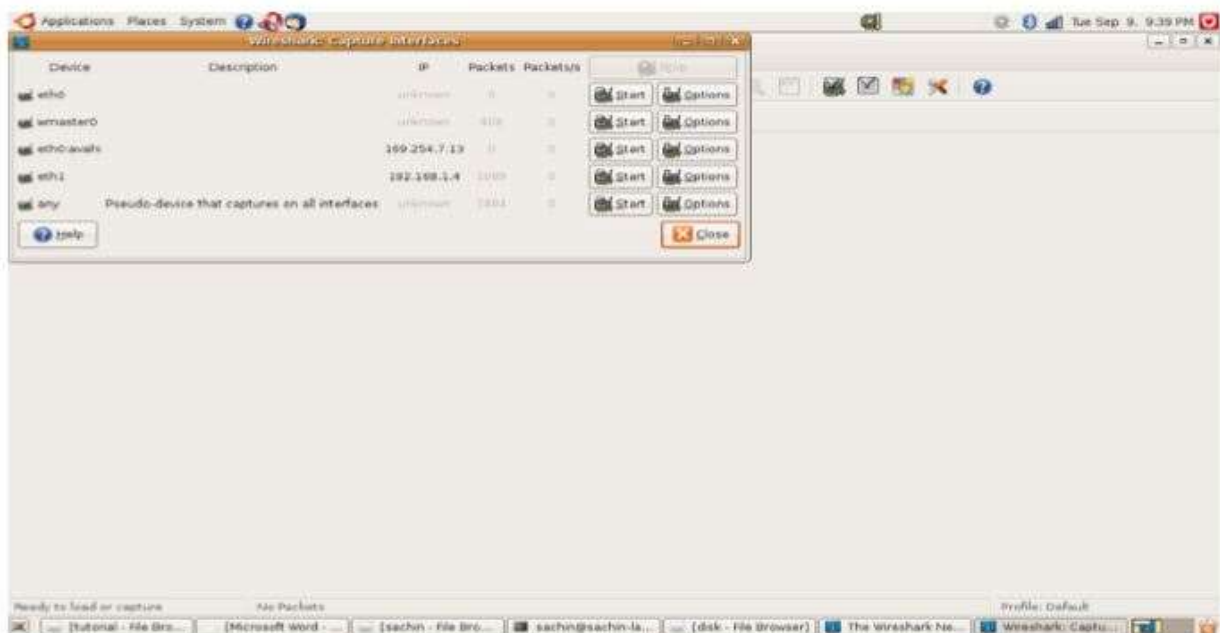


Figure 10.3. Wireshark Capture Interfaces Window

4. The network interfaces (i.e., the physical connections) that your computer has to the network are shown. The attached snapshot was taken from my computer. You may not see the exact same entries when you perform a capture. You will notice that eth0 and eth1 will be displayed. Click “Start” for interface eth0. Packet capture will now begin - all packets being sent / received from/by your computer are now being captured by Wireshark!

Example: If you started your Web browser on PC1, you can only connect to PC2 and PC9 (refer to the interconnections). If you want to connect to PC2, identify the IP address of eth0. The IP address is 10.0.1.3. If you wanted to connect to PC9, the IP address would be 10.0.1.17. While Wireshark is running, enter the URL to connect to the web server in PC2 and have that page displayed in your browser. In order to display this page, your browser will contact the HTTP server at 10.0.1.3(PC2) and exchange HTTP messages with the server in order to download this page. The Ethernet frames containing these HTTP messages will be captured by Wireshark.

5. After browser has displayed the intro.htm page, stop Wireshark packet capture by selecting stop in the Wireshark capture window. This will cause the Wireshark capture window to disappear and the main Wireshark window to display all packets captured since you began packet capture. The main Wireshark window should now look similar to Figure 11.1. You now have live packet data that contains all protocol messages exchanged between your computer and other network entities! The HTTP message exchanges with the PC2 web server should appear somewhere in the listing of packets captured. But there will be many other types of packets displayed as well (see, e.g., the many different protocol types shown in the Protocol column in Figure 11.1). Even though the only action you took was to download a web page, there were evidently many other protocols running on your computer that are unseen by the user.
6. Type in “http” (without the quotes, and in lower case – all protocol names are in lower case in Wireshark) into the display filter specification window at the top of the main Wireshark window. Then select Apply (to the right of where you entered “http”). This will cause only HTTP message to be displayed in the packet-listing window.

7. Select the first http message shown in the packet-listing window. This should be the HTTP GET message that was sent from your computer(ex. PC1) to the PC2 HTTP server. When you select the HTTP GET message, the Ethernet frame, IP datagram, TCP segment, and HTTP message header information will be displayed in the packet-header window². By clicking on right pointing and down-pointing arrows heads to the left side of the packet details window, minimize the amount of Frame, Ethernet, Internet Protocol, and Transmission Control Protocol information displayed. Maximize the amount information displayed about the HTTP protocol. Your Wireshark display should now look roughly as shown in Figure 11.4 (Note, in particular, the minimized amount of protocol information for all protocols except HTTP, and the maximized amount of protocol information for HTTP in the packet-header window).

8. Exit Wireshark

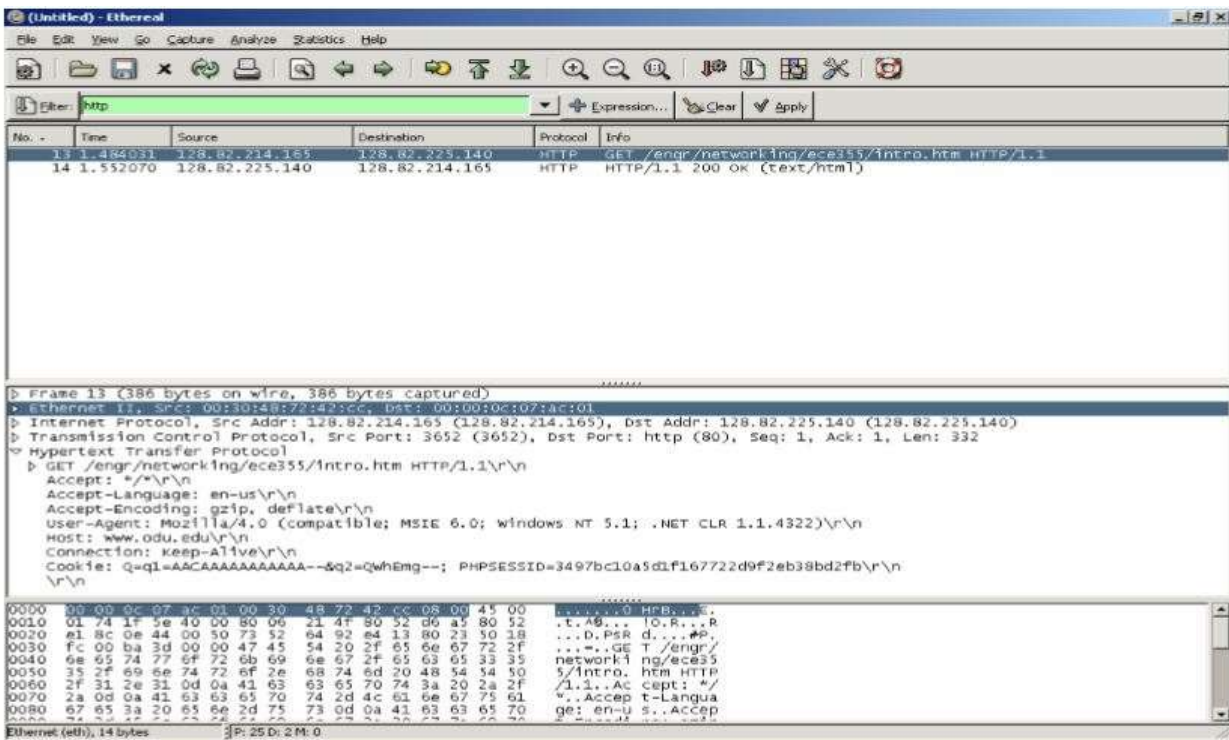


Figure 10.4. Wireshark display after step 8

11. Install network simulator NS-2 in any of the Linux operating system and simulate wired

11.1 Introduction

Network Simulator Version 2, widely known as NS2, is an event driven simulation tool that is useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2. In general, NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviors. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field. Among these are the University of California and Cornell University who developed the REAL network simulator,¹ the foundation which NS is based on. Since 1995 the Defense Advanced Research Projects Agency (DARPA) supported development of NS through the Virtual InterNetwork Testbed (VINT) project. Currently the National Science Foundation (NSF) has joined the ride in development. Last but not the least, the group of researchers and developers in the community are constantly working to keep NS2 strong and versatile.

11.2 Steps to install NS2 on Linux

NS or Network Simulator is a well known open source tool for simulating wired or wireless computer networks. Nam or Network animator is an animator tool for graphical representation of network traces and real world packet traces. NS and nam can be used together to create a simulated network and analyze it manually or graphically.

1. Install NS2

Run the following commands in a terminal:

```
sudo apt-get install ns2
```

2. Install Nam

```
sudo apt-get purge nam
```

```
wget --user-agent="Mozilla/5.0 (Windows NT 5.2; rv:2.0.1)Gecko/20100101  
Firefox/4.0.1" http://tecnobyte.com/wp-content/uploads/2015/11/nam\_1.14\_amd64.zip
```

```
unzip nam_1.14_amd64.zip
```

```
sudo dpkg -i
```

```
nam_1.14_amd64.deb
```

```
sudo apt-mark hold nam
```

This will install nam and ns2 on your Linux. Now let's check if everything is functional by executing a small TCL/Tk simulation script.

```
#Create a simulator object
set ns [new Simulator]
#Define different colors for data flows
$ns color 1 Blue
$ns color 2 Red

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

#Define a 'finish' procedure
proc finish { } {
    global ns nf
    $ns flush-trace
    #Close the trace file
    close $nf
}

#Execute nam on the trace file
exec nam out.nam &
exit 0
}

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

#Create links between the nodes
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n3 $n2 1Mb 10ms SFQ

$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right
```



```
#Monitor the queue for the link between node 2 and node 3
$ns duplex-link-op $n2 $n3 queuePos 0.5
```

```
#Create a UDP agent and attach it to node n0
set udp0 [new Agent/UDP]
$udp0 set class_ 1
$ns attach-agent $n0 $udp0
```

```
# Create a CBR traffic source and attach it to udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.005
$cbr0 attach-agent $udp0
```

```
#Create a UDP agent and attach it to node n1
set udp1 [new Agent/UDP]
$udp1 set class_ 2
$ns attach-agent $n1 $udp1
```

```
# Create a CBR traffic source and attach it to udp1
set cbr1 [new Application/Traffic/CBR]
$cbr1 set packetSize_ 500
$cbr1 set interval_ 0.005
$cbr1 attach-agent $udp1
```

```
#Create a Null agent (a traffic sink) and attach it to node n3
set null0 [new Agent/Null]
$ns attach-agent $n3 $null0
```

```
#Connect the traffic sources with the traffic sink
$ns connect $udp0 $null0 &nbsp;
$ns connect $udp1 $null0
```

```
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 1.0 "$cbr1 start"
$ns at 4.0 "$cbr1 stop"
$ns at 4.5 "$cbr0 stop"
```

```
#Call the finish procedure after 5 seconds of simulation time
```

```
$ns at 5.0 "finish"
```

```
#Run the simulation
```

```
$ns run
```

Save it into example.tcl and run the following command:

```
ns example.tcl
```

if everything goes fine, which shows the animator network on nam as shown in figure 12.1

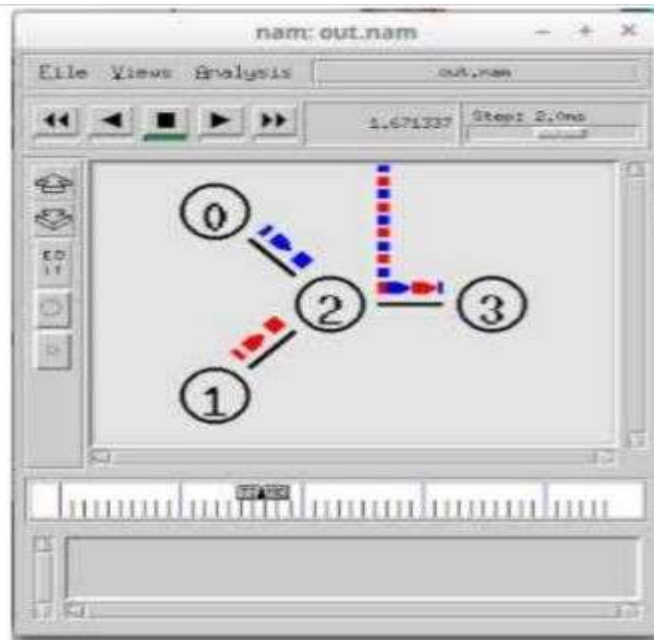


Figure 11.1. Network Animator- Simulated Network