



SAHYADRI

**COLLEGE OF ENGINEERING & MANAGEMENT
MANGALURU**

(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)

**DEPARTMENT OF INFORMATION
SCIENCE & ENGINEERING**

COMPUTER NETWORKS

LABORATORY MANUAL

15CSL57

V Semester - BE

**Prepared by
Mrs. Suchetha G / Mr. Naitik
Asst. Professors**

**Department of Information Science & Engineering
SAHYADRI College of Engineering & management Adyar, Mangaluru
575 007**

DO'S AND DON'TS IN LABORATORY

1. Remove footwear and keep it in the rack before entering the lab.
2. Make entry in the Log Book as soon as you enter the Laboratory.
3. All the students should sit according to their roll numbers starting from their left to right.
4. All the students are supposed to enter the terminal number in the log book.
5. Do not change the terminal on which you are working.
6. All the students are expected to get at least the algorithm of the program/concept to be implemented.
7. Strictly observe the instructions given by the teacher/Lab Instructor.
8. The record should have following details:
 - a.Date
 - b.Aim
 - c.Algorithm
 - d.Program
 - e.Output

CONTENTS

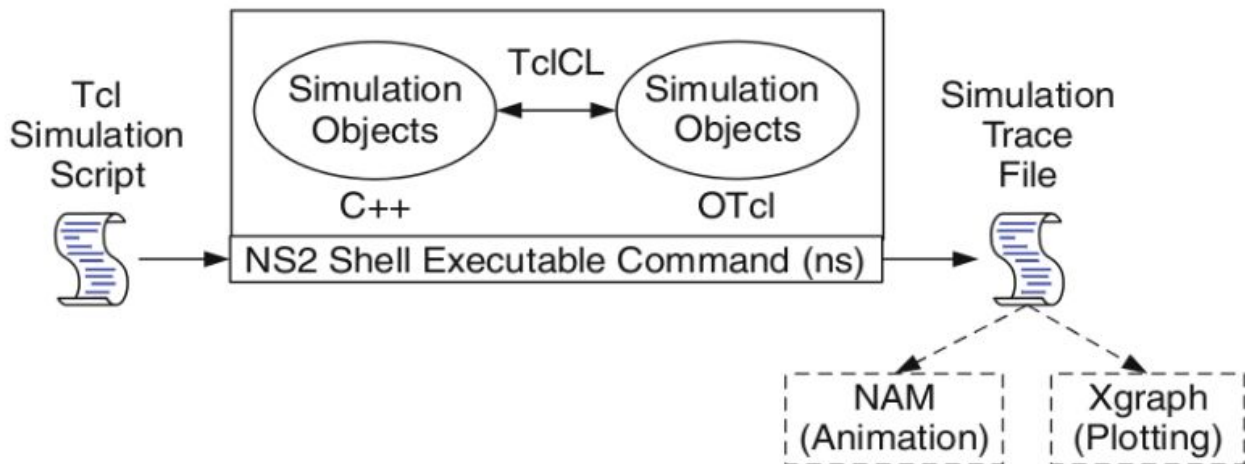
PART A	
1	Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.
2	Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.
3	Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.
4	Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.
5	Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.
6	Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.
PART B	
Implement the following in Java:	
7	Write a program for error detecting code using CRC-CCITT (16- bits).
8	Write a program to find the shortest path between vertices using bellman-ford algorithm.
9	Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.
10	Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.
11	Write a program for simple RSA algorithm to encrypt and decrypt the data.
12	Write a program for congestion control using leaky bucket algorithm.

Introduction to NS-2

- Widely known as NS2, is simply an event driven simulation tool.

- 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.

Basic Architecture of NS2



Tcl scripting

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

Basics of TCL

Syntax: command arg1 arg2 arg3

o Hello World!

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

o Variables

```
set a 5
set b $a
```

Command Substitution

```
set len [string length foobar]
set len [expr [string length foobar] + 9]
```

o Simple Arithmetic

```
expr 7.2 / 4
```

o Procedures

```
proc Diag {a b} {
```

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

o Loops

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

Wired TCL Script Components

Create the event scheduler

Open new files & turn on the tracing

Create the nodes

Setup the links

Configure the traffic type (e.g., TCP, UDP, etc)

Set the time of traffic generation (e.g., CBR, FTP)

Terminate the simulation

NS Simulator Preliminaries.

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

Initialization and Termination of TCL Script in NS-2

An ns simulation starts with the command

set ns [new Simulator]

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

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

#Open the Trace file

```
set tracefile1 [open out.tr w]
$ns trace-all $tracefile1
```

#Open the NAM trace file

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

\$ns namtrace-all \$namfile

The above creates a dta trace file called "out.tr" and a nam visualization trace file called "out.nam". Within the tcl script, these files are not called explicitly by their names, but instead by pointers that are declared above and called "tracefile1" and "namfile" respectively. Remark that they begins with a # symbol. The second line open the file "out.tr" to be used for writing, declared with the letter "w". The third line uses a simulator method called trace-all that have as parameter the name of the file where the traces will go. The last line tells the simulator to record all simulation traces in NAM input format. It also gives the file name that the trace will be written to later by the command \$ns flush-trace. In our case, this will be the file pointed at by the pointer "\$namfile", i.e the file "out.tr". The termination of the program is done using a "finish" procedure.

#Define a "finish" procedure

```
Proc finish { } {  
  global ns tracefile1 namfile  
  $ns flush-trace  
  Close $tracefile1  
  Close $namfile  
  Exec nam out.nam &  
  Exit 0  
}
```

The word proc declares a procedure in this case called **finish** and without arguments. The word **global** is used to tell that we are using variables declared outside the procedure.

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

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

\$ns at 125.0 "finish"

Above script will be used to call "**finish**" at time 125sec. Indeed, the **at** method of the simulator allows us to schedule events explicitly.

The simulation can then begin using the command

\$ns run

Definition of a network of links and nodes

The way to define a node is

set n0 [\$ns node]

The node is created which is printed by the variable n0. When we shall refer to that node in the script we shall thus write \$n0.

Once we define several nodes, we can define the links that connect them. An example of a definition of a link is:

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

Which means that \$n0 and \$n2 are connected using a bi-directional link that has 10ms of propagation delay and a capacity of 10Mb per sec for each direction.

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

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

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

Agents and Applications

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

FTP over TCP

TCP is a dynamic reliable congestion control protocol. It uses Acknowledgements created by the destination to know whether packets are well received.

There are number variants of the TCP protocol, such as Tahoe, Reno, NewReno, Vegas. The type of agent appears in the first line:

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

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

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

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

#Setup a UDP connection

```
set udp [new Agent/UDP]
$ns attach-agent $n1
$udp set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_2
```

#Setup a CBR over UDP connection

The below shows the definition of a CBR application using a UDP agent The command **\$ns attach-agent \$n4 \$sink** defines the destination node.

The command **\$ns connect \$tcp \$sink** finally makes the TCP connection between the source and destination nodes.

```
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set packetize_ 100
$cbr set rate_ 0.01Mb
$cbr set random_ false
```

TCP has many parameters with initial fixed defaults values that can be changed if mentioned explicitly. For example, the default TCP packet size has a size of 1000bytes. This can be changed to another value, say 552bytes, using the command **\$tcp set packetSize_ 552**.

When we have several flows, we may wish to distinguish them so that we can identify them with different colors in the visualization part. This is done by the command **\$tcp set fid_ 1** that assigns to the TCP connection a flow identification of "1". We shall later give the flow identification of "2" to the UDP connection.

CBR over UDP

A UDP source and destination is defined in a similar way as in the case of TCP.

Instead of defining the rate in the command **\$cbr set rate_ 0.01Mb**, one can define the time interval between transmission of packets using the command.

\$cbr set interval_ 0.005

The packet size can be set to some value using

\$cbr set packetSize_ <packet size>

Scheduling Events

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

\$ns at <time><event>

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

\$ns at 0.1 "\$cbr start"

\$ns at 1.0 "\$ftp start"

\$ns at 124.0 "\$ftp stop"

\$ns at 124.5 "\$cbr stop"

Structure of Trace Files

When tracing into an output ASCII file, the trace is organized in 12 fields as follows in fig shown below, The meaning of the fields are:

Event	Time	From Node	To Node	PKT Type	PKT Size	Flags	Fid	Src Addr	Dest Addr	Seq Num	Pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	-----------	---------	--------

1. The first field is the event type. It is given by one of four possible symbols r, +, -, d which correspond respectively to receive (at the output of the link), enqueued, dequeued and dropped.
2. The second field gives the time at which the event occurs.
3. Gives the input node of the link at which the event occurs.
4. Gives the output node of the link at which the event occurs.
5. Gives the packet type (eg CBR or TCP)
6. Gives the packet size
7. Some flags
8. This is the flow id (fid) of IPv6 that a user can set for each flow at the input OTcl script one can further use this field for analysis purposes; it is also used when specifying stream color for the NAM display.
9. This is the source address given in the form of —node.port || .
10. This is the destination address, given in the same form.
11. This is the network layer protocol's packet sequence number. Even though UDP implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes
12. The last field shows the unique id of the packet.

XGRAPH

The xgraph program draws a graph on an x-display given data read from either data file or from standard input if no files are specified. It can display upto 64 independent data sets using different colors and line styles for each set. It annotates the graph with a title, axis labels, grid lines or tick marks, grid labels and a legend.

Syntax:

Xgraph [options] file-name

Options are listed here

/-bd <color> (Border)

This specifies the border color of the xgraph window.

/-bg <color> (Background)

This specifies the background color of the xgraph window.

/-fg<color> (Foreground)

This specifies the foreground color of the xgraph window.

/-lf <fontname> (LabelFont)

All axis labels and grid labels are drawn using this font.

/-t<string> (Title Text)

This string is centered at the top of the graph.

/-x <unit name> (XunitText)

This is the unit name for the x-axis. Its default is "X".

/-y <unit name> (YunitText)

This is the unit name for the y-axis. Its default is "Y".

Awk- An Advanced

Awk is a programmable, pattern-matching, and processing tool available in UNIX. It works equally well with text and numbers.

Awk is not just a command, but a programming language too. In other words, awk utility is a pattern scanning and processing language. It searches one or more files to see if they contain lines that match specified patterns and then perform associated actions, such as writing the line to the standard output or incrementing a counter each time it finds a match.

Syntax:

awk option 'selection_criteria {action}' file(s)

Here, selection_criteria filters input and select lines for the action component to act upon. The selection_criteria is enclosed within single quotes and the action within the curly braces. Both the selection_criteria and action forms an awk program.

Example: \$ awk „/manager/ {print}” emp.lst

Variables

Awk allows the user to use variables of their choice. You can now print a serial number, using the variable kount, and apply it to those directors drawing a salary exceeding 6700:

```
$ awk -F"|" „$3 == "director" && $6 > 6700 {  
kount =kount+1  
printf " %3f %20s %-12s %d\n", kount,$2,$3,$6 }" emp.lst
```

THE -f OPTION: STORING awk PROGRAMS IN A FILE

You should hold large awk programs in separate files and provide them with the awk extension for easier identification. Let's first store the previous program in the file empawk.awk:

\$ cat empawk.awk

Observe that this time we haven't used quotes to enclose the awk program. You can now use awk with the -f filename option to obtain the same output:

Awk -F"|" -f empawk.awk emp.lst

THE BEGIN AND END SECTIONS

Awk statements are usually applied to all lines selected by the address, and if there are no addresses, then they are applied to every line of input. But, if you have to print something before processing the first line, for example, a heading, then the BEGIN section can be used gainfully. Similarly, the end section is useful in printing some totals after processing is over. The BEGIN and END sections are optional and take the form

BEGIN {action}

END {action}

These two sections, when present, are delimited by the body of the awk program. You can use them to print a suitable heading at the beginning and the average salary at the end.

BUILT-IN VARIABLES

Awk has several built-in variables. They are all assigned automatically, though it is also possible for a user to reassign some of them. You have already used NR, which signifies the record number of the current line. We'll now have a brief look at some of the other variables.

The FS Variable: as stated elsewhere, awk uses a contiguous string of spaces as the default field delimiter. FS redefines this field separator, which in the sample database happens to be the |. When

used at all, it must occur in the BEGIN section so that the body of the program knows its value before it starts processing:

BEGIN {FS="|"}

This is an alternative to the -F option which does the same thing.

The OFS Variable: when you used the print statement with comma-separated arguments, each argument was separated from the other by a space. This is awk's default output field separator, and can be reassigned using the variable OFS in the BEGIN section: **BEGIN { OFS="~" }**

When you reassign this variable with a ~ (tilde), awk will use this character for delimiting the print arguments. This is a useful variable for creating lines with delimited fields.

The NF variable: NF comes in quite handy for cleaning up a database of lines that don't contain the right number of fields. By using it on a file, say emp.lst, you can locate those lines not having 6 fields, and which have crept in due to faulty data entry:

\$awk „BEGIN {FS = "|"}

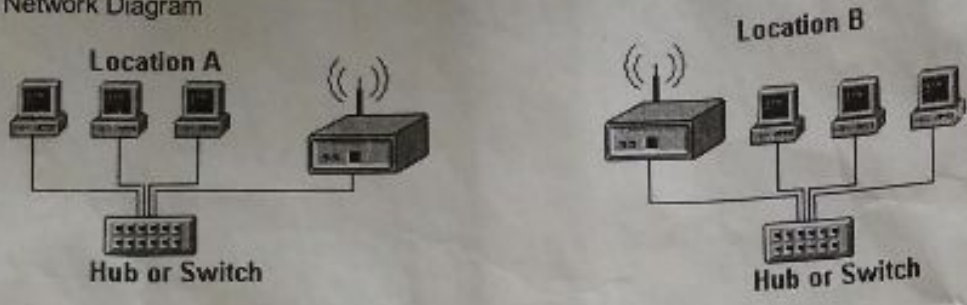
NF! =6 {

Print "Record No ", NR, "has", "fields"}" empx.lst

1. Implement three nodes point – to – point network with duplex links between them. Set the queue size, vary the bandwidth and find the number of packets dropped.

Theory:

Point to Point Network Diagram



Description:

Point to point networks are used to connect one location to one other location. The above diagram shows two connected multi-user networks (using a Hub or Switch). Either location (A or B) may be configured as a Direct connection (without a Hub or Switch).

Air-Frame 100 can be used to create many, co-located point to point networks.

Typical Applications:

Internet, Intranet or Extranet configurations. ISP access networks (bridged or routed). LAN to LAN applications (bridged or routed see below). Remote data capture (Telemetry or SCADA). Remote Control. Remote Monitoring. Security.

- A **duplex** communication system is a point-to-point system composed of two connected parties or devices that can communicate with one another in both directions. There are two types of duplex communication systems: full-duplex and half-duplex.
- In a **full duplex** system, both parties can communicate to the other simultaneously. An example of a full-duplex device is a telephone; the parties at both ends of a call can speak and be heard by the other party simultaneously. The earphone reproduces the speech of the remote party as the microphone transmits the speech of the local party, because there is a two-way communication channel between them.

Program:**1.tcl**

```
# This script is created by NSG2 beta1
# http://wushoupong.googlepages.com/nsg

#=====
# Simulation parameters setup
#=====
set val(stop) 10.0 ;# time of simulation end

#=====
# Initialization
#=====
#Create a ns simulator
```

```
set ns [new Simulator]

#Open the NS trace file
set tracefile [open 1.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open 1.nam w]
$ns namtrace-all $namfile

#=====
# Nodes Definition
#=====
#Create 3 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

#=====
# Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n0 $n1 1000.0Kb 10ms DropTail
$ns queue-limit $n0 $n1 5
$ns duplex-link $n1 $n2 1000.0Kb 10ms DropTail
$ns queue-limit $n1 $n2 3

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

#=====
# Agents Definition
#=====
#Setup a TCP/Vegas connection
set tcp0 [new Agent/TCP/Vegas]
$ns attach-agent $n0 $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n2 $sink1
$ns connect $tcp0 $sink1
$tcp0 set packetSize_ 1500

#=====
# Applications Definition
#=====
#Setup a FTP Application over TCP/Vegas connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ns at 1.0 "$ftp0 start"
```

```
$ns at 2.0 "$ftp0 stop"  
$ns at 10.0 "finish"
```

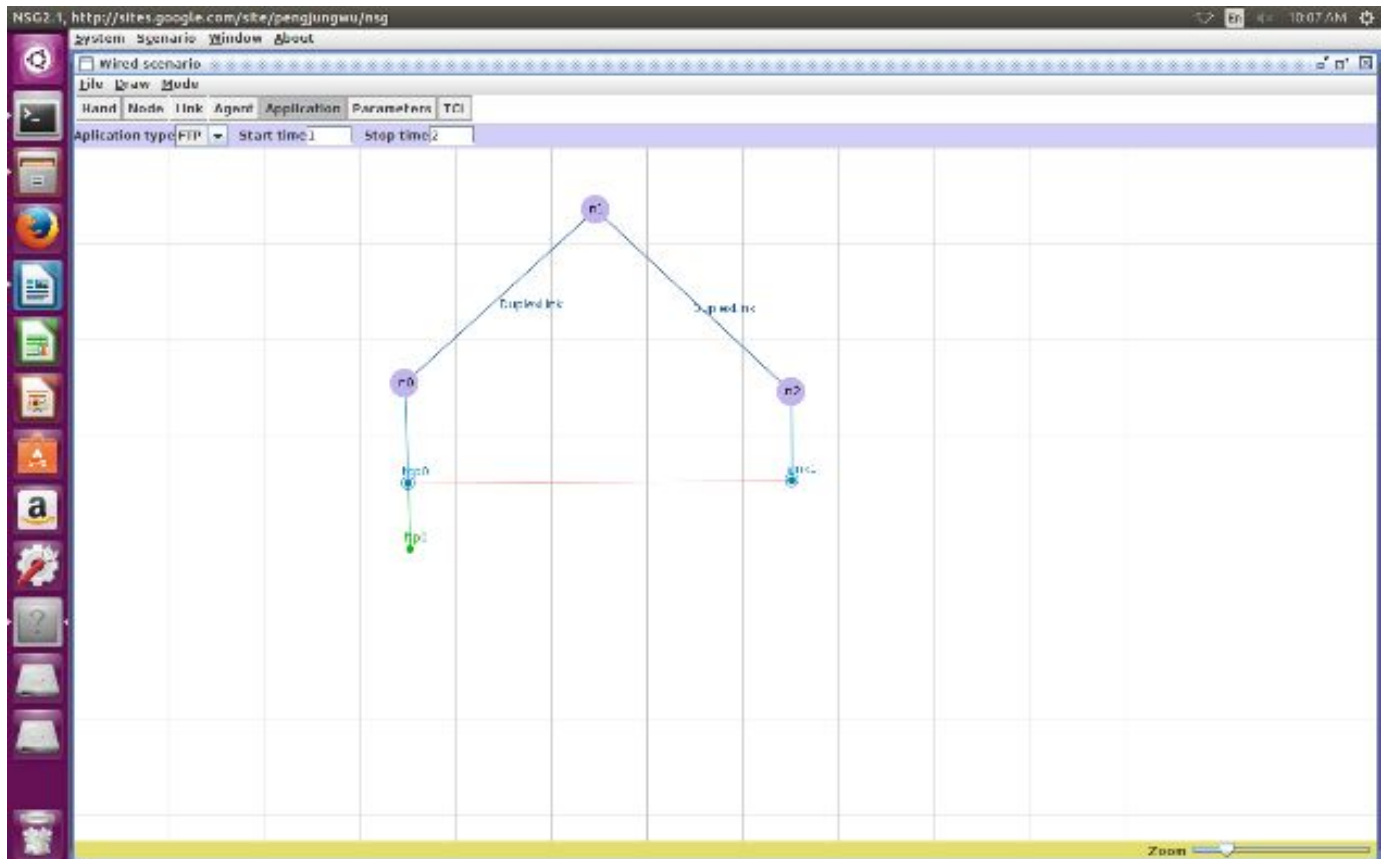
```
#=====
# Termination
#=====
#Define a 'finish' procedure
proc finish {} {
  global ns tracefile namfile
  $ns flush-trace
  close $tracefile
  close $namfile
  exec nam 5.nam &
  exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

1.awk

```
BEGIN{
  count=0;
  total=0;
}
{
  event = $1;
  if(event == "d")
  {
    count++;
  }
}
END{
  printf("No of packets dropped: %d\n",count);
}
```

Execution and Output:

```
[student@student-Veriton-Series:~/ns2$ java -jar nsg2.1.jar
```

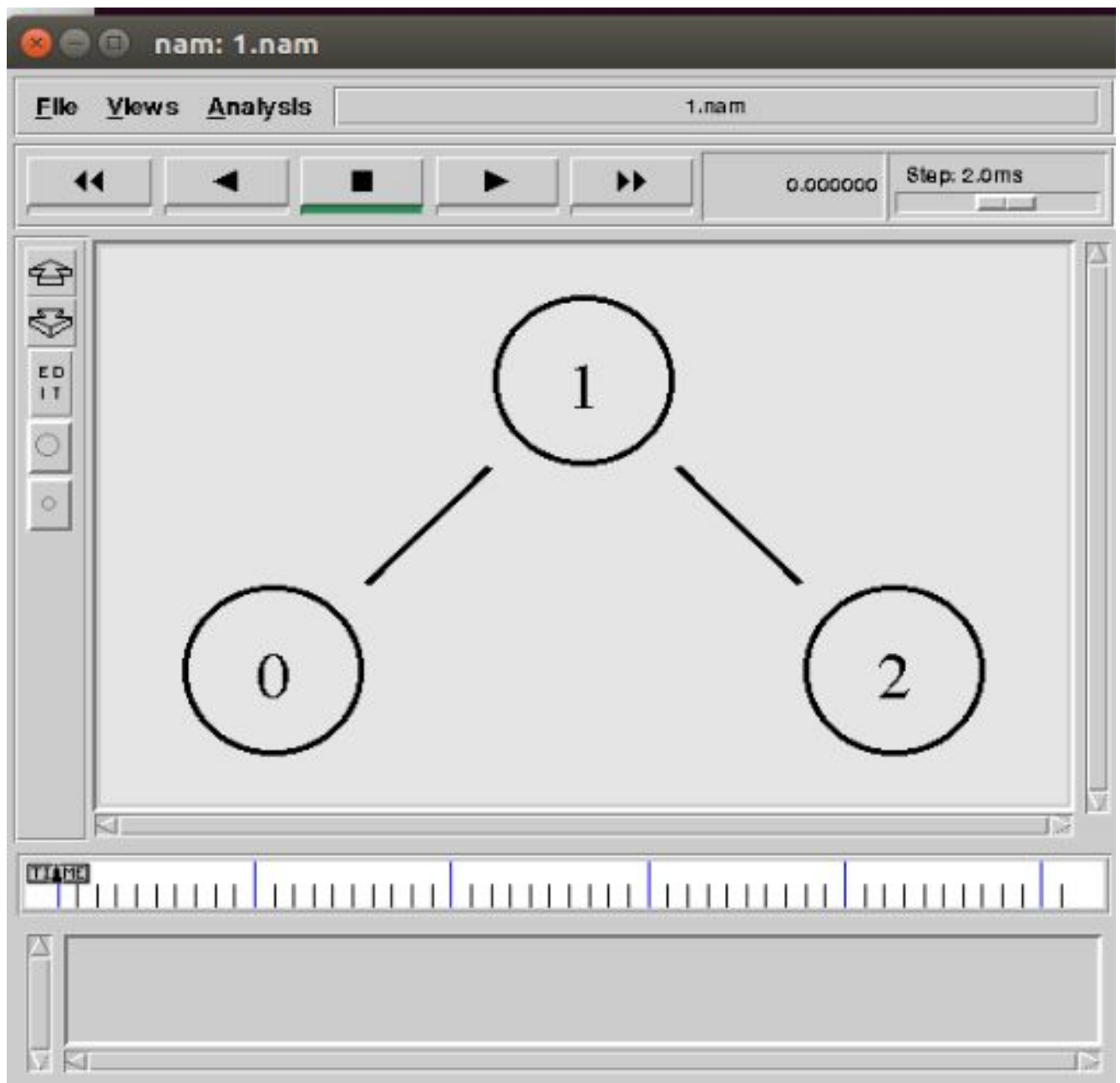


```
[student@student-Veriton-Series:~/ns2$ ns 1.tcl
[student@student-Veriton-Series:~/ns2$ gedit 1.awk
[student@student-Veriton-Series:~/ns2$ awk -f 1.awk 1.tr
```

No of packets dropped: 3

Sl.No.	Data rate		Propagation Delay		Queue Limit		Packets Dropped
	n0-n1	n1-n2	n0-n1	n1-n2	n0-n1	n1-n2	
1	1000.0Kb	1000.0Kb	10ms	10ms	5	3	3
2							
3							
4							

Nam file Window:



2. Implement transmission of ping messages/trace route over a network topology consisting of 6 nodes and find the number of packets dropped due to congestion.

Theory:

Telnet is a network protocol used on the Internet or local area networks to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP). Telnet was developed in 1968 beginning with RFC 15, extended in RFC 854, and standardized as Internet Engineering Task Force (IETF) Internet Standard STD 8, one of the first Internet standards.

The File Transfer Protocol (FTP) is a standard network protocol used to transfer computer files from one host to another host over a TCP-based network, such as the Internet. FTP is built on a client-server architecture and uses separate control and data connections between the client and the server.[1] FTP users may authenticate themselves using a clear-text sign-in protocol, normally in the form of a username and password, but can connect anonymously if the server is configured to allow it. For secure transmission that protects the username and password, and encrypts the content, FTP is often secured with SSL/TLS (FTPS). SSH File Transfer Protocol (SFTP) is sometimes also used instead, but is technologically different.

Program:

2.tcl

```
# This script is created by NSG2 beta1
# <http://wushoupong.googlepages.com/nsg>
#=====
# Simulation parameters setup
#=====
set val(stop) 10.0 ;# time of simulation end

#=====
#Initialization
#=====
#Create a ns simulator
set ns [new Simulator]
```

```
#Open the NS trace file
set tracefile [open 2.tr w]
$ns trace-all $tracefile
```

```
#Open the NAM trace file
set namfile [open 2.nam w]
$ns namtrace-all $namfile
$ns color 1 Blue
$ns color 2 Red
```

```
#=====
# Nodes Definition
#=====
#Create 7 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
```

```
#=====
# Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n0 $n1 1Mb 50ms DropTail
$ns queue-limit $n0 $n1 50
$ns duplex-link $n0 $n3 1Mb 50ms DropTail
$ns queue-limit $n0 $n3 50
$ns duplex-link $n0 $n4 1Mb 50ms DropTail
$ns queue-limit $n0 $n4 50
$ns duplex-link $n0 $n5 1Mb 50ms DropTail
$ns queue-limit $n0 $n5 2
$ns duplex-link $n0 $n2 1Mb 50ms DropTail
$ns queue-limit $n0 $n2 2
$ns duplex-link $n0 $n6 1Mb 50ms DropTail
$ns queue-limit $n0 $n6 1
```

```
#Give node position (for NAM)
$ns duplex-link-op $n0 $n1 orient left-up
$ns duplex-link-op $n0 $n4 orient right-down
$ns duplex-link-op $n0 $n2 orient right-up
$ns duplex-link-op $n0 $n3 orient right
$ns duplex-link-op $n0 $n5 orient left-down
$ns duplex-link-op $n0 $n6 orient left
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received ping answer $from with round-trip-time $rtt ms."
```

```
}
set p1 [new Agent/Ping]
set p2 [new Agent/Ping]
set p3 [new Agent/Ping]
set p4 [new Agent/Ping]
set p5 [new Agent/Ping]
set p6 [new Agent/Ping]
$ns attach-agent $n1 $p1
$ns attach-agent $n2 $p2
$ns attach-agent $n3 $p3
$ns attach-agent $n4 $p4
$ns attach-agent $n5 $p5
$ns attach-agent $n6 $p6
$ns connect $p1 $p4
$ns connect $p2 $p5
$ns connect $p3 $p6
$ns at 0.2 "$p1 send"
$ns at 0.4 "$p2 send"
$ns at 0.6 "$p3 send"
$ns at 1.0 "$p4 send"
$ns at 1.2 "$p5 send"
$ns at 1.4 "$p6 send"
$ns at 10.0 "finish"

#=====
# Agents Definition
#=====

#=====
# Applications Definition
#=====

#=====
# Termination
#=====
#Define a 'finish' procedure
proc finish {} {
global ns tracefile namfile
$ns flush-trace
close $tracefile
close $namfile
exec nam 2.nam &
exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

2.awk:

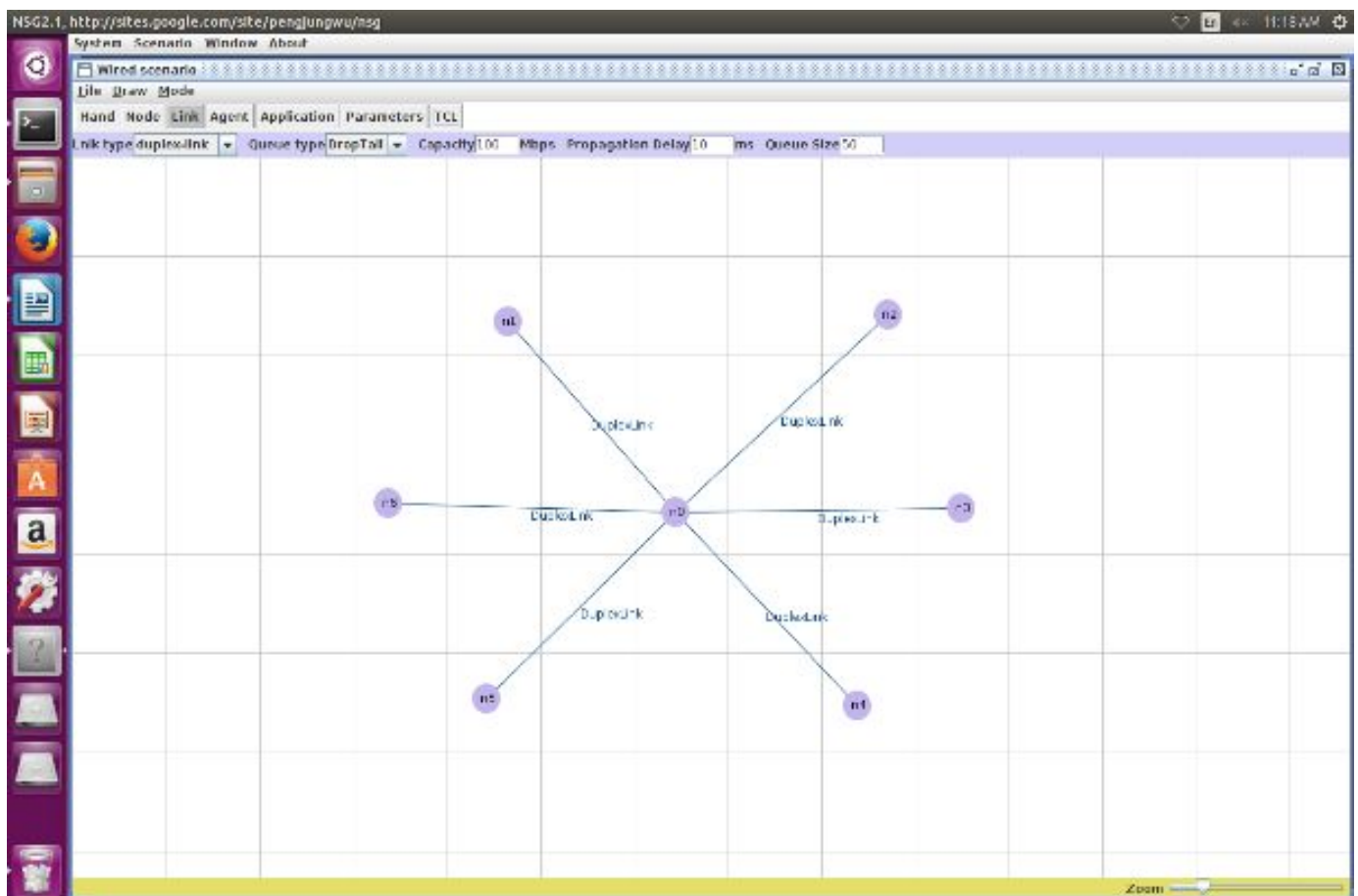
```

BEGIN{
count=0;
}
{
event=$1;
if(event=="d")
{
count++;
}
}
END{
printf("No.of packets dropped:%d\n",count);
}

```

Execution and Output:

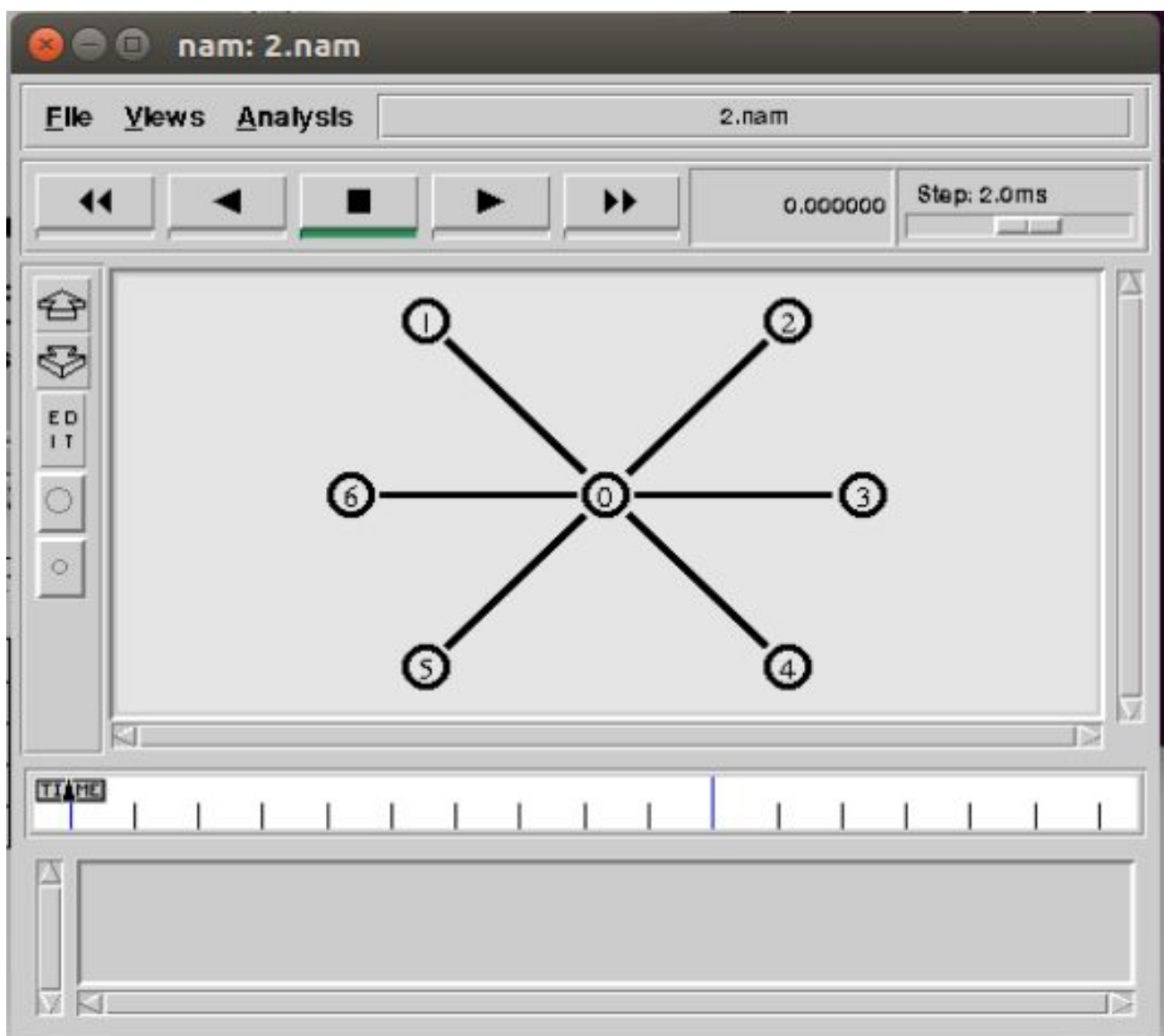
[student@student-Veriton-Series:~/ns2\$ java -jar nsg2.1.jar



```
[student@student-Veriton-Series:~/ns2$ ns 2.tcl  
node 1 received ping answer 4 with round-trip-time 202.0 ms.  
node 2 received ping answer 5 with round-trip-time 202.0 ms.  
node 4 received ping answer 1 with round-trip-time 202.0 ms.  
node 5 received ping answer 2 with round-trip-time 202.0 ms.
```

```
[student@student-Veriton-Series:~/ns2$ gedit 2.awk  
[student@student-Veriton-Series:~/ns2$ awk -f 2.awk 2.tr  
No of packets dropped: 2
```

Nam file window:



3. Implement an Ethernet LAN using n nodes and set multiple traffic nodes and plot congestion window for different source / destination.

3.tcl:

```
# This script is created by NSG2 beta1
# <http://wushoupong.googlepages.com/nsg>
#=====
# Simulation parameters setup
#=====
set val(stop) 10.0 ;# time of simulation end

#=====
# Initialization
#=====
#Create a ns simulator
set ns [new Simulator]

#Open the NS trace file
set tracefile [open 3.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open 3.nam w]
$ns namtrace-all $namfile
$ns color 1 Blue
$ns color 2 Red

#=====
# Nodes Definition
#=====
#Create 9 nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set n7 [$ns node]
set n8 [$ns node]
$ns make-lan "$n3 $n4 $n5 $n6 $n7 $n8" 512Kb 50ms LL Queue/DropTail

#=====
# Links Definition
#=====
#Createlinks between nodes
$ns duplex-link $n1 $n0 2.0Mb 50ms DropTail
```

```
$ns queue-limit $n1 $n0 50
$ns duplex-link $n2 $n0 2.0Mb 50ms DropTail
$ns queue-limit $n2 $n0 50
$ns duplex-link $n0 $n3 1.0Mb 50ms DropTail
$ns queue-limit $n0 $n3 7
```

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

```
#=====
# Agents Definition
#=====
```

```
#Setup a TCP/Reno connection
set tcp0 [new Agent/TCP/Reno]
$ns attach-agent $n1 $tcp0
set sink3 [new Agent/TCPSink]
$ns attach-agent $n7 $sink3
$ns connect $tcp0 $sink3
$tcp0 set packetSize_ 1500
$tcp0 set class_ 1
set tfile1 [open cwnd1.tr w]
$tcp0 attach $tfile1
$tcp0 trace cwnd_
```

```
#Setup a TCP/Vegas connection
set tcp1 [new Agent/TCP/Vegas]
$ns attach-agent $n2 $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n4 $sink2
$ns connect $tcp1 $sink2
$tcp1 set packetSize_ 1500
$tcp1 set class_ 2
set tfile2 [open cwnd2.tr w]
$tcp1 attach $tfile2
$tcp1 trace cwnd_
```

```
#=====
# Applications Definition
#=====
```

```
#Setup a FTP Application over TCP/Vegas connection
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp1
$ns at 0.3 "$ftp0 start"
$ns at 8.0 "$ftp0 stop"
```

```
#Setup a FTP Application over TCP/Reno connection
set ftp1 [new Application/FTP]
```



```
$ftp1 attach-agent $tcp0
$ns at 0.3 "$ftp1 start"
$ns at 8.0 "$ftp1 stop"
$ns at 10.0 "finish"
```

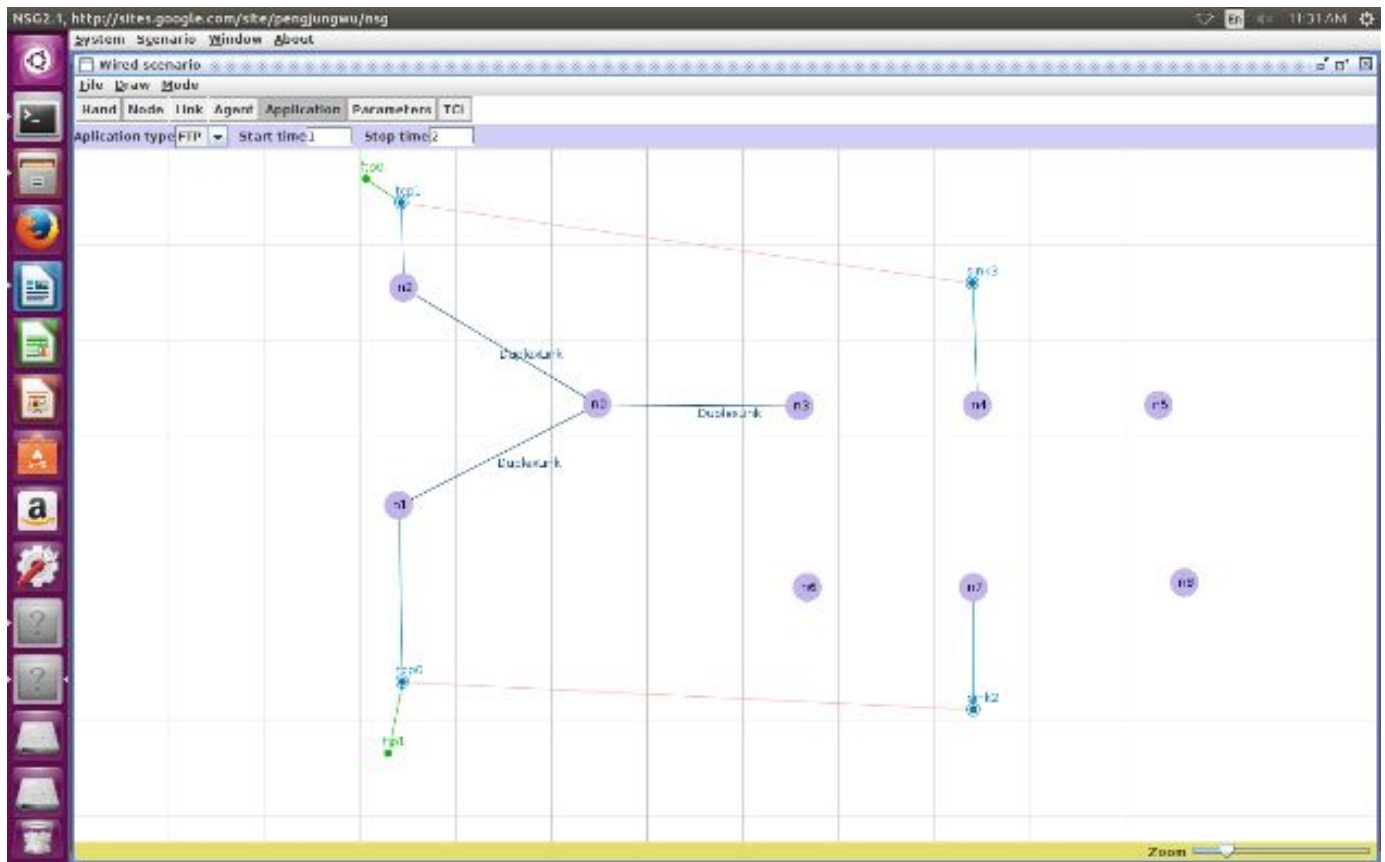
```
#=====
# Termination
#=====
#Define a 'finish' procedure
proc finish {} {
  global ns tracefile namfile
  $ns flush-trace
  close $tracefile
  close $namfile
  exec nam 3.nam &
  exit 0
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

3.awk:

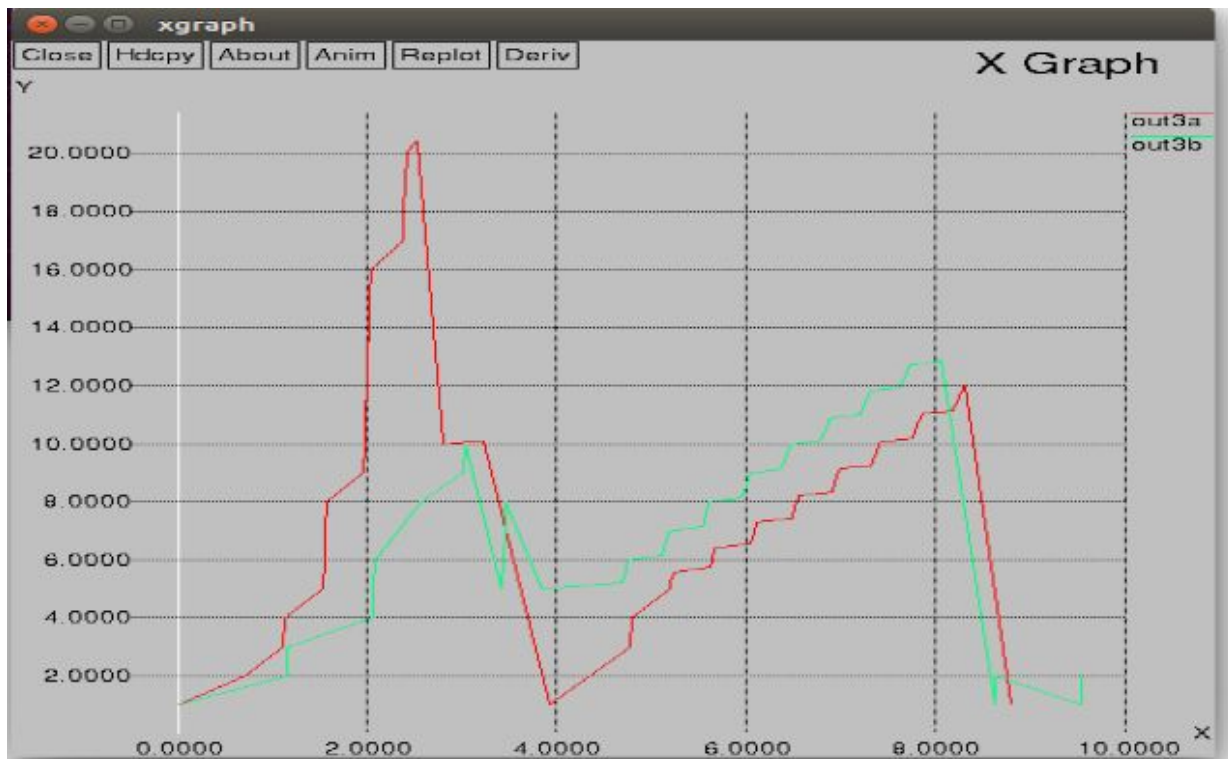
```
BEGIN{
}
{
  if($6=="cwnd_")
  {
    printf("%f\t%f\n",$1,$7);
  }
}
END{
}
```

Execution:

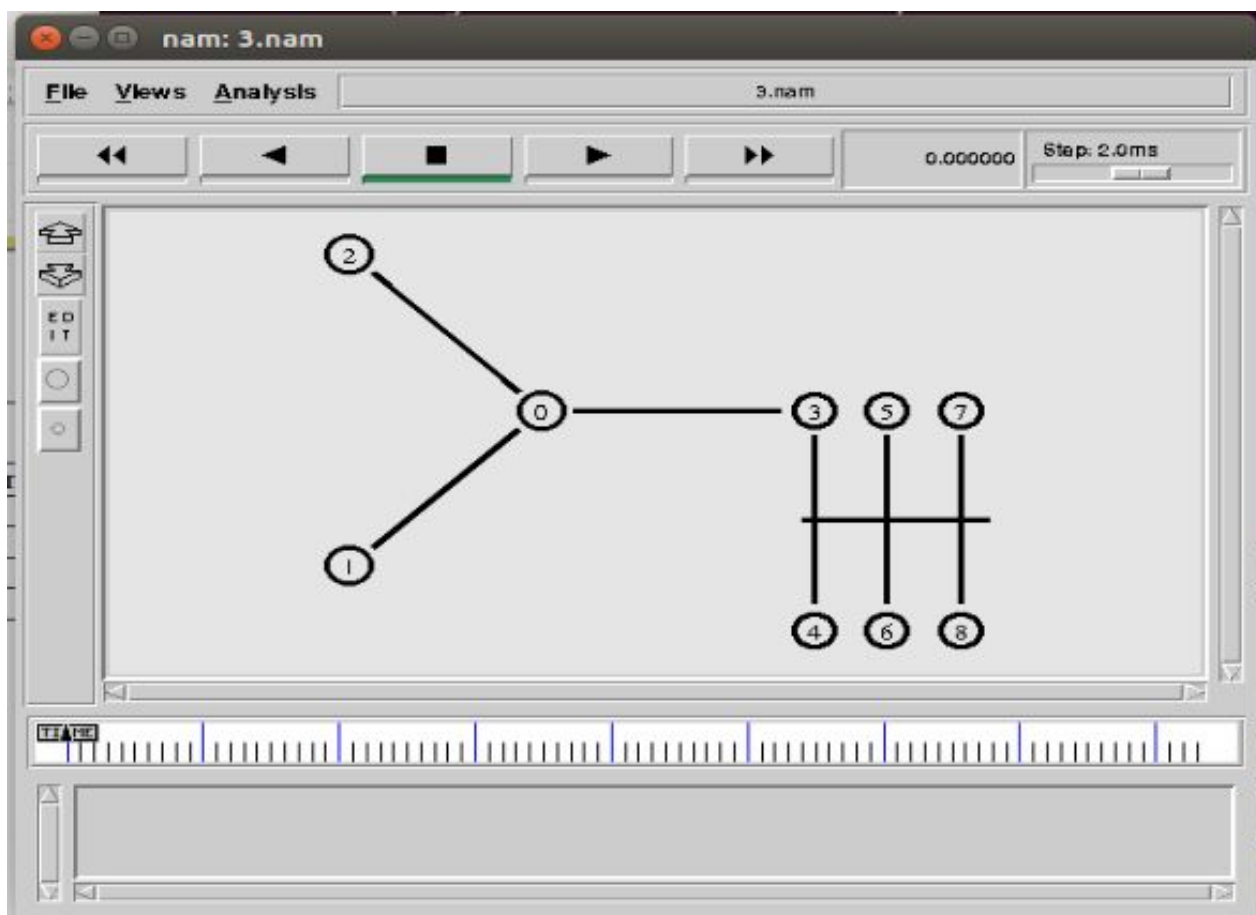
```
[student@student-Veriton-Series:~/ns2$ java -jar nsg2.1.jar
```



```
student@student-Veriton-Series:~/ns2$ ns 3.tcl
student@student-Veriton-Series:~/ns2$ gedit 3.awk
student@student-Veriton-Series:~/ns2$ awk -f 3.awk cwnd1.tr>out3a
student@student-Veriton-Series:~/ns2$ xgraph out3a
student@student-Veriton-Series:~/ns2$ awk -f 3.awk cwnd2.tr>out3b
student@student-Veriton-Series:~/ns2$ xgraph out3b
student@student-Veriton-Series:~/ns2$ xgraph out3a out3b
```



Nam file window:



4. Implement simple ESS and with transmitting nodes in wire-less LAN by simulation and determine the performance with respect to transmission of packets.**4.tcl**

```
# This script is created by NSG2 beta1
# <http://wushoupong.googlepages.com/nsg>
#=====
# Simulation parameters setup
#=====
set val(chan) Channel/WirelessChannel ;# channel type
set val(prop) Propagation/TwoRayGround ;# radio-propagation model
set val(netif) Phy/WirelessPhy ;# network interface type
set val(mac) Mac/802_11 ;# MAC type
set val(ifq) Queue/DropTail/PriQueue ;# interface queue type
set val(ll) LL ;# link layer type
set val(ant) Antenna/OmniAntenna ;# antenna model
set val(ifqlen) 50 ;# max packet in ifq
set val(nn) 2 ;# number of mobilenodes
set val(rp) DSDV ;# routing protocol
set val(x) 700 ;# X dimension of topography
set val(y) 444 ;# Y dimension of topography
set val(stop) 10.0 ;# time of simulation end

#=====
# Initialization
#=====
#Create a ns simulator
set ns [new Simulator]

#Setup topography object
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)
create-god $val(nn)

#Open the NS trace file
set tracefile [open 4.tr w]
$ns trace-all $tracefile

#Open the NAM trace file
set namfile [open 4.nam w]
$ns namtrace-all $namfile
$ns namtrace-all-wireless $namfile $val(x) $val(y)
set chan [new $val(chan)];#Create wireless channel

#=====
# Mobile node parameter setup
#=====
$ns node-config -adhocRouting $val(rp) \
```

```
-llType      $val(ll) \  
-macType     $val(mac) \  
-ifqType     $val(ifq) \  
-ifqLen      $val(ifqlen) \  
-antType     $val(ant) \  
-propType    $val(prop) \  
-phyType     $val(netif) \  
-channel     $chan \  
-topoInstance $topo \  
-agentTrace  ON \  
-routerTrace ON \  
-macTrace    ON \  
-movementTrace ON
```

```
#=====
```

```
# Nodes Definition
```

```
#=====
```

```
#Create 2 nodes
```

```
set n0 [$ns node]
```

```
$n0 set X_ 268
```

```
$n0 set Y_ 339
```

```
$n0 set Z_ 0.0
```

```
$ns initial_node_pos $n0 20
```

```
set n1 [$ns node]
```

```
$n1 set X_ 428
```

```
$n1 set Y_ 344
```

```
$n1 set Z_ 0.0
```

```
$ns initial_node_pos $n1 20
```

```
$ns at .1 "$n0 setdest 600 344 100"
```

```
$ns at .1 "$n1 setdest 300 339 100"
```

```
#=====
```

```
# Agents Definition
```

```
#=====
```

```
#Setup a TCP connection
```

```
set tcp0 [new Agent/TCP]
```

```
$ns attach-agent $n0 $tcp0
```

```
set sink1 [new Agent/TCPSink]
```

```
$ns attach-agent $n1 $sink1
```

```
$ns connect $tcp0 $sink1
```

```
$tcp0 set packetSize_ 1500
```

```
#=====
```

```
# Applications Definition
```

```
#=====
```

```
#Setup a FTP Application over TCP connection
```

```
set ftp0 [new Application/FTP]
```

```
$ftp0 attach-agent $tcp0
```

```
$ns at 1.0 "$ftp0 start"
```

```
$ns at 5.0 "$ftp0 stop"  
$ns at 10.0 "finish"
```

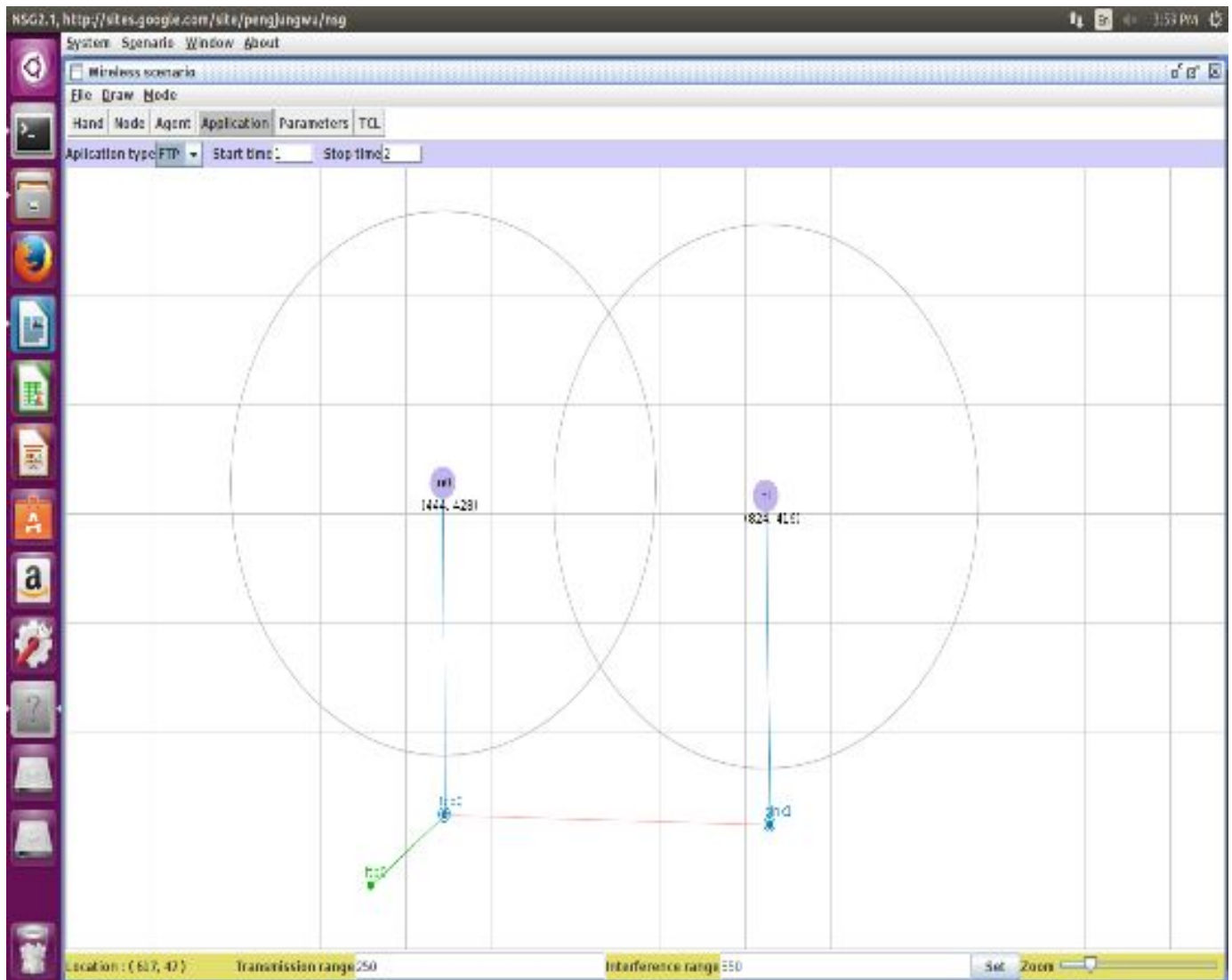
```
#=====
# Termination
#=====
#Define a 'finish' procedure
proc finish {} {
  global ns tracefile namfile
  $ns flush-trace
  close $tracefile
  close $namfile
  exec nam 4.nam &
  exit 0
}
for {set i 0} {$i < $val(nn)} {incr i} {
  $ns at $val(stop) "\n$i reset"
}
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"
$ns at $val(stop) "finish"
$ns at $val(stop) "puts \"done\" ; $ns halt"
$ns run
```

4.awk:

```
BEGIN{
  count1=0;
  fack1=0;
  time1=0;
}
{
  if($1=="r" && $3=="1" && $4=="AGT")
  {
    count1++;
    fack1= fack1+$8;
    time1=$2;
  }
}
END{
  printf("The throughput from n0 to n1 : %f Mbps\n",((count1*fack1*8)/(time1*1024)));
}
```

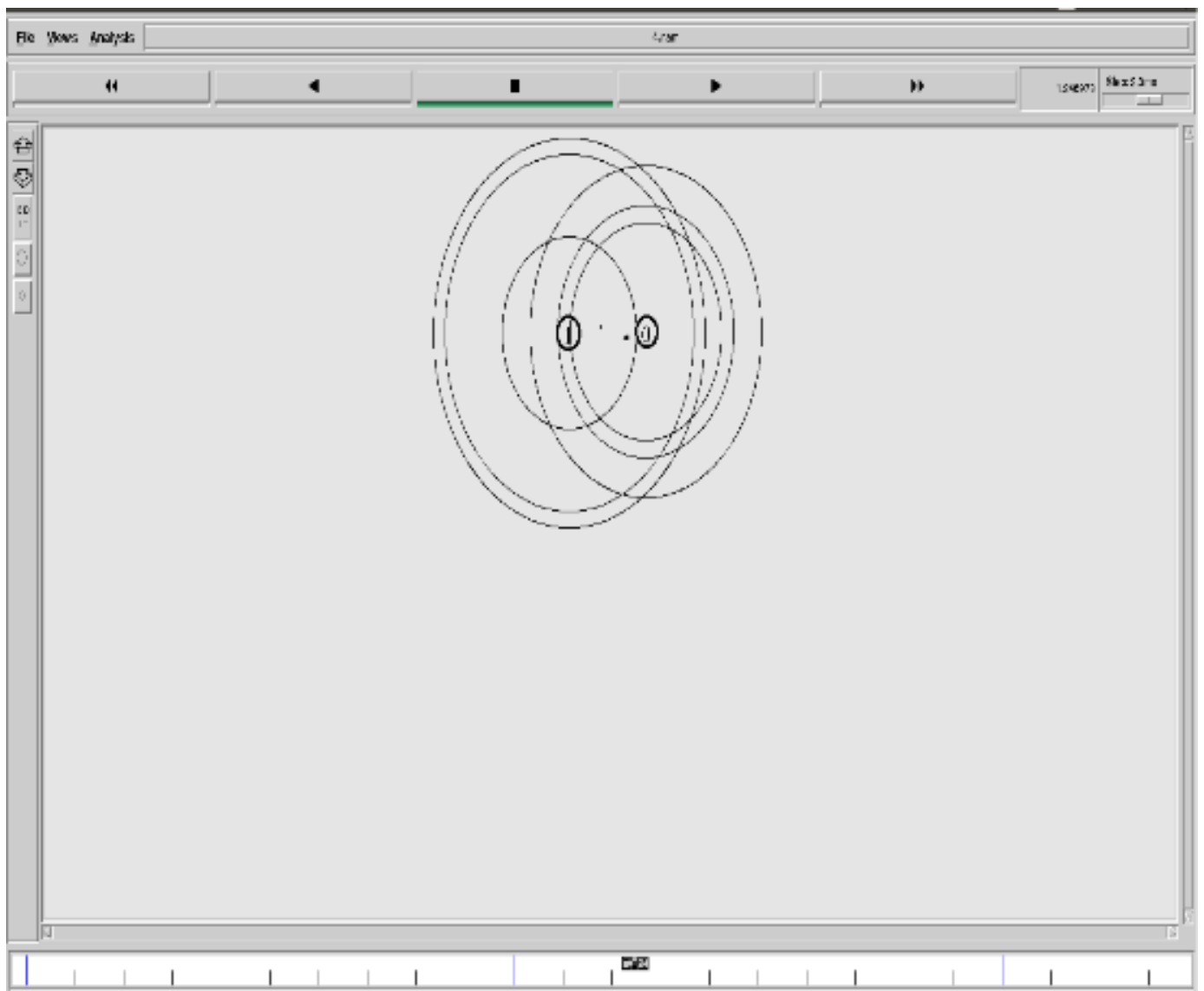
Execution:

student@student-Veriton-Series:~/ns2\$ java -jar nsg2.1.jar



```
student@student-Veriton-Series:~/ns2$ ns 4.tcl
student@student-Veriton-Series:~/ns2$ gedit 4.awk
student@student-Veriton-Series:~/ns2$ awk -f 4.awk 4.tr
```

The throughput from n0 to n1 : 97.158654 Mbps

Nam file window:

5. Implement and study the performance of GSM on NS2/NS3 (Using MAC layer) or equivalent environment.

Steps in Draw (D) Mode:

Step 1: Select GPRS Mobile Phones, GPRS Base Stations (BS), SGSN. Place on the working place to make required topology.

Step 2: Connect GPRS BS and SGSN using Point-to-Point link.

Step 3: provide a moving path for the GPRS Phone around the Base Station.

Step 4: Create a subnet between GPRS Phone and Base Station.

Steps in Edit(E) Mode:

Step 5: Go to N_Tools ☰ GPRS Network ☰ GPRS BS ☰ Assign frequency channels for Base Station.

Step 6: Double Click on GPRS Phone and open 'Action' window to attach the phone to BS at 1.00

Step 7: Open 'Application' window and add the following command to send packets to the receiver.

stcp -p 3000 -l 1024 1.0.2.1

Where 3000 is the port number, 1024 is the packet length and 1.0.2.1 is the receiving phone's IP Address.

NOTE: Application Start time should be greater than attach time

Step 8: Double Click on GPRS Phone and open 'Action' window to attach the phone to BS at 1.00

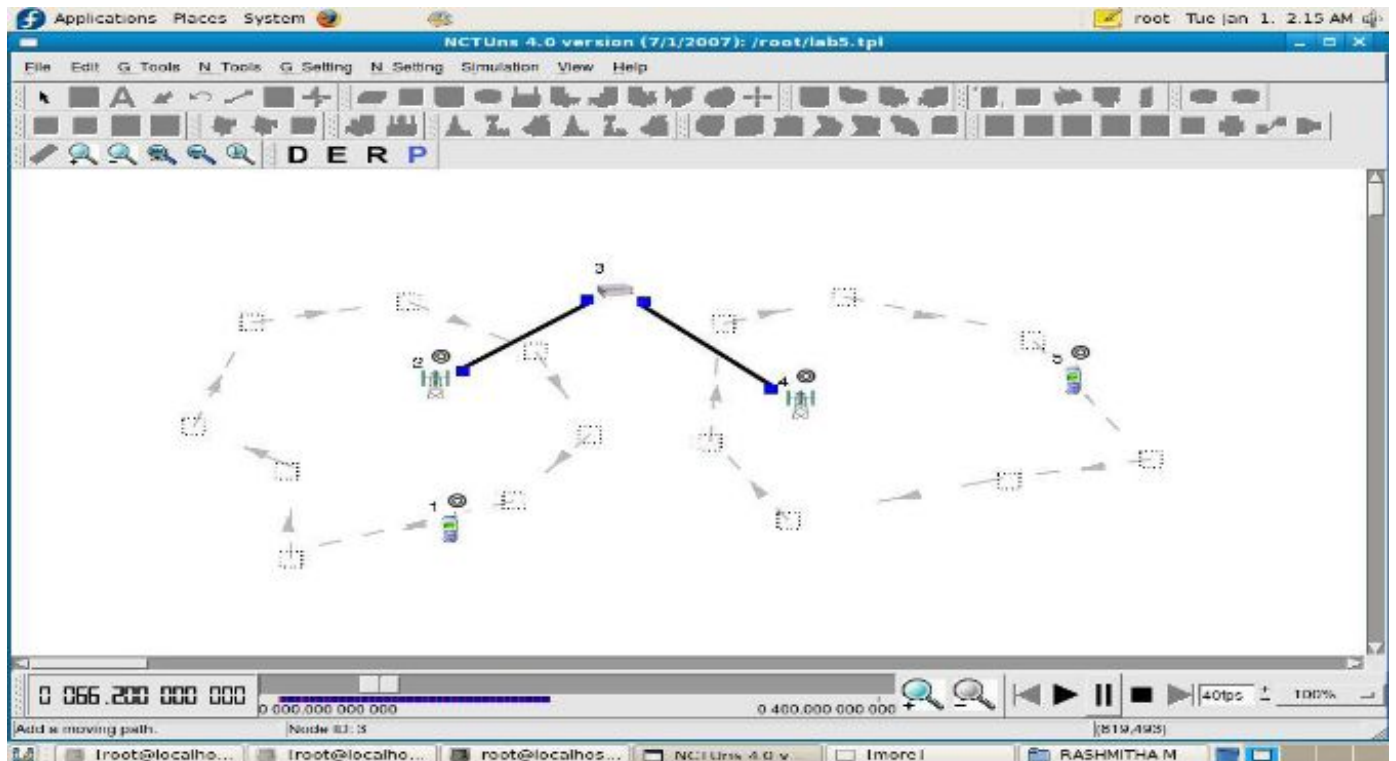
Step 9: Open 'Application' window and add the following command to send packets to the receiver.

rtcp -p 3000 -l 1024

Step 10: Run the Simulation in Run(R) Mode

Step 11: See the Output in Play Back (P) Mode.

Step 12: Go to G_Tools in menubar and select "View Packet Trace" || to view the trace file. Analyze the trace file.

Simulation Results:**Trace File**

The screenshot shows a terminal window displaying a network trace file. The trace contains various network events, including GPRS and GPRS_CTL messages, with timestamps and node identifiers. The terminal window has a title bar 'more' and a status bar at the bottom showing 'root@localhost:~' and 'NCTUns 4.0 versio...'.

```

002.5 TX 0 048 DATA <0 0> <2 2> 0 20 0 NONE
002.5 TX 0 047 DATA <0 0> <4 2> 0 20 0 NONE
002.5 RX 0 040 DATA <0 0> <2 2> 0 20 0 NONE
002.5 RX 0 040 DATA <0 0> <4 2> 0 20 0 NONE
002.5 RX 000 160 DATA <0 0> <2 2> 0 4 0 NONE
002.5 TX 000 175 DATA <0 0> <2 2> 0 4 0 NONE
002.5 RX 000 160 DATA <0 0> <2 2> 0 4 0 NONE
002.5 TX 000 160 DATA <0 0> <2 2> 0 4 0 NONE
GPRS TX 12600434 5770 GPRS_ACCESS <0 0> <5 2> 0 11 0 NONE 1
GPRS RX 12600448 5770 GPRS_ACCESS <0 0> <5 2> 0 11 0 NONE 1
GPRS TX 13693073 5770 GPRS_CTL <0 0> <5 2> 0 114 0 NONE 2
GPRS RX 13693086 5770 GPRS_CTL <0 0> <5 2> 0 114 0 NONE 2
GPRS TX 13693049 5770 GPRS_CTL <0 0> <5 2> 1 114 0 NONE 2
GPRS RX 13693062 5770 GPRS_CTL <0 0> <5 2> 1 114 0 NONE 2
GPRS TX 13693425 5770 GPRS_CTL <0 0> <5 2> 2 114 0 NONE 2
GPRS RX 13693438 5770 GPRS_CTL <0 0> <5 2> 2 114 0 NONE 2
GPRS TX 13731601 5770 GPRS_CTL <0 0> <5 2> 3 114 0 NONE 2
GPRS RX 13731614 5770 GPRS_CTL <0 0> <5 2> 3 114 0 NONE 2
GPRS TX 13805204 5770 GPRS_CTL <0 0> <2 0> 0 114 0 NONE 127
GPRS RX 13805237 5770 GPRS_CTL <0 0> <2 0> 0 114 0 NONE 127
GPRS TX 13852800 5770 GPRS_CTL <0 0> <2 0> 1 114 0 NONE 127
GPRS RX 13852813 5770 GPRS_CTL <0 0> <2 0> 1 114 0 NONE 127
GPRS TX 13899076 5770 GPRS_CTL <0 0> <2 0> 2 114 0 NONE 127

```

6. Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.**Steps in Draw (D) Mode:**

Step 1: Select GPRS Mobile Phone, GPRS Base Station (BS), SGSN, Pseudo Switch, GGSN, Router and Host. Place on the working place to make required topology.

Step 2: Connect GPRS BS ↔ SGSN ↔ Pseudo Switch ↔ GGSN ↔ Router ↔ Host using Point-to-Point link.

Step 3: provide a moving path for the GPRS Phone around the Base Station.

Step 4: Create a subnet between GPRS Phone and Base Station.

Steps in Edit(E) Mode:

Step 5: Go to N_Tools ↔ GPRS Network ↔ GPRS BS ↔ Assign frequency channels for Base Station.

Step 6: Double Click on GPRS Phone and open 'Action' window to attach the phone to BS at 1.00

Step 7: Double Click on GPRS Phone and open 'Application' window and add the following command to send packets to the receiver.

stcp -p 3000 -l 1024 1.0.2.1

Where 3000 is the port number, 1024 is the packet length and 1.0.2.1 is the receiving phone's IP Address.

NOTE: Application Start time should be greater than attach time

Step 8: Double click on Host and open 'Application' window and add the following command to receive the packets from the sender.

rtcp -p 3000 -l 1024

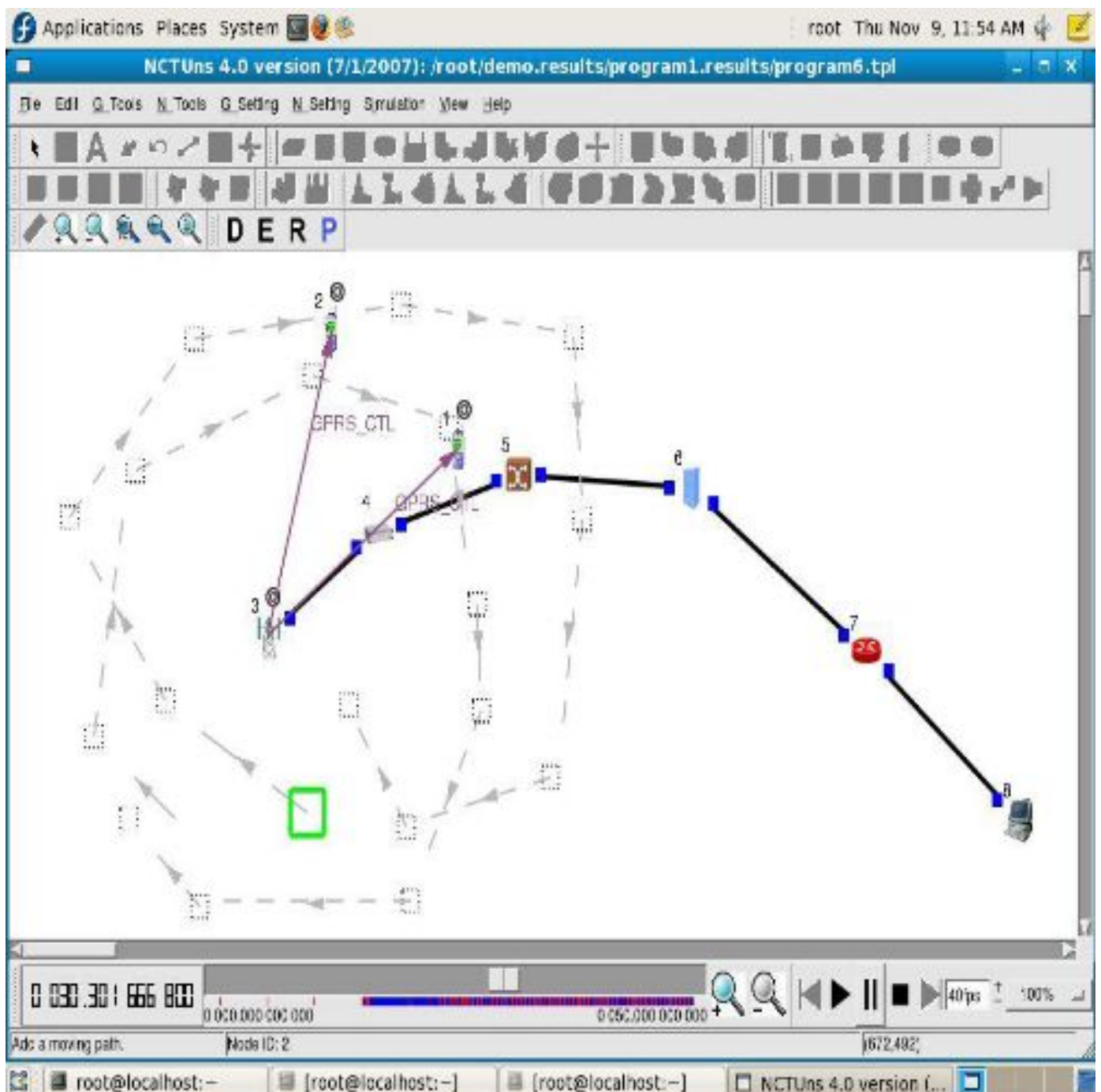
Step 9: Open the Node Editor of the Host and set Incoming_throughput statistics.

Step 10: Double Click on Router, Open "Node Editor" and double click on MAC8023 layer. Enable the log statistics and set Incoming_throughput and Outgoing_throughput.

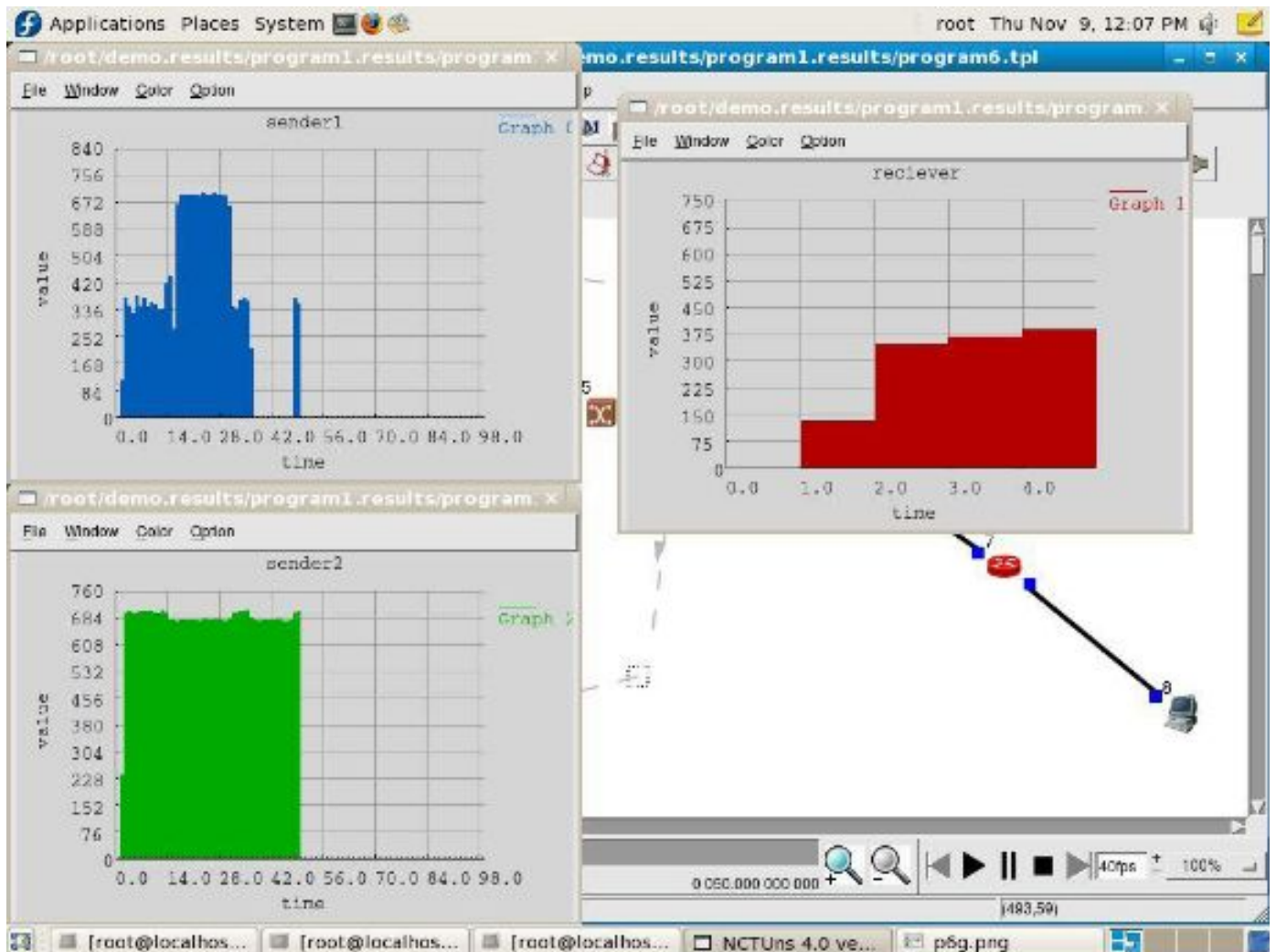
Step 11: Run the Simulation in Run(R) Mode

Step 12: See the Output in Play Back (P) Mode.

Step 13 : Go to G_Tools in menubar and select "Plot Graph" to plot the required Outgoing and Incoming throughput.

Simulation Results:

Topology



Graph for Outgoing Throughput & Incoming Throughput

7. Write a program for error detecting code using CRC-CCITT (16- bits).

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The CRC calculation or cyclic redundancy check was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC. Also each data block on your hard disk has a CRC value attached to it. Modern computer world cannot do without these CRC calculations. So let's see why they are so widely used. The answer is simple; they are powerful, detect many types of errors and are extremely fast to calculate especially when dedicated hardware chips are used.

The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school. We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation— by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar.

$$\begin{array}{r}
 101 \\
 \hline
 10011 \, / \, 1101101 \\
 10011 \, | \, | \\
 \hline
 10000 \, | \\
 00000 \, | \\
 \hline
 100001 \\
 10011 \\
 \hline
 1110 = 14 \text{ remainder}
 \end{array}$$

- The message bits are appended with c zero bits; this augmented message is the dividend
- A predetermined c+1-bit binary sequence, called the generator polynomial, is the divisor
- The checksum is the c-bit remainder that results from the division operation

With decimal calculations you can quickly check that 109 divided by 19 gives a quotient of 5 with 14 as the remainder. But what we also see in the scheme is that every bit extra to check only costs one binary comparison and in 50% of the cases one binary subtraction. You can easily increase the number of bits of the test data string—for example to 56 bits if we use our example value

"Lammert"—and the result can be calculated with 56 binary comparisons and an average of 28 binary subtractions. This can be implemented in hardware directly with only very few transistors involved. Also software algorithms can be very efficient.

All of the CRC formulas you will encounter are simply checksum algorithms based on modulo-2 binary division where we ignore carry bits and in effect the subtraction will be equal to an exclusive or operation. Though some differences exist in the specifics across different CRC formulas, the basic mathematical process is always the same:

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder.

So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.

	CRC - CCITT	CRC - 16	CRC - 32
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

International Standard CRC Polynomials

Source Code:

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

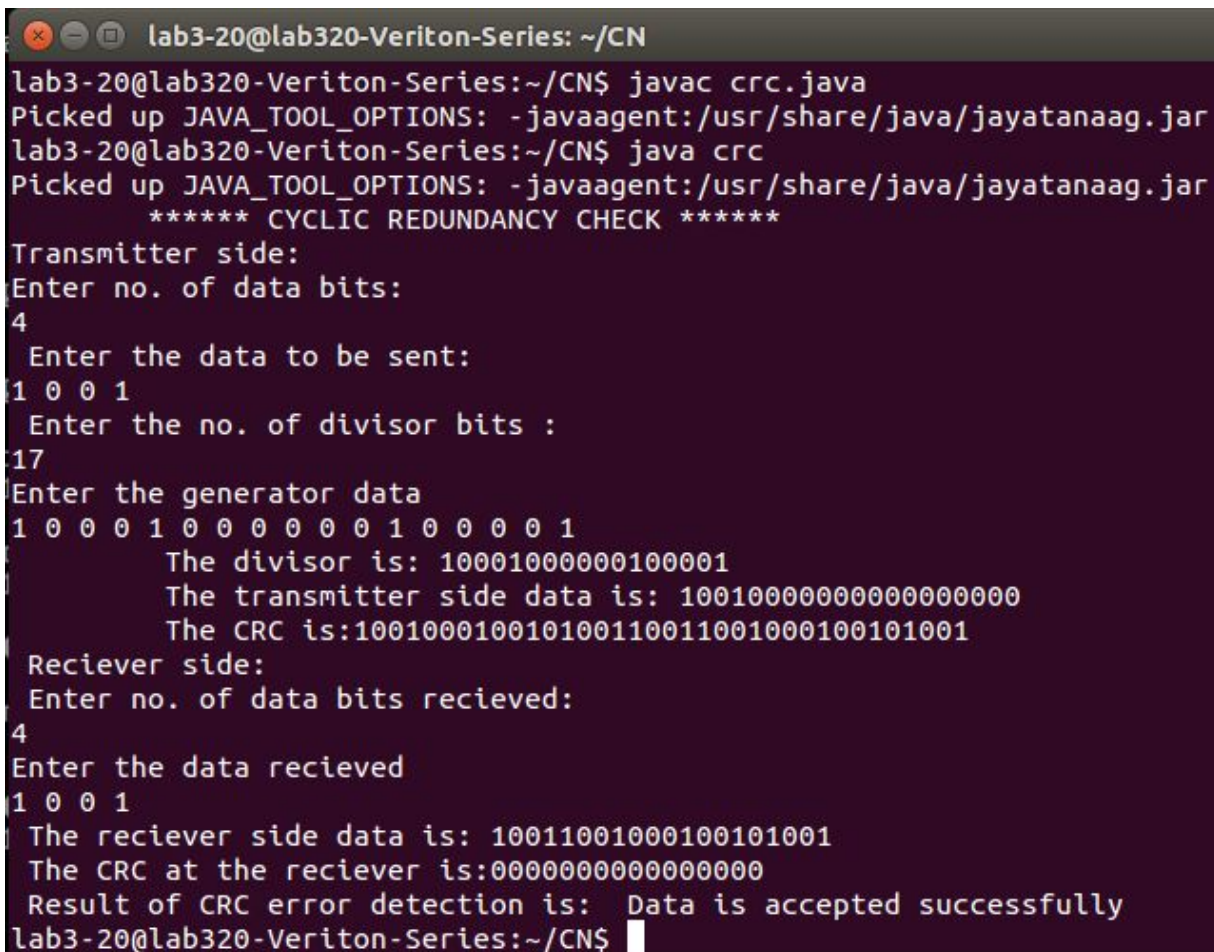
class crc
{
    public static void main(String args[])
    {
        int i,j,n,g,check,flag,s,a;
        int arr[]=new int[30],gen[]=new int[20],b[]=new int[20],q[]=new int[20],x[]=new int[20];
        check = 0;
        flag=0;
        System.out.println("\t***** CYCLIC REDUNDANCY CHECK *****");
        System.out.println("Transmitter side:");
        System.out.println("Enter no. of data bits:");
        Scanner in = new Scanner(System.in);
        n= in.nextInt();
        System.out.println(" Enter the data to be sent:");
        for(i=0;i<n;i++)
            arr[i]=in.nextInt();
        System.out.println(" Enter the no. of divisor bits : ");
```

```
g = in.nextInt();
do
{
    System.out.println("Enter the generator data");
    for(j=0;j<g;j++)
        gen[j] = in.nextInt();
}while(gen[0]!=1);
System.out.print("\t The divisor is: ");
for(j=0;j<g;j++)
    System.out.print(gen[j]);
System.out.println();
a=n+(g-1);
System.out.print("\t The transmitter side data is: ");
for(i=0;i<j;i++)
    arr[n+i] =0;
for(i=0;i<a;i++)
    System.out.print(arr[i]);
System.out.println();
for(i=0;i<n;i++)
    q[i]=arr[i];
for(i=0;i<n;i++)
{
    if(arr[i]==0)
    {
        for(j=i;j<g;j++)
            arr[j]=arr[j]^0;
    }
    else
    {
        for(int k=0;k<17;k++)
        {
            arr[i+k]=arr[i+k]^gen[k];
        }
    }
}
System.out.print("\t The CRC is:");
for(i=n;i<a;i++)
    System.out.print(arr[i]);
for(i=n;i<a;i++)
    q[i]=arr[i];
//for(i=0;i<a;i++)
//System.out.print(q[i]);
```



```
System.out.println();
System.out.println(" Reciever side:");
System.out.println(" Enter no. of data bits recieved:");
n=in.nextInt();
System.out.println("Enter the data recieved");
for(i=0;i<n;i++)
arr[i]=in.nextInt();
for(i=n;i<a;i++)
for(j=g-1;j<0;j--)
arr[i]=q[j];
System.out.print(" The reciever side data is: ");
for(i=0;i<a;i++)
System.out.print(arr[i]);
for(i=0;i<n;i++)
q[i]=arr[i];
for(i=0;i<n;i++)
{
    if(arr[i]==0)
    {
        for(j=i;j<g+i;j++)
            arr[j]=arr[j]^0;
    }
    else
    {
        for(int k=0;k<17;k++)
        {
            arr[i+k]=arr[i+k]^gen[k];
        }
    }
}
System.out.println();
System.out.print(" The CRC at the reciever is:");
for(i=n;i<a;i++)
System.out.print(arr[i]);
for(i=n;i<a;i++)
{
    if(arr[i]==1)
    {
        flag=1;
        break;
    }
    else
        check=0;
```

```
}  
System.out.println();  
System.out.print(" Result of CRC error detection is: ");  
if(flag==0 && check==0)  
System.out.println(" Data is accepted successfully");  
else  
System.out.println(" Resend the data again");  
}  
}
```

Output:

```
lab3-20@lab320-Veriton-Series: ~/CN  
lab3-20@lab320-Veriton-Series:~/CN$ javac crc.java  
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar  
lab3-20@lab320-Veriton-Series:~/CN$ java crc  
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar  
***** CYCLIC REDUNDANCY CHECK *****  
Transmitter side:  
Enter no. of data bits:  
4  
Enter the data to be sent:  
1 0 0 1  
Enter the no. of divisor bits :  
17  
Enter the generator data  
1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1  
The divisor is: 10001000000100001  
The transmitter side data is: 10010000000000000000  
The CRC is:1001000100101001100110010001001001  
Reciever side:  
Enter no. of data bits recieved:  
4  
Enter the data recieved  
1 0 0 1  
The reciever side data is: 10011001000100101001  
The CRC at the reciever is:000000000000000000  
Result of CRC error detection is: Data is accepted successfully  
lab3-20@lab320-Veriton-Series:~/CN$
```

8. Write a program to find the shortest path between vertices using bellman-ford algorithm

Distance Vector Algorithm is a decentralized routing algorithm that requires that each router simply inform its neighbours of its routing table. For each network path, the receiving routers pick the neighbour advertising the lowest cost, then add this entry into its routing table for re-advertisement. To find the shortest path, Distance Vector Algorithm is based on one of two basic algorithms: Bellman-Ford and Dijkstra algorithms. Routers that use this algorithm have to maintain the distance tables (which is a one- dimension array -- "a vector"), which tell the distances and shortest path to sending packets to each node in the network. The information in the distance table is always up date by exchanging information with the neighbouring nodes. The number of data in the table equals to that of all nodes in networks (excluded itself). The columns of table represent the directly attached neighbours whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance/or time to transmit on that path (we call this as "cost"). The measurements in this algorithm are the number of hops, latency, the number of outgoing packets, etc. The Bellman-Ford algorithm is an algorithm that computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph. It is slower than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers. Negative edge weights are found in various applications of graphs, hence the usefulness of this algorithm. If a graph contains a "negative cycle" (i.e. a cycle whose edges sum to a negative value) that is reachable from the source, then there is no cheapest path: any path that has a point on the negative cycle can be made cheaper by one more walk around the negative cycle. In such a case, the Bellman-Ford algorithm can detect negative cycles and report their existence.

Source Code:

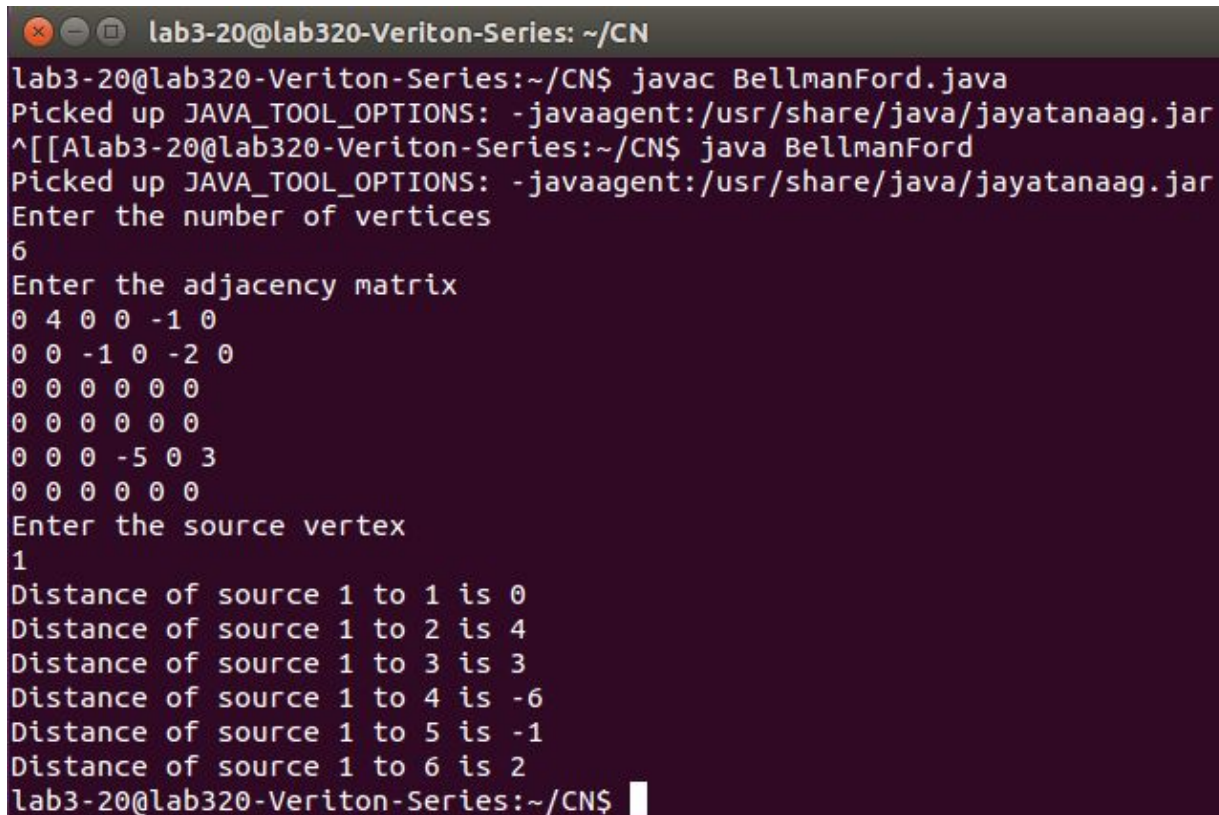
```
import java.util.Scanner;
public class BellmanFord
{
    private int distances[];
    private int numberofvertices;
    public static final int MAX_VALUE = 999;
    public BellmanFord(int numberofvertices)
    {
        this.numberofvertices= numberofvertices;
        distances = new int[numberofvertices+1];
    }
    public void BellmanFordEvaluation(int source,int adjacencymatrix[][])
    {
        for(int node=1;node<=numberofvertices;node++)
        {
            distances[node]=MAX_VALUE;
        }
        distances[source]=0;
    }
}
```

```
for(int node=1;node<=numberofvertices-1;node++)
{
    for(int sourcenode=1;sourcenode<=numberofvertices;sourcenode++)
    {
        for(int destinationnode=1;destinationnode<=numberofvertices;
        destinationnode++)
        {
            if(adjacencymatrix[sourcenode][destinationnode]!=MAX_VALUE)
            {
                if(distances[destinationnode]>distances[sourcenode] +
adjacencymatrix[sourcenode][destinationnode])
                distances[destinationnode]=distances[sourcenode]+adjacencymatrix[sourcenode]
[destinationnode];
            }
        }
    }
}
for (int sourcenode=1;sourcenode<=numberofvertices;sourcenode++)
{
    for(int destinationnode=1;destinationnode<=numberofvertices;
    destinationnode++)
    {
        if(adjacencymatrix[sourcenode][destinationnode]!=MAX_VALUE)
        {
            if(distances[destinationnode]>distances[sourcenode] +
adjacencymatrix[sourcenode][destinationnode])
                System.out.println("The graph contains negative edge cycle");
        }
    }
}
for(int vertex=1;vertex<=numberofvertices;vertex++)
{
    System.out.println("Distance of source "+source+" to "+vertex+" is
    "+distances[vertex]);
}
}
public static void main(String args[])
{
    int numberofvertices=0;
    int source,destination;
    Scanner scanner = new Scanner(System.in);
    System.out.println("Enter the number of vertices");
    numberofvertices=scanner.nextInt();
    int adjacencymatrix[][]= new int[numberofvertices+1][numberofvertices+1];
    System.out.println("Enter the adjacency matrix");
    for(int sourcenode=1;sourcenode<=numberofvertices;sourcenode++)
    {
        for(int destinationnode=1;destinationnode<=numberofvertices;
        destinationnode++)
```

```

        {
            adjacencymatrix[sourcenode][destinationnode]=scanner.nextInt();
            if(sourcenode==destinationnode)
            {
                adjacencymatrix[sourcenode][destinationnode]=0;
                continue;
            }
            if(adjacencymatrix[sourcenode][destinationnode]==0)
            {
                adjacencymatrix[sourcenode][destinationnode]=MAX_VALUE;
            }
        }
    }
    System.out.println("Enter the source vertex");
    source=scanner.nextInt();
    BellmanFord bellmanford = new BellmanFord(numberofvertices);
    bellmanford.BellmanFordEvaluation(source,adjacencymatrix);
}
}

```

Output:


```

lab3-20@lab320-Veriton-Series: ~/CN
lab3-20@lab320-Veriton-Series:~/CN$ javac BellmanFord.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
^[[Alab3-20@lab320-Veriton-Series:~/CN$ java BellmanFord
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Enter the number of vertices
6
Enter the adjacency matrix
0 4 0 0 -1 0
0 0 -1 0 -2 0
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 -5 0 3
0 0 0 0 0 0
Enter the source vertex
1
Distance of source 1 to 1 is 0
Distance of source 1 to 2 is 4
Distance of source 1 to 3 is 3
Distance of source 1 to 4 is -6
Distance of source 1 to 5 is -1
Distance of source 1 to 6 is 2
lab3-20@lab320-Veriton-Series:~/CN$

```

9. Using TCP/IP sockets, write a client – server program to make the client send the file name and to make the server send back the contents of the requested file if present.

Socket is an interface which enables the client and the server to communicate and pass on information from one another. Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server. When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

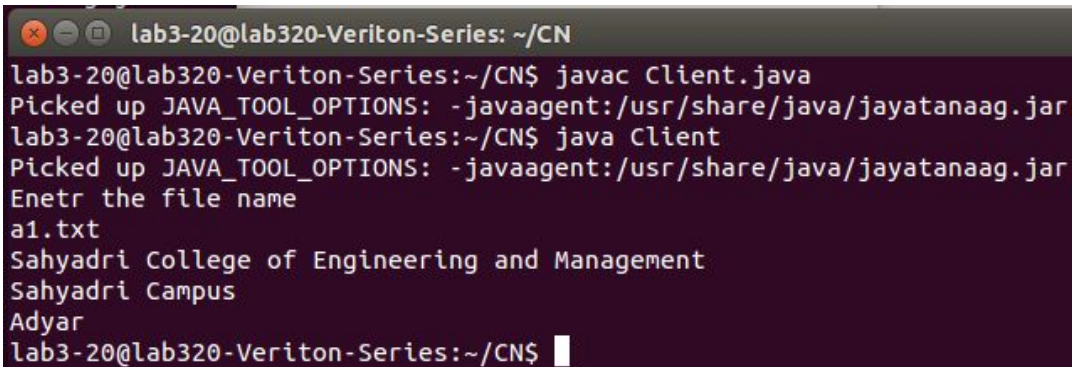
Source Code:**Server Program:**

```
import java.net.*;
import java.io.*;
public class Cserver
{
    public static void main(String args[]) throws Exception
    {
        ServerSocket sersock = new ServerSocket(4000);
        System.out.println("Server ready for connection ");
        Socket sock = sersock.accept();
        System.out.println("Connection is successful and waiting for chatting");
        InputStream istream = sock.getInputStream();
        BufferedReader fileRead = new BufferedReader (new InputStreamReader(istream));
        String fname = fileRead.readLine();
        BufferedReader contentRead = new BufferedReader (new FileReader(fname));
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter (ostream,true);
        String str;
        while((str=contentRead.readLine())!=null)
        {
            pwrite.println(str);
        }
        sock.close();
        sersock.close();
        pwrite.close();
        fileRead.close();
        contentRead.close();
    }
}
```

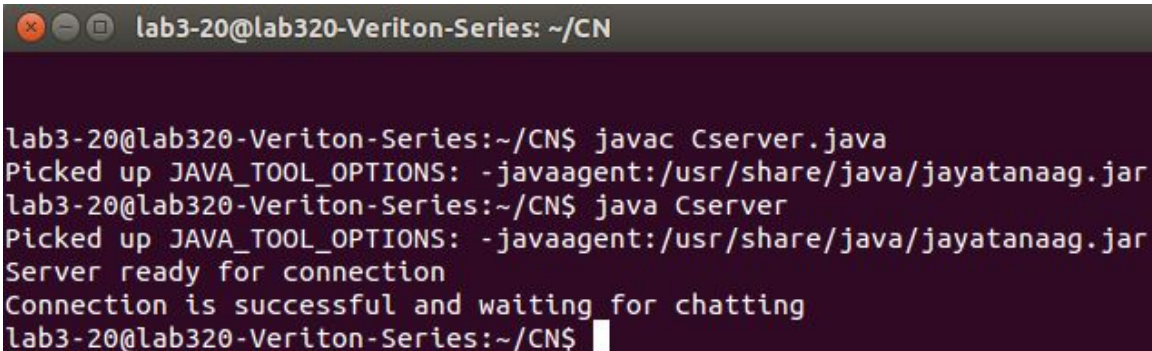
Client Program:

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

public class Client
{
    public static void main(String[] args) throws Exception
    {
        Socket sock = new Socket("127.0.0.1",4000);
        System.out.println("Enetr the file name");
        BufferedReader keyRead = new BufferedReader(new InputStreamReader(System.in));
        String fname = keyRead.readLine();
        OutputStream ostream = sock.getOutputStream();
        PrintWriter pwrite = new PrintWriter(ostream,true);
        pwrite.println(fname);
        InputStream istream = sock.getInputStream();
        BufferedReader socketRead = new BufferedReader(new InputStreamReader
(istream));
        String str;
        while((str=socketRead.readLine())!=null)
        {
            System.out.println(str);
        }
        pwrite.close();
        socketRead.close();
        keyRead.close();
    }
}
```


Output:**Server Side**A terminal window titled 'lab3-20@lab320-Veriton-Series: ~/CN' with a dark background. The text inside shows the compilation and execution of a Java client program. The user enters 'javac Client.java', followed by 'java Client'. The program prompts for a file name, and the user enters 'a1.txt'. The program then prints the address of Sahyadri College of Engineering and Management.

```
lab3-20@lab320-Veriton-Series: ~/CN$ javac Client.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
lab3-20@lab320-Veriton-Series:~/CN$ java Client
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Enetr the file name
a1.txt
Sahyadri College of Engineering and Management
Sahyadri Campus
Adyar
lab3-20@lab320-Veriton-Series:~/CN$
```

Client Side:A terminal window titled 'lab3-20@lab320-Veriton-Series: ~/CN' with a dark background. The text inside shows the compilation and execution of a Java server program. The user enters 'javac Cserver.java', followed by 'java Cserver'. The program prints 'Server ready for connection' and 'Connection is successful and waiting for chatting'.

```
lab3-20@lab320-Veriton-Series:~/CN$ javac Cserver.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
lab3-20@lab320-Veriton-Series:~/CN$ java Cserver
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Server ready for connection
Connection is successful and waiting for chatting
lab3-20@lab320-Veriton-Series:~/CN$
```

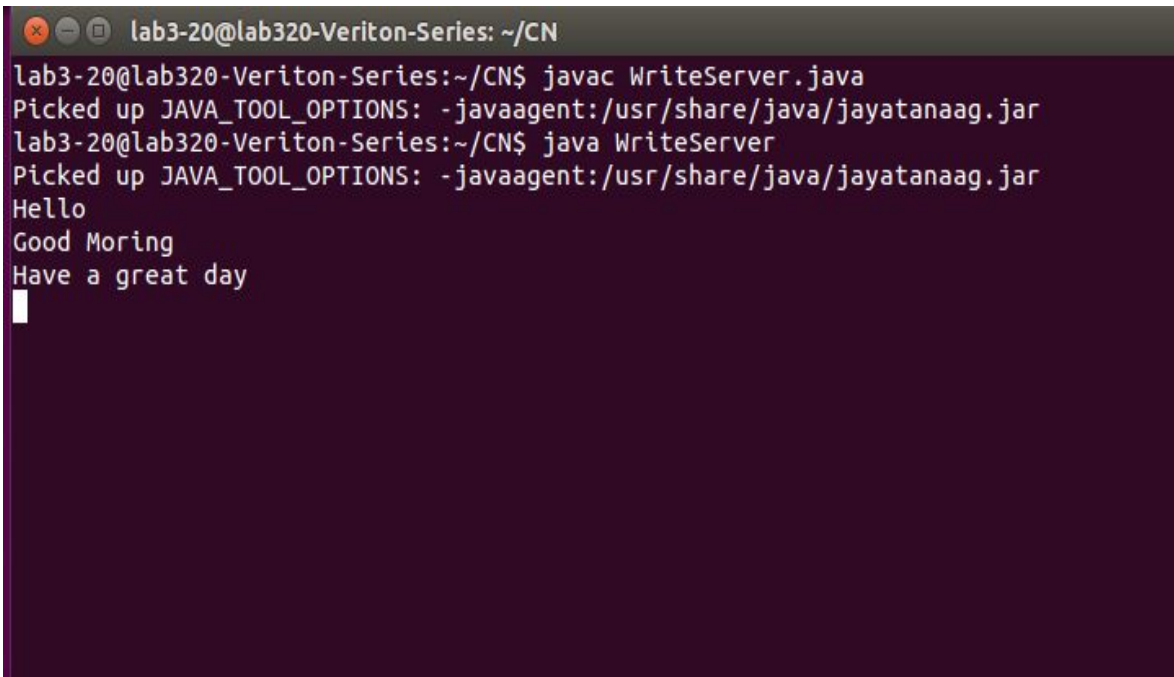

10. Write a program on datagram socket for client/server to display the messages on client side, typed at the server side.

A datagram socket is the one for sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order.

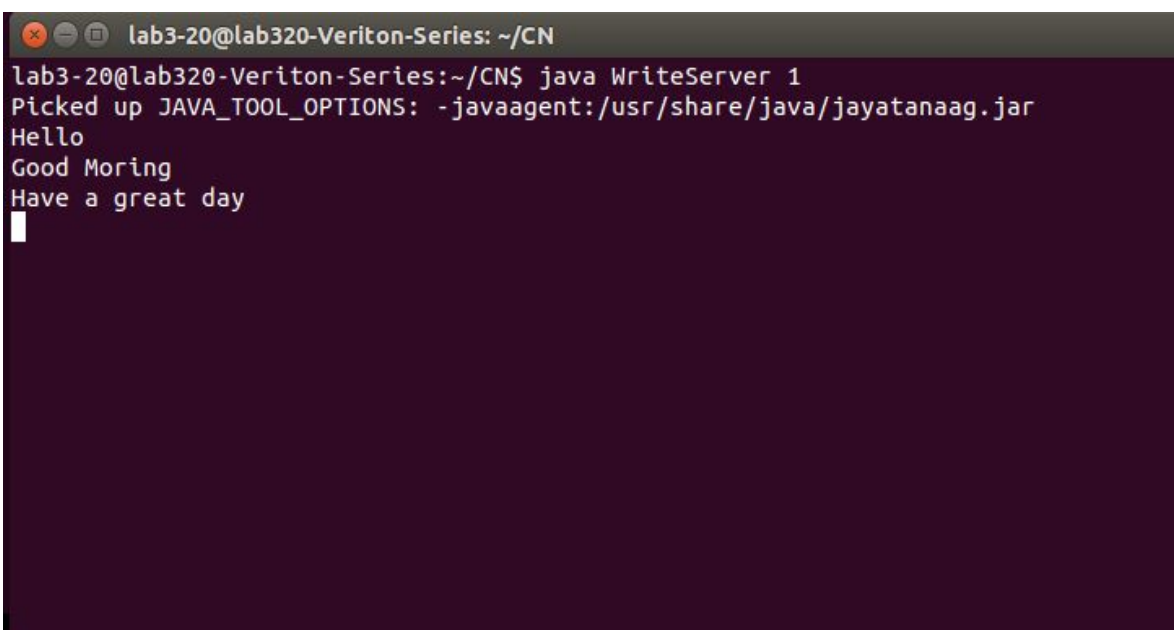
Source Code:

```
import java.net.*;
class WriteServer
{
    public static int serverPort = 1150;
    public static int clientPort = 1160;
    public static int buffer_size = 1024;
    public static DatagramSocket ds;
    public static byte buffer[] = new byte[buffer_size];
    public static void TheServer() throws Exception
    {
        int pos=0;
        while(true)
        {
            int c = System.in.read();
            switch(c)
            {
                case -1: System.out.println("Server Quits");
                    return;
                case '\r':break;
                case '\n':ds.send(new
DatagramPacket(buffer,pos,InetAddress.getLocalHost(),clientPort));
                    pos=0;
                    break;
                default:buffer[pos++]=(byte) c;
            }
        }
    }
    public static void TheClient() throws Exception
    {
        while(true)
        {
            DatagramPacket p = new DatagramPacket(buffer,buffer.length);
            ds.receive(p);
            System.out.println(new String (p.getData(),0,p.getLength()));
        }
    }
    public static void main(String args[]) throws Exception
    {
        if(args.length==1)
        {
```

```
        ds = new DatagramSocket(serverPort);
        TheServer();
    }
    else
    {
        ds = new DatagramSocket(clientPort);
        TheClient();
    }
}
```

OUTPUT:

```
lab3-20@lab320-Veriton-Series: ~/CN
lab3-20@lab320-Veriton-Series:~/CN$ javac WriteServer.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
lab3-20@lab320-Veriton-Series:~/CN$ java WriteServer
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Hello
Good Moring
Have a great day
█
```



```
lab3-20@lab320-Veriton-Series: ~/CN
lab3-20@lab320-Veriton-Series:~/CN$ java WriteServer 1
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Hello
Good Moring
Have a great day
█
```

11. Write a program for simple RSA algorithm to encrypt and decrypt the data.

RSA is an example of public key cryptography. It was developed by Rivest, Shamir and Adelman. The RSA algorithm can be used for both public key encryption and digital signatures. Its security is based on the difficulty of factoring large integers. The RSA algorithm's efficiency requires a fast method for performing the modular exponentiation operation. A less efficient, conventional method includes raising a number (the input) to a power (the secret or public key of the algorithm, denoted e and d , respectively) and taking the remainder of the division with N . A straight-forward implementation performs these two steps of the operation sequentially: first, raise it to the power and second, apply modulo. The RSA algorithm comprises of three steps, which are depicted below:

Key Generation Algorithm

1. Generate two large random primes, p and q , of approximately equal size such that their product $n = p \cdot q$
2. Compute $n = p \cdot q$ and Euler's totient function (ϕ) $\phi(n) = (p-1)(q-1)$.
3. Choose an integer e , $1 < e < \phi$, such that $\gcd(e, \phi) = 1$.
4. Compute the secret exponent d , $1 < d < \phi$, such that $e \cdot d \equiv 1 \pmod{\phi}$.
5. The public key is (e, n) and the private key is (d, n) . The values of p , q , and ϕ should also be kept secret.

Encryption

Sender A does the following:-

1. Using the public key (e, n)
2. Represents the plaintext message as a positive integer M
3. Computes the cipher text $C = M^e \pmod{n}$.
4. Sends the cipher text C to B (Receiver).

Decryption

Recipient B does the following:-

1. Uses his private key (d, n) to compute $M = C^d \pmod{n}$.
2. Extracts the plaintext from the integer representative m .

Source Code:

```
import java.util.*;
import java.io.*;
class rsa
{
    static int mult(int x,int y,int n)
    {
        int k=1;
        int j;
        for (j=1; j<=y; j++)
            k = (k * x) % n;
        return ( int) k;
    }
    public static void main (String arg[])throws Exception
    {
```

```
Scanner s=new Scanner(System.in);
InputStreamReader r=new InputStreamReader(System.in);
BufferedReader br=new BufferedReader(r);
String msg1;
int pt[]=new int[100];
int ct[]=new int[100];
int a,b, n, d, e,Z, p, q, i,temp,et;
System.out.println("Enter prime No.s p,q :");
p=s.nextInt();
q=s.nextInt();
n = p*q;
Z=(p-1)*(q-1);
System.out.println("\nSelect e value:");
e=s.nextInt();
System.out.printf("Enter message : ");
msg1=br.readLine();
char msg[]=msg1.toCharArray();
for(i=0;i<msg.length;i++)
    pt[i]=msg[i];
for(d=1;d<Z;++d)
    if(((e*d)%Z)==1) break;
    System.out.println("p="+p+"q="+q+"\tn="+n+"\tz="+Z+"\te="+e+
    "\td="+d);
    System.out.println("\nCipher Text = ");
for(i=0; i<msg.length; i++)
    ct[i] = mult(pt[i], e,n);
for(i=0; i<msg.length; i++)
    System.out.print("\t"+ct[i]);
System.out.println("\nPlain Text = ");
for(i=0; i<msg.length; i++)
    pt[i] = mult(ct[i], d,n) ;
for(i=0; i<msg.length; i++)
    System.out.print((char)pt[i]);
    }
}
```

OUTPUT:

```
lab3-20@lab320-Veriton-Series: ~/CN
lab3-20@lab320-Veriton-Series:~/CN$ javac rsa.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
lab3-20@lab320-Veriton-Series:~/CN$ java rsa
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Enter prime No.s p,q :
13
11

Select e value:
7
Enter message : Computer Networks Laboratory
p=13    q=11    n=143    z=120    e=7    d=103

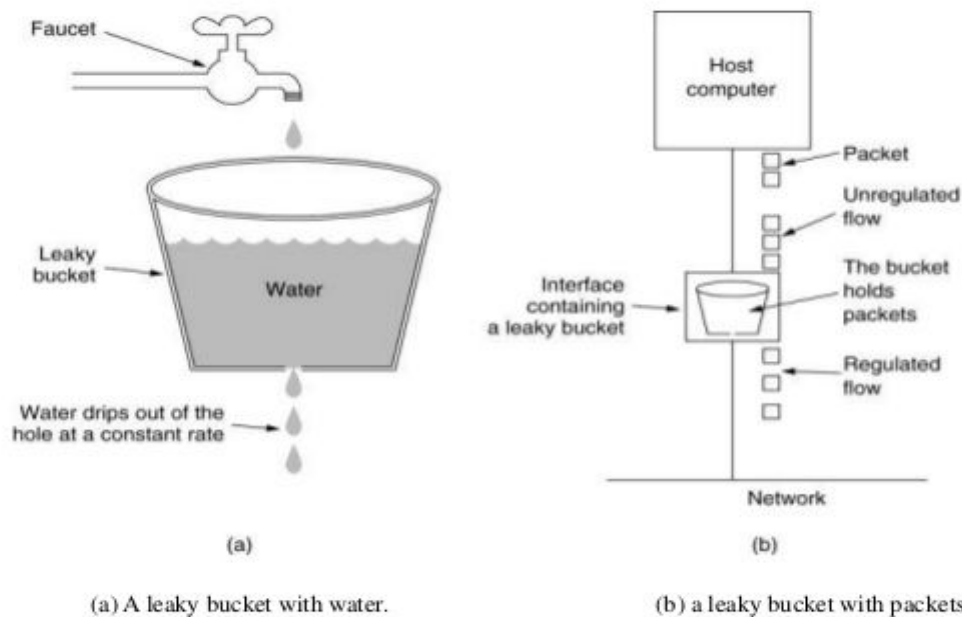
Cipher Text =
      89      45      21      18      39      129      62      49      98      78      62
129      37      45      49      68      80      98      54      59      32      45      49
59      129      45      49      121

Plain Text =
Computer Networks Laboratory
lab3-20@lab320-Veriton-Series:~/CN$
```

12. Write a program for congestion control using leaky bucket algorithm.

The main concept of the leaky bucket algorithm is that the output data flow remains constant despite the variant input traffic, such as the water flow in a bucket with a small hole at the bottom. In case the bucket contains water (or packets) then the output flow follows a constant rate, while if the bucket is full any additional load will be lost because of spill over. In a similar way if the bucket is empty the output will be zero. From network perspective, leaky bucket consists of a finite queue (bucket) where all the incoming packets are stored in case there is space in the queue, otherwise the packets are discarded. In order to regulate the output flow, leaky bucket transmits one packet from the queue in a fixed time (e.g. at every clock tick). In the following figure we can notice the main rationale of leaky bucket algorithm, for both the two approaches (e.g. leaky bucket with water (a) and with packets (b)).

The Leaky Bucket Algorithm



While leaky bucket eliminates completely bursty traffic by regulating the incoming data flow its main drawback is that it drops packets if the bucket is full. Also, it doesn't take into account the idle

process of the sender which means that if the host doesn't transmit data for some time the bucket becomes empty without permitting the transmission of any packet.

Source Code:

```
import java.io.*;
import java.util.Scanner;
class Leaky
{
    public static int min(int x,int y)
    {
        if(x<y)
            return x;
        else
            return y;
    }
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int drop=0,mini,n,cap,count=0,i;
        int inp[ ]= new int[25];
        int process;
        System.out.println("Enter The Bucket Size\n");
        cap=sc.nextInt();
        System.out.println("Enter The Output Rate\n");
        process=sc.nextInt();
        System.out.println("Enter the number of packets\n");
        n=sc.nextInt();
        System.out.println("Enter the size of packets to be sent:");
        for(i=0;i<n;i++)
        {
            inp[i]=sc.nextInt();
        }
        System.out.println("\nSecond|Packet Recieved|Packet Sent|PacketLeft|Packet
Dropped|\n");
        System.out.println("-----\n");
        for(i=0;i<n;i++)
        {
            count+=inp[i];
            if(count>cap)
            {
                drop=count-cap;
                count=cap;
            }
            System.out.print(i+1);
            System.out.print("\t" +inp[i]);
            mini=min(count,process);
            System.out.print("\t\t" + mini);
```

```
        count=count-mini;
        System.out.print("\t\t" +count);
        System.out.println("\t\t"+ drop);
        drop=0;
    }
    for(;count!=0;i++)
    {
        if(count>cap)
        {
            drop=count-cap;
            count=cap;
        }
        System.out.print(i+1);
        System.out.print("\t0");
        mini=min(count,process);
        System.out.print("\t\t" +mini);
        count=count-mini;
        System.out.print("\t\t" +count);
        System.out.println("\t\t" +drop);
    }
}
```

OUTPUT:


```
lab3-20@lab320-Veriton-Series: ~/CN
lab3-20@lab320-Veriton-Series:~/CN$ javac Leaky.java
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
lab3-20@lab320-Veriton-Series:~/CN$ java Leaky
Picked up JAVA_TOOL_OPTIONS: -javaagent:/usr/share/java/jayatanaag.jar
Enter The Bucket Size
5
Enter The Output Rate
2
Enter the number of packets
3
Enter the size of packets to be sent:
5
4
3

Second|Packet Recieved|Packet Sent|PacketLeft|Packet Dropped|
-----
1      5              2          3          0
2      4              2          3          2
3      3              2          3          1
4      0              2          1          0
5      0              1          0          0
lab3-20@lab320-Veriton-Series:~/CN$
```