Introduction

Computer Networks laboratory covers the implementation of basic networking concepts and simulation of advanced concepts. The prerequisite for this laboratory is the understanding of fundamentals of computer networks. There are two parts in this laboratory. The first part deals with simulation of networking concepts. The second part deals with how to implement the message transfer among the systems using inter process communication (IPC) techniques, security algorithms, routing algorithms, error detection and flow and congestion control techniques.

Network Simulation Using NS2:
Network simulation is an important tool in developing, testing and evaluating network protocols. Simulation can be used without the target physical hardware, making it economical and practical for almost any scale of network topology and setup. It is possible to simulate a link of any bandwidth and delay, even if such a link is currently impossible in the real world. With simulation, it is possible to set each simulated node to use any desired software. Most network simulators use abstractions of network protocols, rather than the real thing, making their results less convincing. S.Y. Wang reports that the simulator OPNET uses a simplified finite state machine to model complex TCP protocol processing. NS-2 uses a model based on TCP, it is implemented as a set of classes using inheritance.

Wang states that “Simulation results are not as convincing as those produced by real hardware and software equipment.” This statement is followed by an explanation of the fact that most existing network simulators can only simulate real life network protocol implementations with limited details, which can lead to incorrect results. Another paper includes a similar statement, “running the actual TCP code is preferred to running an abstract specification of the protocol.” Brakmo and Peterson go on to discuss how the BSD implementations of TCP are quite important with respect to timers. Simulators often use more accurate round trip time measurements than those used in the BSD implementation, making results differ.

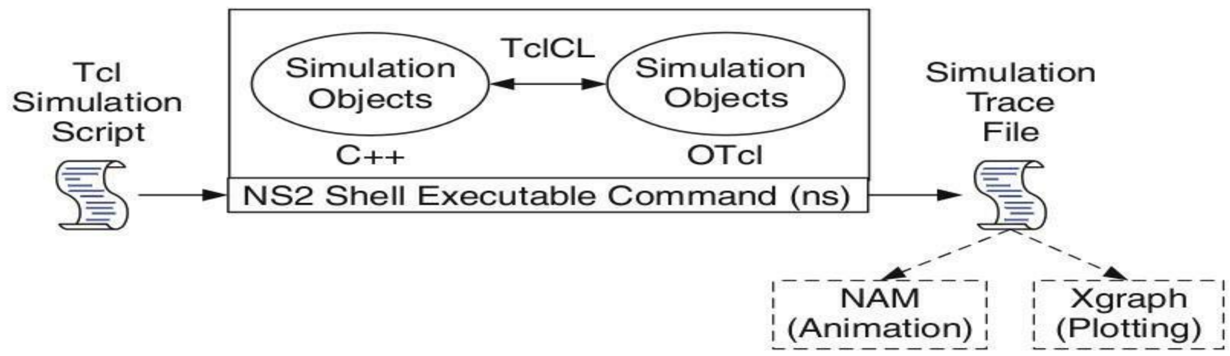
Starting ns

To start ns with the command 'ns <tclscript>' (assuming that current path is in the directory with the ns executable) should be used, where '<tclscript>' is the name of a Tcl (Tool Command Language) script file which defines the simulation scenario (i.e. the topology and the events). We can also start ns without any arguments and enter the Tcl commands in the Tcl shell, but that is definitely less comfortable**.**

Starting nam (Network Animator)

To start Network Animator, either start nam with the command 'nam <nam-file>'can be used where '<nam-file>' is the name of a nam trace file that was generated by ns, or can execute it directly out of the Tcl simulation script for the simulation which should be visualized.

Basic Architecture of NS2



<u>Tcl scripting</u>

- 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.

<u>Basics of TCL</u>

Syntax: command arg1 arg2 arg3

- Hello World!

  Puts stdout{Hello,World!}

  Hello, World

- Variables

  Command Substitution

set a 5 set len [string length foobar]

set b $a set len [expr [string length foobar] + 9]

- Simple Arithmetic

expr 7.2 / 4

- 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

- 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 data 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.

```
Proc finish { } {

global ns tracefile1 namfile

$ns flush-trace

Close $tracefile1

Close $namfile

Exec nam out.nam &

Exit 0
```

#Define a 'finish' procedure

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

```
$ns at 125.0 "finish"
```

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, 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]
```

We created a node that 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

```
#set Queue Size of link (n0-n2) to 20

$ns queue-limit $n0 $n2 20
```

be 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.

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

The command defines the behaviour 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

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

Above 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.

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 packet Size_ 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

```
$cbr set interval_ 0.005
```

time interval between transmission of packets using the command.

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”
```

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

12. Implementations in a real network do not use sequence number, ns keeps track of UDP packet sequence number for analysis purposes

13. The last field shows the unique id of the packet.

<u>XGRAPH</u>

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 there choice. You can now print a serial number, using the variable kount, and apply it those directors drawing a salary exceeding 6700:

```
$ awk –F"|" '$3 == "director" && $6 > 6700
{count =count+1

Printf "%3f %20s %-12s %d\n", count, $2, $3, $6}' empn.lst
```

THE –f OPTION: STORING awk PROGRAMS INA FILE

You should holds large awk programs in separate file 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

```
Awk –F"|" –f empawk.awk empn.lst
```

use awk with the –f *filename* option to obtain the same output:

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 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 variable. 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 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
```

The FILENAME Variable

FILENAME stores the name of the current file being processed. Like grep and sed, awk can also handle multiple filenames in the command line. By default, awk doesn't print the filename, but you can instruct it to do so:

```
'$6<4000 {print FILENAME, $0 }'
```

With FILENAME, you can device logic that does different things depending on the file that is processed.

NS2 Installation

- NS2 is a free simulation tool.
- It runs on various platforms including UNIX (or Linux), Windows, and Mac systems.
- NS2 source codes are distributed in two forms: the all-in-one suite and the component-wise.
- ‘all-in-one’ package provides an “install” script which configures the NS2 environment and creates NS2 executable file using the “make” utility.

<u>NS-2 installation steps in Linux</u>

➢ Go to Computer → File System → now paste the zip file “ns-allinone-2.34.tar.gz” into opt folder.

➢ Now unzip the file by typing the following command

```
[root@localhost  opt] # tar  -xzvf ns-allinone-2.34.tar.gz
```

➢ After the files get extracted, we get ns-allinone-2.34 folder as well as zip file ns-allinone-2.34.tar.gz

```
[root@localhost  opt] # ns-allinone-2.34           ns-allinone-2.34.tar.gz
```

➢ Now go to ns-allinone-2.33 folder and install it

```
[root@localhost   opt]   #   cd   ns-allinone-2.34
[root@localhost  ns-allinone-2.33] # ./install
```

➢ Once the installation is completed successfully we get certain pathnames in that terminal which must be pasted in “.bash_profile” file.

➢ First minimize the terminal where installation is done and open a new terminal and open the file “.bash_profile”

```
[root@localhost  ~] #  vi .bash_profile
```

➢ When we open this file, we get a line in that file which is shown below

```
PATH=$PATH:$HOME/bin
```

To this line we must paste the path which is present in the previous terminal where ns was installed. First put “:” then paste the path in-front of bin. That path is shown below. “:/opt/ns-allinone-2.33/bin:/opt/ns-allinone-2.33/tcl8.4.18/unix:/opt/ns-allinone-2.33/tk8.4.18/unix”.

In the next line type “LD_LIBRARY_PATH=$LD_LIBRARY_PATH:” and paste the two paths separated by “:” which are present in the previous terminal i.e Important notices section (1) “/opt/ns-allinone-2.33/otcl-1.13:/opt/ns-allinone-2.33/lib”

In the next line type “TCL_LIBRARY=$TCL_LIBRARY:” and paste the path which is present in previous terminal i.e Important Notices section (2)

“/opt/ns-allinone-2.33/tcl8.4.18/library”

➢ In the next line type “export LD_LIBRARY_PATH”

➢ In the next line type “export TCL_LIBRARY”

➢ The next two lines are already present the file “export PATH” and “unsetUSERNAME”

➢ Save the program (ESC + shift : wq and press enter)

➢ Now in the terminal where we have opened .bash_profile file, type the following command to check if path is updated correctly or not

```
[root@localhost  ~] #  vi .bash_profile

[root@localhost  ~] #  source .bash_profile
```

➢ If path is updated properly, then we will get the prompt as shown below

```
[root@localhost  ~] #
```

➢ Now open the previous terminal where you have installed ns

```
[root@localhost  ns-allinone-2.33] #
```

➢ Here we need to configure three packages “ns-2.33”, “nam-1.13” and “xgraph-12.1”

➢ First, configure “ns-2.33” package as shown below

```
[root@localhost    ns-allinone-2.33]    #    cd    ns-2.33
[root@localhost ns-2.33] # ./configure [root@localhost
ns-2.33]  #  make  clean  [root@localhost  ns-2.33]  #
make    [root@localhost    ns-2.33]    #    make    install
[root@localhost  ns-2.33] # ns

%
```

➢ If we get “%” symbol it indicates that ns-2.33 configuration was successful.

➢ Second, configure “nam-1.13” package as shown below

```
[root@localhost  ns-2.33] #  cd  . .
[root@localhost ns-allinone-2.33] # cd nam-1.13
[root@localhost nam-1.13] # ./configure
 [root@localhost nam-1.13] # make clean
[root@localhost nam-1.13] # make
```

```
[root@localhost nam-1.13] # make install
[root@localhost  nam-1.13] # ns
%
```

If we get “%” symbol it indicates that nam-1.13 configuration was successful.

- ➢ Third, configure “xgraph-12.1” package as shown below
- ➢ [root@localhost nam-1.13] # cd . .

```
[root@localhost ns-allinone-2.33] # cd xgraph-12.1

[root@localhost xgraph-12.1] # ./configure

[root@localhost xgraph-12.1] # make clean

[root@localhost xgraph-12.1] # make

 [root@localhost xgraph-12.1] # make install

 [root@localhost  xgraph-12.1] # ns

%
```

This completes the installation process of “NS-2” simulator

Part-A

Lab Program-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.

```
set ns [new Simulator]
set nf [open PA1.nam w]
$ns namtrace-all $nf
set tf [open PA1.tr w]
$ns trace-all $tf

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam PA1.nam &
exit 0
}
set n0 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

$ns duplex-link $n0 $n2 200Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 1000ms DropTail
$ns queue-limit $n0 $n2 10

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
```

```
$cbr0 set interval_ 0.005

$cbr0 attach-agent $udp0


set null0 [new Agent/Null]

$ns attach-agent $n3 $null0

$ns connect $udp0 $null0


$ns at 0.1 "$cbr0 start"

$ns at 1.0 "finish"

$ns run
```

<u>AWK file</u> (Open a new editor using "gedit command" and write awk file and save with ".awk" extension)

/*immediately after BEGIN should open braces '{'

```
BEGIN {
c=0;
}
{
If ($1=="d")
{
c++;
printf("%s\t%s\n",$5,$11);
}
}
END{
printf("The number of packets dropped =%d\n",c);
}
```

<u>Steps for execution</u>

- Open gedit editor and type program. Program name should have the extension " .tcl "
- gedit lab1.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" Lab Program-1and press Enter key.
- Open gedit editor and type awk program. Program name should have the extension ".awk "
- gedit lab1.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

- Run the simulation program
- ns lab1.tcl

- Here "ns" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begins.
- After simulation is completed run awk file to see the output ,
- awk –f lab1.awk lab1.tr
- To see the trace file contents open the file as ,
- gedit lab1.tr

<u>Trace file contains 12 columns:-</u>

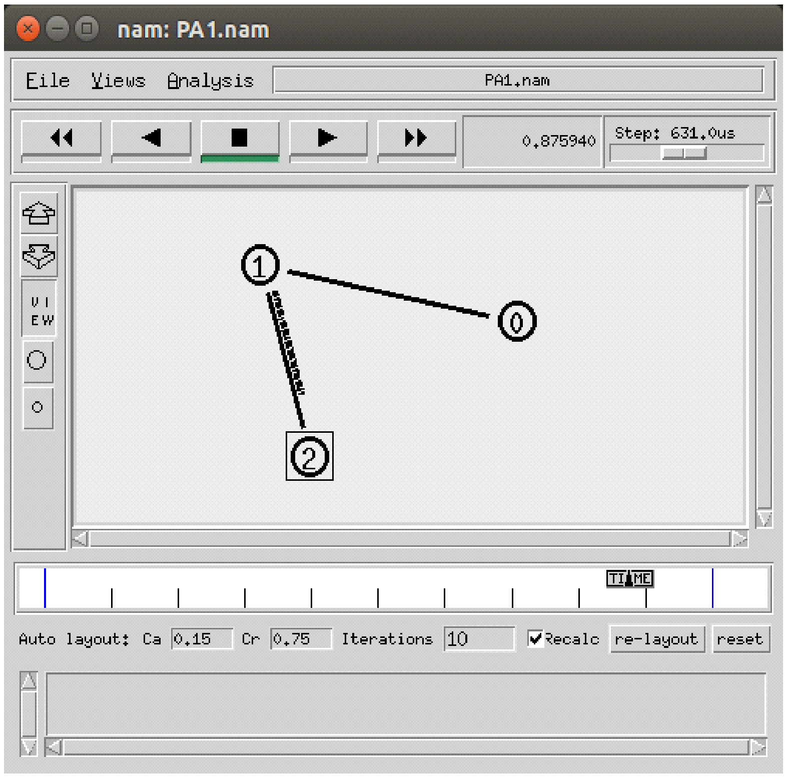
Event type, Event time, From Node, Source Node, Packet Type, Packet Size, Flags (indicated by --------), Flow ID, Source address, Destination address, Sequence ID, Packet ID

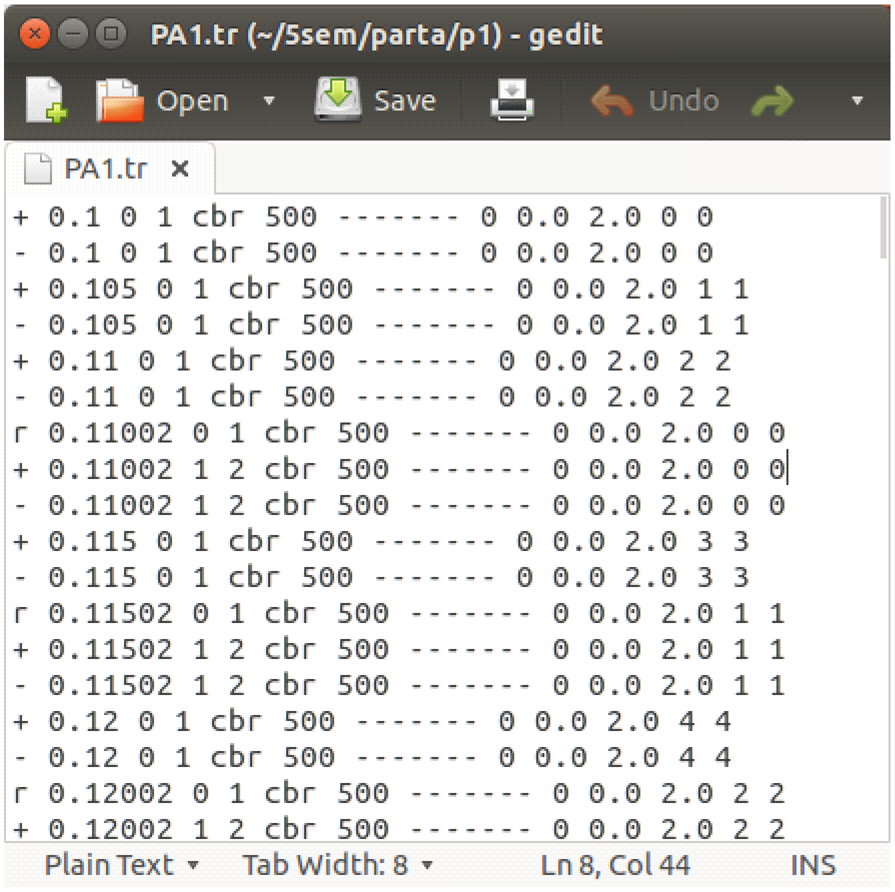
<u>Output</u>

ns lab1.tcl

awk –f lab1.awk PA1.tr

The number of packets dropped =0





```
+ 0.1 0 1 cbr 500 ------- 0 0.0 2.0 0 0
- 0.1 0 1 cbr 500 ------- 0 0.0 2.0 0 0
+ 0.105 0 1 cbr 500 ------- 0 0.0 2.0 1 1
- 0.105 0 1 cbr 500 ------- 0 0.0 2.0 1 1
+ 0.11 0 1 cbr 500 ------- 0 0.0 2.0 2 2
- 0.11 0 1 cbr 500 ------- 0 0.0 2.0 2 2
r 0.11002 0 1 cbr 500 ------- 0 0.0 2.0 0 0
+ 0.11002 1 2 cbr 500 ------- 0 0.0 2.0 0 0
- 0.11002 1 2 cbr 500 ------- 0 0.0 2.0 0 0
+ 0.115 0 1 cbr 500 ------- 0 0.0 2.0 3 3
- 0.115 0 1 cbr 500 ------- 0 0.0 2.0 3 3
r 0.11502 0 1 cbr 500 ------- 0 0.0 2.0 1 1
+ 0.11502 1 2 cbr 500 ------- 0 0.0 2.0 1 1
- 0.11502 1 2 cbr 500 ------- 0 0.0 2.0 1 1
+ 0.12 0 1 cbr 500 ------- 0 0.0 2.0 4 4
- 0.12 0 1 cbr 500 ------- 0 0.0 2.0 4 4
r 0.12002 0 1 cbr 500 ------- 0 0.0 2.0 2 2
+ 0.12002 1 2 cbr 500 ------- 0 0.0 2.0 2 2
```

Note:

- Set the queue size fixed from n0 to n2 as 10, n1-n2 to n3 as 5. Syntax: To set the queue size
- 10 and from n2- $ns set queue-limit <from> <to> <size> Eg:
- $ns set queue-limit $n0 $n2 10
- Go on varying the bandwidth from 10, 20 30 . . and find the number of packets
- dropped at the node 2

Lab Program-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.

```
set ns [ new Simulator ]
set nf [ open lab2.nam w ]
$ns namtrace-all $nf
set tf [ open lab2.tr w ]
$ns trace-all $tf

set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n4 shape box

$ns duplex-link $n0 $n4 1005Mb 1ms DropTail
$ns duplex-link $n1 $n4 50Mb 1ms DropTail
$ns duplex-link $n2 $n4 2000Mb 1ms DropTail
$ns duplex-link $n3 $n4 200Mb 1ms DropTail
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set p1 [new Agent/Ping]
$ns attach-agent $n0 $p1
$p1 set packetSize_ 50000
$p1 set interval_ 0.0001

set p2 [new Agent/Ping]
$ns attach-agent $n1 $p2

set p3 [new Agent/Ping]
$ns attach-agent $n2 $p3
$p3 set packetSize_ 30000
$p3 set interval_ 0.00001

set p4 [new Agent/Ping]
$ns attach-agent $n3 $p4

set p5 [new Agent/Ping]
$ns attach-agent $n5 $p5

$ns queue-limit $n0 $n4 5
```

```
$ns queue-limit $n2 $n4 3

$ns queue-limit $n4 $n5 2
Agent/Ping instproc recv {from rtt} {
$self instvar node_
puts "node [$node_ id] received answer from $from with round trip time $rtt msec"
}
# please provide space between $node_ and id. No space between $ and from. No
#space between and $ and rtt */
$ns connect $p1 $p5
$ns connect $p3 $p4

proc finish { } {
global ns nf tf
$ns flush-trace
close $nf
close $tf
exec nam lab2.nam &
exit 0
}
$ns at 0.1 "$p1 send"
$ns at 0.2 "$p1 send"
$ns at 0.3 "$p1 send"
$ns at 0.4 "$p1 send"
$ns at 0.5 "$p1 send"
$ns at 0.6 "$p1 send"
$ns at 0.7 "$p1 send"
$ns at 0.8 "$p1 send"
$ns at 0.9 "$p1 send"
$ns at 1.0 "$p1 send"
$ns at 1.1 "$p1 send"
$ns at 1.2 "$p1 send"
$ns at 1.3 "$p1 send"
$ns at 1.4 "$p1 send"
$ns at 1.5 "$p1 send"
$ns at 1.6 "$p1 send"
$ns at 1.7 "$p1 send"
$ns at 1.8 "$p1 send"
$ns at 1.9 "$p1 send"
$ns at 2.0 "$p1 send"
$ns at 2.1 "$p1 send"
$ns at 2.2 "$p1 send"
$ns at 2.3 "$p1 send"
$ns at 2.4 "$p1 send"
$ns at 2.5 "$p1 send"
$ns at 2.6 "$p1 send"
$ns at 2.7 "$p1 send"
```

```
$ns at 2.8 "$p1 send"
$ns at 2.9 "$p1 send"
$ns at 0.1 "$p3 send"
$ns at 0.2 "$p3 send"
$ns at 0.3 "$p3 send"
$ns at 0.4 "$p3 send"
$ns at 0.5 "$p3 send"
$ns at 0.6 "$p3 send"
$ns at 0.7 "$p3 send"
$ns at 0.8 "$p3 send"
$ns at 0.9 "$p3 send"
$ns at 1.0 "$p3 send"
$ns at 1.1 "$p3 send"
$ns at 1.2 "$p3 send"
$ns at 1.3 "$p3 send"
$ns at 1.4 "$p3 send"
$ns at 1.5 "$p3 send"
$ns at 1.6 "$p3 send"
$ns at 1.7 "$p3 send"
$ns at 1.8 "$p3 send"
$ns at 1.9 "$p3 send"
$ns at 2.0 "$p3 send"
$ns at 2.1 "$p3 send"
$ns at 2.2 "$p3 send"
$ns at 2.3 "$p3 send"
$ns at 2.4 "$p3 send"
$ns at 2.5 "$p3 send"
$ns at 2.6 "$p3 send"
$ns at 2.7 "$p3 send"
$ns at 2.8 "$p3 send"
$ns at 2.9 "$p3 send"
$ns at 3.0 "finish"
$ns run
```

AWK file (Open a new editor using "gedit command" and write awk file and save with ".awk" extension)

```
BEGIN {
drop=0;
}
{
 if($1=="d" )
 {
  drop++;
  }
  }
 END{
```

```
printf("Total number of %s packets dropped due to congestion =%d\n",$5,drop);
}
```

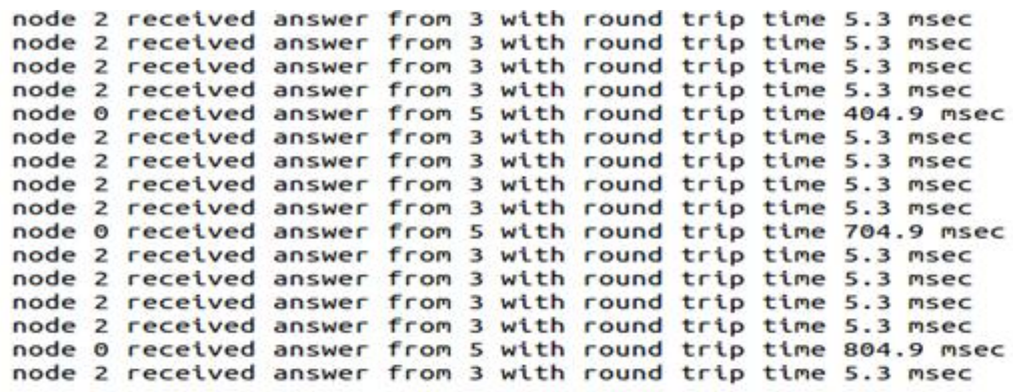
Steps For Execution

1. Open gedit editor and type program. Program name should have the extension " .tcl "
gedit lab2.tcl
2. Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" Lab Program-1and press Enter key.
3. Open gedit editor and type awk program. Program name should have the extension ".awk "
gedit lab2.awk
4. Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

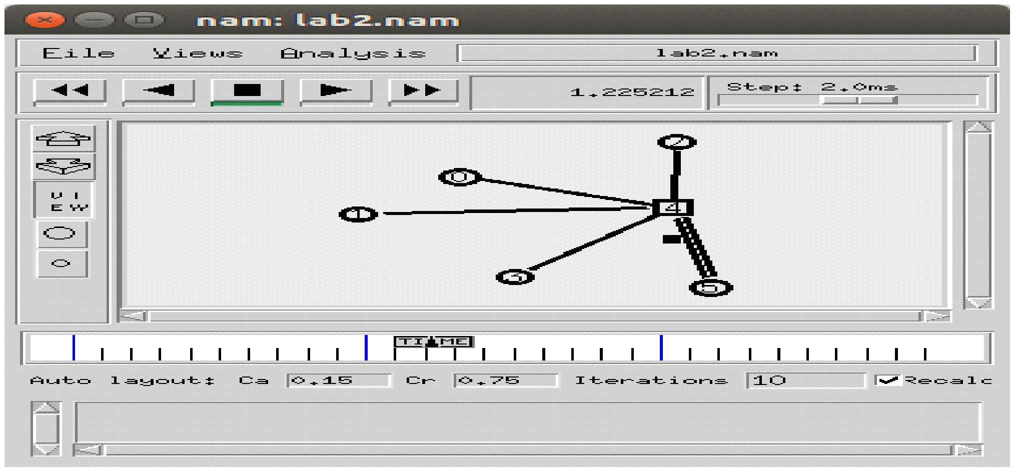
5. Run the simulation program
ns lab2.tcl
Here "ns" indicates network simulator. We get the topology shown in the snapshot.
Now press the play button in the simulation window and the simulation will begins.
6. After simulation is completed run awk file to see the output ,
awk –f lab2.awk lab2.tr
7. To see the trace file contents open the file as ,
gedit lab2.tr

Output:

ns lab2.tcl



```
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 0 received answer from 5 with round trip time 404.9 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 0 received answer from 5 with round trip time 704.9 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 2 received answer from 3 with round trip time 5.3 msec
node 0 received answer from 5 with round trip time 804.9 msec
node 2 received answer from 3 with round trip time 5.3 msec
```



awk –f lab2.awk lab2.tr
Total number of ping packets dropped due to congestion =0

Note:

Vary the bandwidth and queue size between the nodes n0-n2 , n2-n4. n6-n2 and n2- n5 and see the number of packets dropped at the nodes.

Lab Program-3

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

```
set ns [new Simulator]
set tf [open lab3.tr w]
$ns trace-all $tf
set nf [open lab3.nam w]
$ns namtrace-all $nf

set n0 [$ns node]
$n0 color "magenta"
$n0 label "src1"

set n1 [$ns node]

set n2 [$ns node]
$n2 color "magenta"
$n2 label "src2"

set n3 [$ns node]
$n3 color "blue"
$n3 label "dest2"

set n4 [$ns node]

set n5 [$ns node]
$n5 color "blue"
$n5 label "dest1"
```

```
$ns make-lan "$n0 $n1 $n2 $n3 $n4" 100Mb 100ms LL Queue/DropTail Mac/802_3
$ns duplex-link $n4 $n5 1Mb 1ms DropTail

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0

set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$ftp0 set packetSize_ 500
$ftp0 set interval_ 0.0001

set sink5 [new Agent/TCPSink]
$ns attach-agent $n5 $sink5
$ns connect $tcp0 $sink5

set tcp2 [new Agent/TCP]
$ns attach-agent $n2 $tcp2

set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set packetSize_ 600
$ftp2 set interval_ 0.001

set sink3 [new Agent/TCPSink]
$ns attach-agent $n3 $sink3
$ns connect $tcp2 $sink3

set file1 [open file1.tr w]
$tcp0 attach $file1

set file2 [open file2.tr w]
$tcp2 attach $file2
```

```
$tcp0 trace cwnd_
$tcp2 trace cwnd_

proc finish { } {
global ns nf tf
$ns flush-trace
close $tf
close $nf
exec nam lab3.nam &
exit 0
}

$ns at 0.1 "$ftp0 start"
$ns at 5 "$ftp0 stop"
$ns at 7 "$ftp0 start"
$ns at 0.2 "$ftp2 start"
$ns at 8 "$ftp2 stop"
$ns at 14 "$ftp0 stop"
$ns at 10 "$ftp2 start"
$ns at 15 "$ftp2 stop"
$ns at 16 "finish"
$ns run
```

AWK file (Open a new editor using "gedit command" and write awk file and save with ".awk" extension)

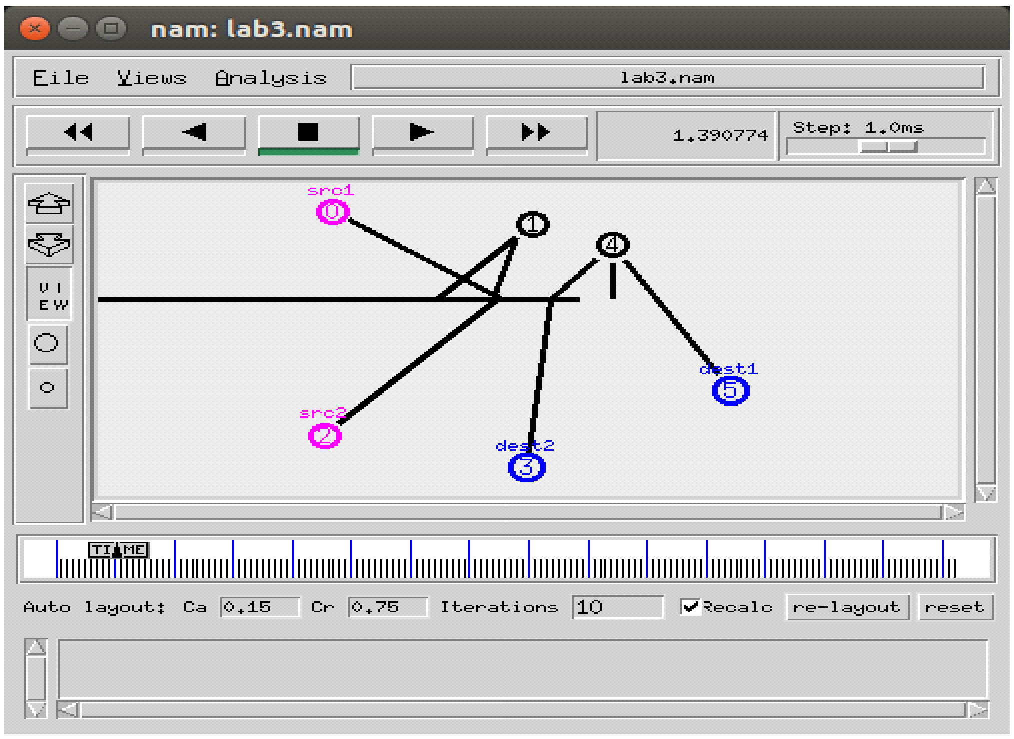
```
BEGIN {
}
{
if($6=="cwnd_")
}
END {
}
```

Steps for execution

- Open gedit editor and type program. Program name should have the extension ".tcl "
  gedit lab3.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" Lab Program-1and press Enter key.
- Open gedit editor and type awk program. Program name should have the extension ".awk "
  gedit lab3.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

- Run the simulation program
  ns lab3.tcl
  Here "ns" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begins.
- After simulation is completed run awk file to see the output ,
  awk –f lab3.awk lab3.tr
- To see the trace file contents open the file as ,
  gedit lab3.tr

Output:



Lab Program-4

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

```
set ns [new Simulator]

#setup trace support by opening file lab4.tr and call the procedure trace-all
set tf [open lab4.tr w]
$ns trace-all $tf

#create a topology object that keeps track of movements of mobile nodes within the topological
boundary.
set topo [new Topography]
$topo load_flatgrid 1000 1000

set nf [open lab4.nam w]
$ns namtrace-all-wireless $nf 1000 1000

# creating a wireless node you MUST first select (configure) the node configuration parameters to
"become" a wireless node.
$ns node-config -adhocRouting DSDV \
-llType LL \
-macType Mac/802_11 \
-ifqType Queue/DropTail \
-ifqLen 50 \
-phyType Phy/WirelessPhy \
-channelType Channel/WirelessChannel \
-propType Propagation/TwoRayGround \
-antType Antenna/OmniAntenna \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON

# Create god object
create-god 3
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]

$n0 label "tcp0"
$n1 label "sink1/tcp1"
$n2 label "sink2"

$n0 set X_ 50
$n0 set Y_ 50
$n0 set Z_ 0
$n1 set X_ 100
```

```
$n1 set Y_ 100
$n1 set Z_ 0
$n2 set X_ 600
$n2 set Y_ 600
$n2 set Z_ 0

$ns at 0.1 "$n0 setdest 50 50 15"
$ns at 0.1 "$n1 setdest 100 100 25"
$ns at 0.1 "$n2 setdest 600 600 25"

set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
set sink1 [new Agent/TCPSink]
$ns attach-agent $n1 $sink1
$ns connect $tcp0 $sink1
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
set sink2 [new Agent/TCPSink]
$ns attach-agent $n2 $sink2
$ns connect $tcp1 $sink2

$ns at 5 "$ftp0 start"
$ns at 5 "$ftp1 start"

$ns at 100 "$n1 setdest 550 550 15"
$ns at 190 "$n1 setdest 70 70 15"

proc finish { } {
global ns nf tf
$ns flush-trace
exec nam lab4.nam &
close $tf
exit 0
}
$ns at 250 "finish"
$ns run
```

<u>AWK file</u> (Open a new editor using "gedit command" and write awk file and save with ".awk" extension)

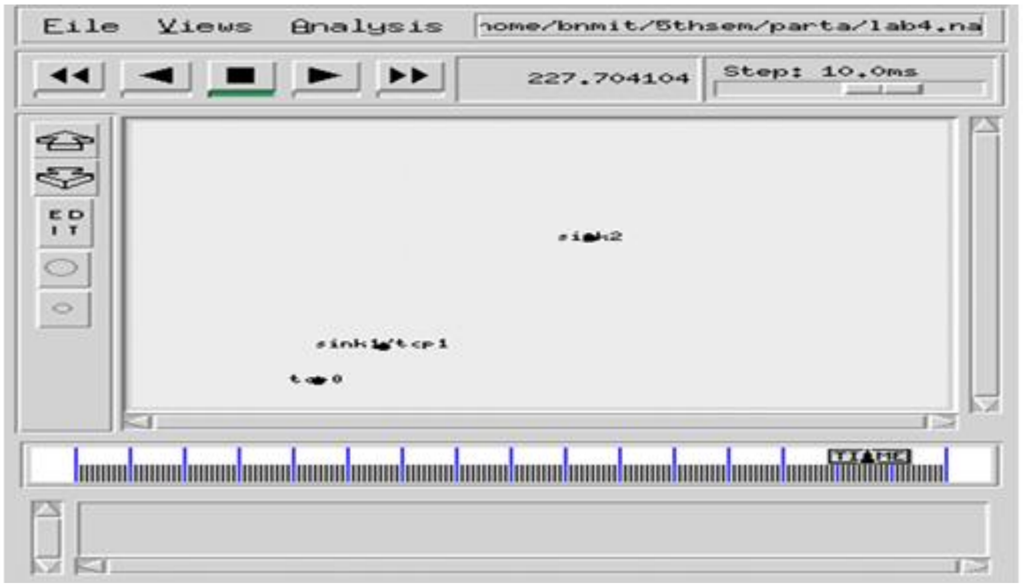
```
BEGIN {
count1=0
count2=0
pack1=0
pack2=0
time1=0
time2=0
}
{
if($1=="r"&& $3=="_1_" && $4=="AGT")
{
count1++
pack1=pack1+$8

time1=$2
}
if($1=="r" && $3=="_2_" && $4=="AGT")
{
count2++
pack2=pack2+$8
time2=$2
}
}
END {
printf("The Throughput from  n0 to  n1 : %f Mbps\n",
((count1*pack1*8)/(time1*1000000)));
printf("The Throughput from n1 to n2: %f Mbps\n", ((count2*pack2*8)/(time2*1000000)));
}
```

Steps for execution:

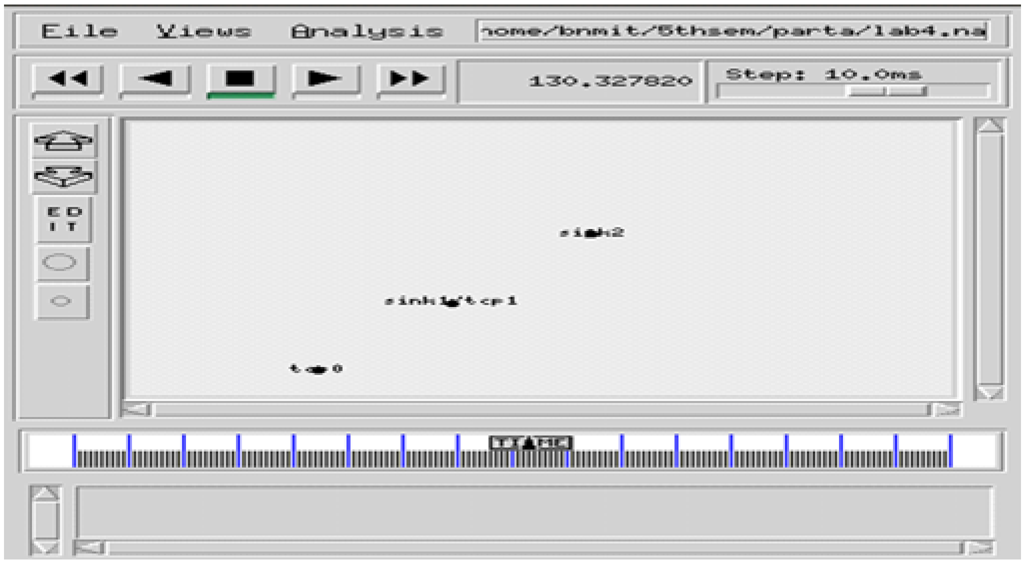
- Open gedit editor and type program. Program name should have the extension ".tcl "
  gedit lab4.tcl
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" Lab Program-1and press Enter key.
- Open gedit editor and type awk program. Program name should have the extension ".awk "
  gedit lab4.awk
- Save the program by pressing "ESC key" first, followed by "Shift and :" keys simultaneously and type "wq" and press Enter key.

- Run the simulation program
  ns lab4.tcl
  Here "ns" indicates network simulator. We get the topology shown in the snapshot.
- Now press the play button in the simulation window and the simulation will begins.
- After simulation is completed run awk file to see the output ,
  awk –f lab4.awk lab4.tr
- To see the trace file contents open the file as ,
  gedit lab4.tr

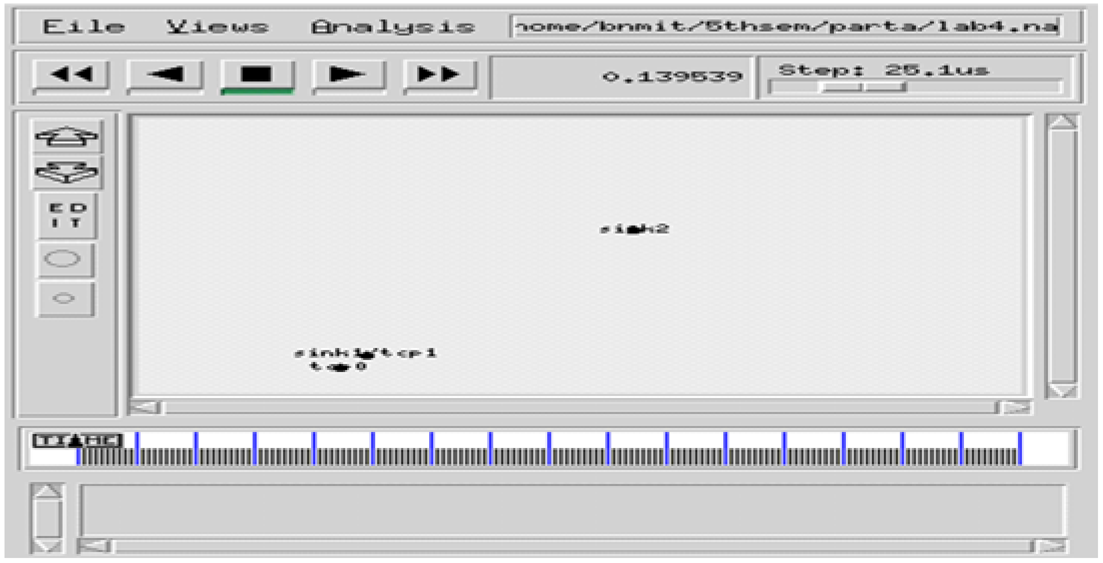
Output:



Node 1 and 2 are communicating



Node 2 and 3 are communicating



Node 2 and 1 are communicating

Lab Program-5

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

Second Generation (2G) technology is based on the technology known as global system for mobile communication (GSM). This technology enabled various networks to provide services like text messages, picture messages and MMS. The technologies used in 2G are either TDMA (Time Division Multiple Access) which divides signal into different time slots or CDMA (Code Division Multiple Access) which allocates a special code to each user so as to communicate over a multiplex physical channel.

GSM uses a variation of time division multiple access (TDMA). 2G networks developed as a replacement for first generation (1G) analog cellular networks, and the GSM standard originally described as a digital, circuit-switched network optimized for full duplex voice telephony. This expanded over time to include data communications, first by circuit-switched transport, then by packet data transport via GPRS (General Packet Radio Services).

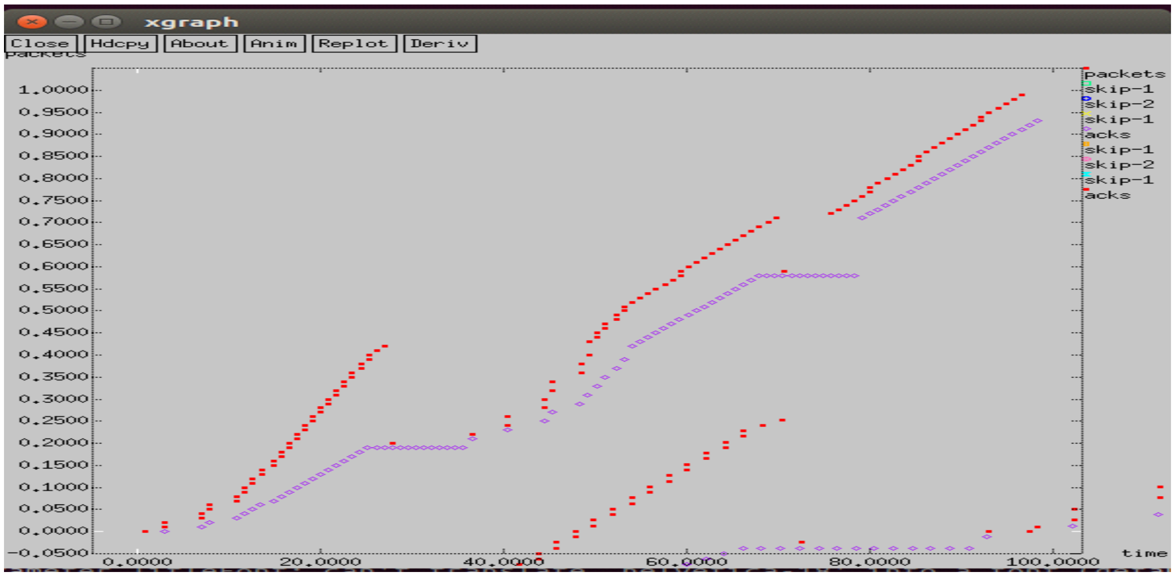
GSM can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2)

```
# General Parameters
set stop 100     ;# Stop time.
# Topology
set type gsm     ;#type of link:
# AQM parameters
set minth 0      ;
set maxth 30     ;
set adaptive 1   ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set window 30 ;# window for long-lived traffic
set web 2                ;# number of web sessions
# Plotting statics.
set opt(wrap)    100     ;# wrap plots?
set opt(srcTrace) is     ;# where to plot traffic
set opt(dstTrace) bs2    ;# where to plot traffic

#default downlink bandwidth in bps
set bwDL(gsm)  9600
#default uplink bandwidth in bps
set bwUL(gsm)  9600
#default downlink propagation delay in seconds
set propDL(gsm)  .500
#default uplink propagation delay in seconds
set propUL(gsm)  .500
set ns [new Simulator]
set tf [open out.tr w]
$ns trace-all $tf

set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]


proc cell_topo {} {
  global ns nodes
  $ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10nodes(ms) DropTail
  $ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
  $ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
  $ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50nodes(ms) DropTail
  puts " GSM Cell Topology"
}
proc set_link_para {t} {
  global ns nodes bwUL bwDL propUL propDL buf
  $ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
  $ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
  $ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
  $ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
  $ns queue-limit $nodes(bs1) $nodes(ms) 10
  $ns queue-limit $nodes(bs2) $nodes(ms) 10
}
# RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window

source web.tcl

#Create topology
switch $type {
 gsm -
 gprs -
 umts {cell_topo}
}

set_link_para $type

# Set up forward TCP connection
set tcp1 [new Agent/TCP]
$ns attach-agent $nodes(is) $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $nodes(lp) $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 0.8 "[set ftp1] start"


proc stop {} {
        global nodes opt nf
    set wrap $opt(wrap)
        set sid [$nodes($opt(srcTrace)) id]
        set did [$nodes($opt(dstTrace)) id]
        set a "out.tr"
        set GETRC "../../../bin/getrc"
    set RAW2XG "../../../bin/raw2xg"
    exec $GETRC -s $sid -d $did -f 0 out.tr | \
      $RAW2XG -s 0.01  -m $wrap -r > plot.xgr
    exec $GETRC -s $did -d $sid -f 0 out.tr | \
      $RAW2XG -a -s 0.01 -m $wrap  >> plot.xgr
    exec xgraph -x time -y packets plot.xgr &
        exit 0
}
$ns at $stop "stop"
$ns run
```

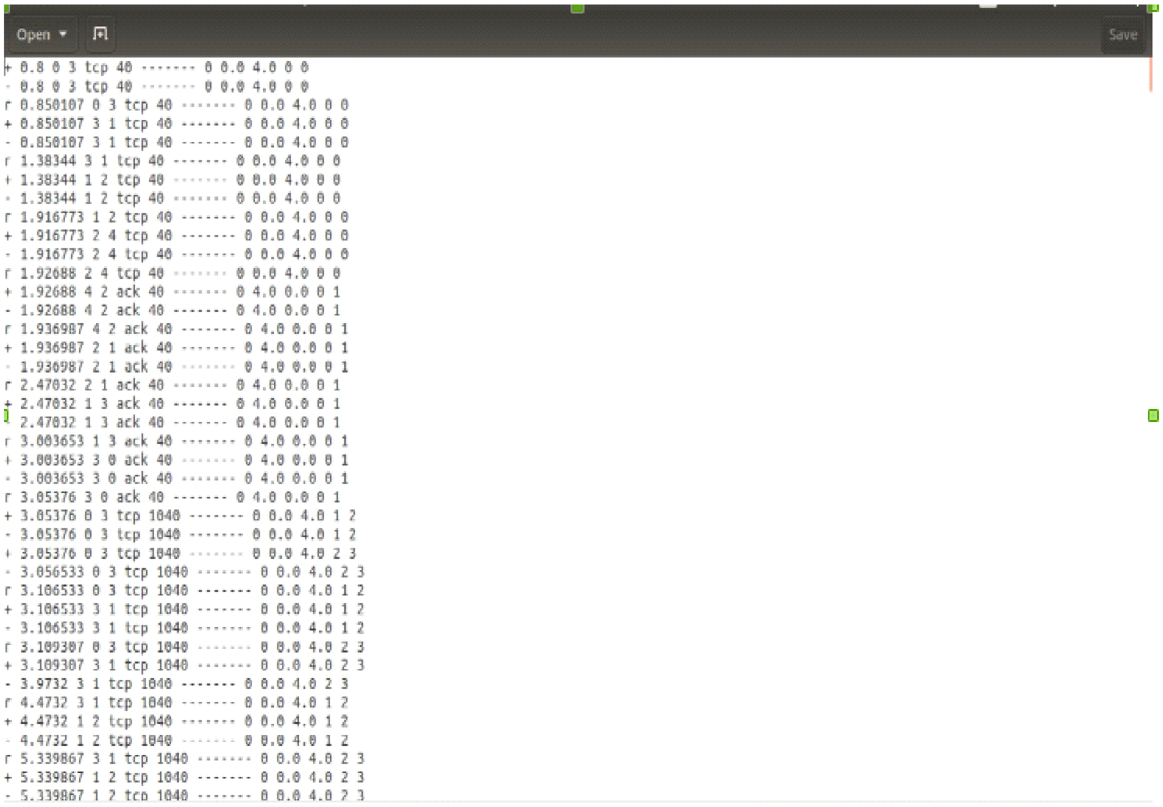
Steps for execution:

1. Open the following path in the terminal by switching to super user:
   - ➢ **Sudo su** (Enter the password)
   - ➢ Change directory to the path : **ns-allinone/ns-2.35/tcl/ex/wireless-scripts**
2. Create a file: **gedit GSM.tcl** and save in the path specified above
3. Run the program: **ns GSM.tcl**

Output:



Trace file contents



```
+ 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
- 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
r 0.850107 0 3 tcp 40 ------- 0 0.0 4.0 0 0
+ 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
- 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
r 1.38344 3 1 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.38344 1 2 tcp 40 ------- 0 0.0 4.0 0 0
- 1.38344 1 2 tcp 40 ------- 0 0.0 4.0 0 0
r 1.916773 1 2 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.916773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
- 1.916773 2 4 tcp 40 ------- 0 0.0 4.0 0 0
r 1.92688 2 4 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.92688 4 2 ack 40 ------- 0 4.0 0.0 0 1
- 1.92688 4 2 ack 40 ------- 0 4.0 0.0 0 1
r 1.936987 4 2 ack 40 ------- 0 4.0 0.0 0 1
+ 1.936987 2 1 ack 40 ------- 0 4.0 0.0 0 1
- 1.930987 2 1 ack 40 ------- 0 4.0 0.0 0 1
r 2.47032 2 1 ack 40 ------- 0 4.0 0.0 0 1
+ 2.47032 1 3 ack 40 ------- 0 4.0 0.0 0 1
- 2.47032 1 3 ack 40 ------- 0 4.0 0.0 0 1
r 3.003653 1 3 ack 40 ------- 0 4.0 0.0 0 1
+ 3.003653 3 0 ack 40 ------- 0 4.0 0.0 0 1
- 3.003653 3 0 ack 40 ------- 0 4.0 0.0 0 1
r 3.05376 3 0 ack 40 ------- 0 4.0 0.0 0 1
+ 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
- 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
+ 3.05376 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
- 3.056533 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
r 3.106533 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
+ 3.106533 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
- 3.106533 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
r 3.109307 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
+ 3.109307 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
- 3.9732 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
r 4.4732 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
+ 4.4732 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
- 4.4732 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
r 5.339867 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
+ 5.339867 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
- 5.339867 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
```

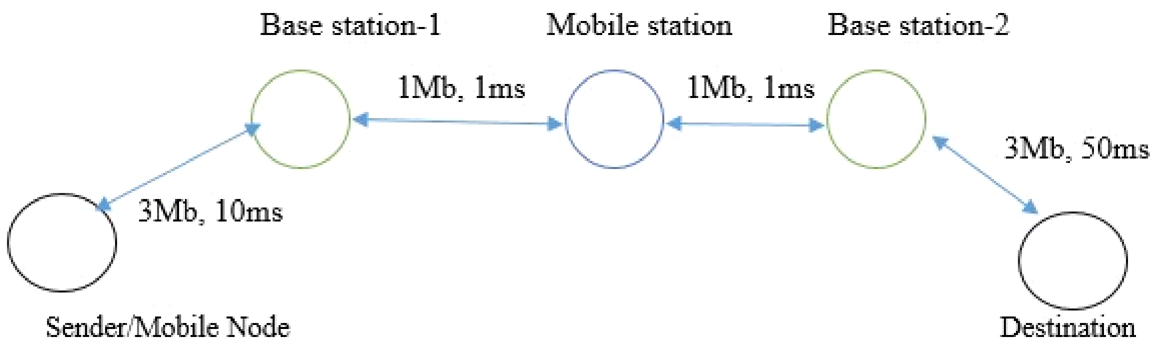
Lab Program-6

Implement and study the performance of CDMA on NS2/NS3 (Using stack called Call net) or equivalent environment.

3G networks developed as a replacement for second generation (2G) GSM standard network with full duplex voice telephony. CDMA is used as the access method in many mobile phone standards. IS-95, also called cdmaOne, and its 3G evolution CDMA2000, are often simply referred to as CDMA, but UMTS(The Universal Mobile Telecommunications System is a third generation mobile cellular system for networks based on the GSM standard.), the 3G standard used by GSM carriers, also uses wideband CDMA. Long-Term Evolution (LTE) is a standard for high-speed wireless communication which uses CDMA network technology.

3G technology generally refers to the standard of accessibility and speed of mobile devices. The standards of the technology were set by the International Telecommunication Union (ITU). This technology enables use of various services like GPS (Global Positioning System), mobile television and video conferencing. It not only enables them to be used worldwide, but also provides with better bandwidth and increased speed. The main aim of this technology is to allow much better coverage and growth with minimum investment.
CDMA can be implemented on all the versions of NS2 (Since year 2004: ns-2.27, and later versions of NS2) 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.

Design:



```
# General Parameters
set stop 100     ;# Stop time.
# Topology
set type cdma  ;#type of link:
# AQM parameters
set minth 0      ;
set maxth 30    ;
set adaptive 1  ;# 1 for Adaptive RED, 0 for plain RED
# Traffic generation.
set window 30 ;# window for long-lived traffic
set web 2                    ;# number of web sessions
# Plotting statics.
set opt(wrap)    100     ;# wrap plots?
set opt(srcTrace) is     ;# where to plot traffic
set opt(dstTrace) bs2   ;# where to plot traffic

#default downlink bandwidth in bps
set bwDL(cdma)  384000
#default uplink bandwidth in bps
set bwUL(cdma)  64000
#default downlink propagation delay in seconds
set propDL(cdma)  .150
#default uplink propagation delay in seconds
set propUL(cdma)  .150

set ns [new Simulator]
set tf [open out.tr w]
$ns trace-all $tf

set nodes(is) [$ns node]
set nodes(ms) [$ns node]
set nodes(bs1) [$ns node]
set nodes(bs2) [$ns node]
set nodes(lp) [$ns node]


proc cell_topo {} {
  global ns nodes
  $ns duplex-link $nodes(lp) $nodes(bs1) 3Mbps 10nodes(ms) DropTail
  $ns duplex-link $nodes(bs1) $nodes(ms) 1 1 RED
  $ns duplex-link $nodes(ms) $nodes(bs2) 1 1 RED
  $ns duplex-link $nodes(bs2) $nodes(is) 3Mbps 50nodes(ms) DropTail
  puts " CDMA Cell Topology"
}
proc set_link_para {t} {
  global ns nodes bwUL bwDL propUL propDL buf
  $ns bandwidth $nodes(bs1) $nodes(ms) $bwDL($t) duplex
  $ns bandwidth $nodes(bs2) $nodes(ms) $bwDL($t) duplex
  $ns delay $nodes(bs1) $nodes(ms) $propDL($t) duplex
  $ns delay $nodes(bs2) $nodes(ms) $propDL($t) duplex
  $ns queue-limit $nodes(bs1) $nodes(ms) 10
  $ns queue-limit $nodes(bs2) $nodes(ms) 10
}
# RED and TCP parameters
Queue/RED set adaptive_ $adaptive
Queue/RED set thresh_ $minth
Queue/RED set maxthresh_ $maxth
Agent/TCP set window_ $window

source web.tcl

#Create topology
switch $type {
 cdma {cell_topo}
}

set_link_para $type

# Set up forward TCP connection
set tcp1 [new Agent/TCP]
$ns attach-agent $nodes(is) $tcp1
set sink1 [new Agent/TCPSink]
$ns attach-agent $nodes(lp) $sink1
$ns connect $tcp1 $sink1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ns at 0.8 "[set ftp1] start"



proc stop {} {
        global nodes opt nf
    set wrap $opt(wrap)
        set sid [$nodes($opt(srcTrace)) id]
        set did [$nodes($opt(dstTrace)) id]
        set a "out.tr"
        set GETRC "../../../bin/getrc"
    set RAW2XG "../../../bin/raw2xg"
    exec $GETRC -s $sid -d $did -f 0 out.tr | \
      $RAW2XG -s 0.01  -m $wrap -r > plot.xgr
    exec $GETRC -s $did -d $sid -f 0 out.tr | \
      $RAW2XG -a -s 0.01 -m $wrap  >> plot.xgr
    exec xgraph -x time -y packets plot.xgr &
        exit 0
}
$ns at $stop "stop"
$ns run
```

<u>Steps for execution:</u>

1. Open the following path in the terminal by switching to super user:

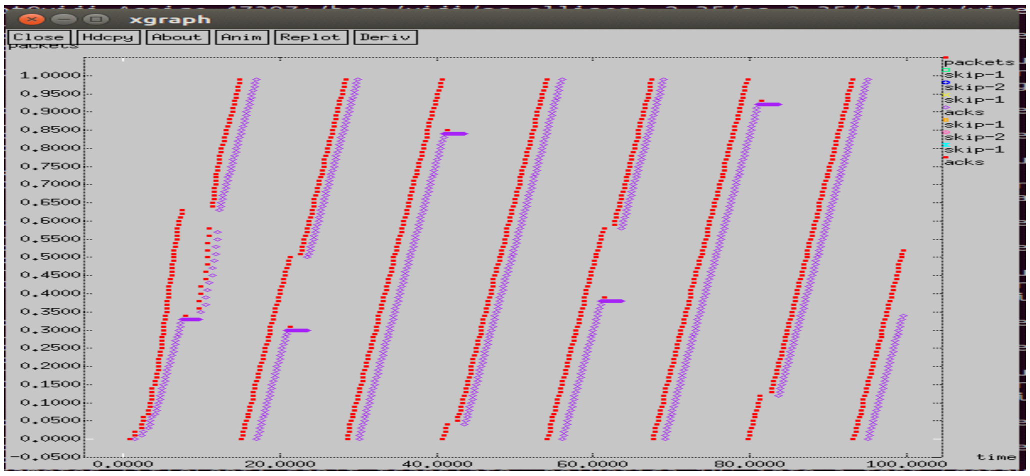
➢ **Sudo su** (Enter the password)

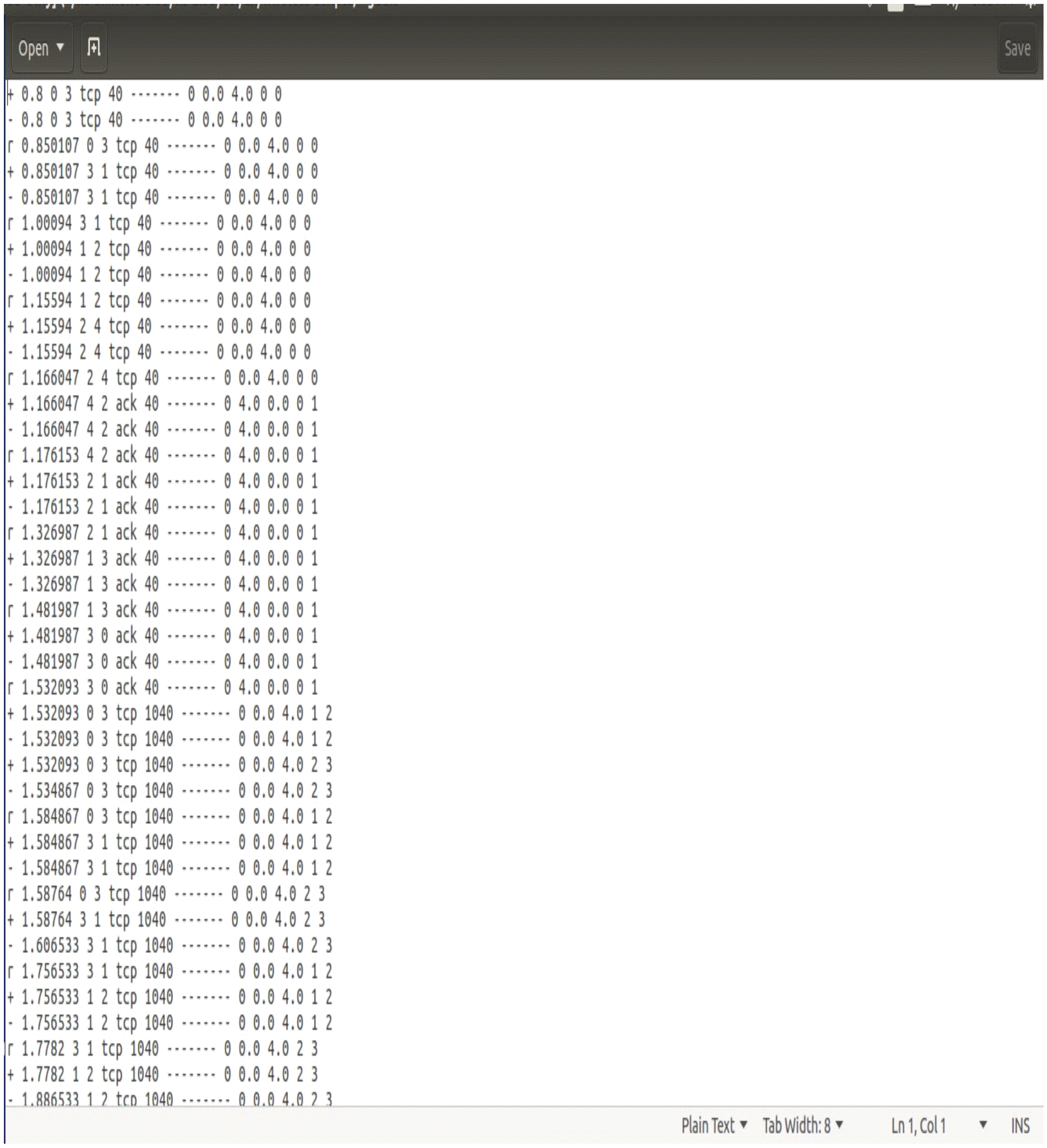
➢ Change directory to the path : **ns-allinone/ns-2.35/tcl/ex/wireless-scripts**

2. Create a file: **gedit CDMA.tcl** and save in the path specified above

3. Run the program: **ns CDMA.tcl**

<u>Output:</u>





```
+ 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
- 0.8 0 3 tcp 40 ------- 0 0.0 4.0 0 0
r 0.850107 0 3 tcp 40 ------- 0 0.0 4.0 0 0
+ 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
- 0.850107 3 1 tcp 40 ------- 0 0.0 4.0 0 0
r 1.00094 3 1 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.00094 1 2 tcp 40 ------- 0 0.0 4.0 0 0
- 1.00094 1 2 tcp 40 ------- 0 0.0 4.0 0 0
r 1.15594 1 2 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.15594 2 4 tcp 40 ------- 0 0.0 4.0 0 0
- 1.15594 2 4 tcp 40 ------- 0 0.0 4.0 0 0
r 1.166047 2 4 tcp 40 ------- 0 0.0 4.0 0 0
+ 1.166047 4 2 ack 40 ------- 0 4.0 0.0 0 1
- 1.166047 4 2 ack 40 ------- 0 4.0 0.0 0 1
r 1.176153 4 2 ack 40 ------- 0 4.0 0.0 0 1
+ 1.176153 2 1 ack 40 ------- 0 4.0 0.0 0 1
- 1.176153 2 1 ack 40 ------- 0 4.0 0.0 0 1
r 1.326987 2 1 ack 40 ------- 0 4.0 0.0 0 1
+ 1.326987 1 3 ack 40 ------- 0 4.0 0.0 0 1
- 1.326987 1 3 ack 40 ------- 0 4.0 0.0 0 1
r 1.481987 1 3 ack 40 ------- 0 4.0 0.0 0 1
+ 1.481987 3 0 ack 40 ------- 0 4.0 0.0 0 1
- 1.481987 3 0 ack 40 ------- 0 4.0 0.0 0 1
r 1.532093 3 0 ack 40 ------- 0 4.0 0.0 0 1
+ 1.532093 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
- 1.532093 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
+ 1.532093 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
- 1.534867 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
r 1.584867 0 3 tcp 1040 ------- 0 0.0 4.0 1 2
+ 1.584867 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
- 1.584867 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
r 1.58764 0 3 tcp 1040 ------- 0 0.0 4.0 2 3
+ 1.58764 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
- 1.606533 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
r 1.756533 3 1 tcp 1040 ------- 0 0.0 4.0 1 2
+ 1.756533 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
- 1.756533 1 2 tcp 1040 ------- 0 0.0 4.0 1 2
r 1.7782 3 1 tcp 1040 ------- 0 0.0 4.0 2 3
+ 1.7782 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
- 1.886533 1 2 tcp 1040 ------- 0 0.0 4.0 2 3
```

PART-B

Lab Program-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.

```
                    1 0 1 = 5
            -------------
1 0 0 1 1 / 1 1 0 1 1 0 1
            1 0 0 1 1 | |
            --------- | |
              1 0 0 0 0 |
              0 0 0 0 0 |
              --------- |
              1 0 0 0 0 1
                1 0 0 1 1
                ---------
                  1 1 1 0 = 14 = remainder
```

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 easilyincrease 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:

- 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

Below table 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

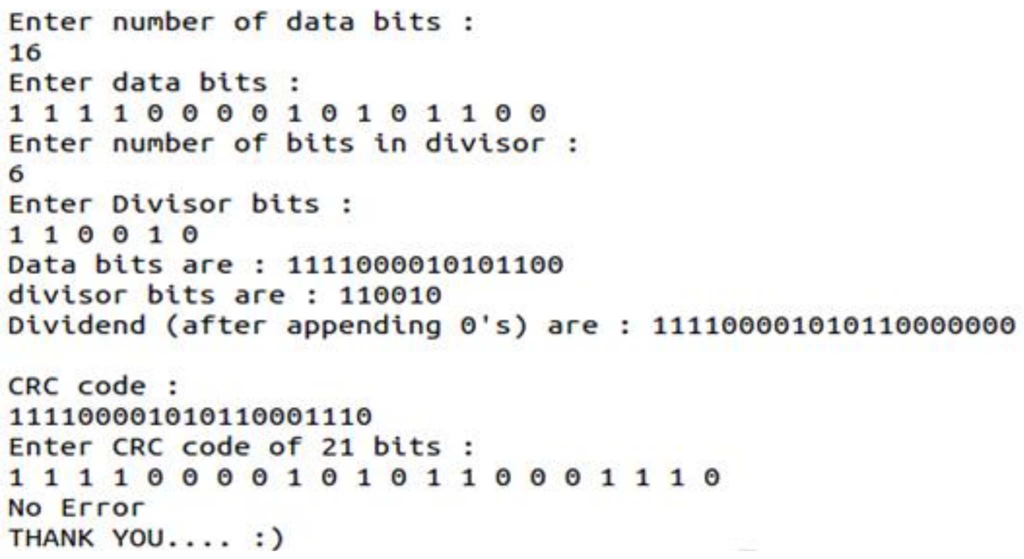
International Standard CRC Polynomials

<u>Source code</u>:

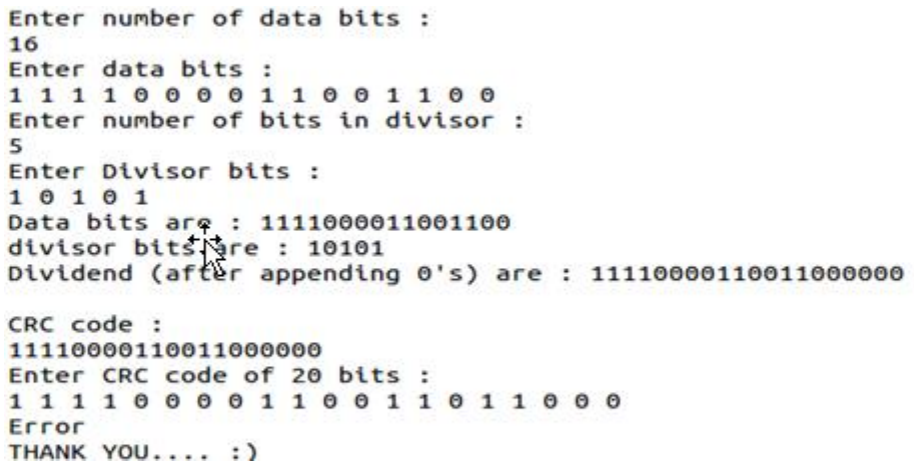
```
import java.io.*;
import java.util.Scanner;
class crc
{
public static void main(String args[])
throws IOException
{
//BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
Scanner scan= new Scanner(System.in);
int[ ] data;
int[ ]div;
int[ ]divisor;
int[ ]rem;
int[ ] crc;
```

```
int data_bits, divisor_bits, tot_length;
System.out.println("Enter number of data bits : ");
data_bits=scan.nextInt();
data=new int[data_bits];
System.out.println("Enter data bits : ");
for(int i=0; i<data_bits; i++)
data[i]=scan.nextInt();
System.out.println("Enter number of bits in divisor : ");
divisor_bits=scan.nextInt();
divisor=new int[divisor_bits];
System.out.println("Enter Divisor bits : ");
for(int i=0; i<divisor_bits; i++)
divisor[i]=scan.nextInt();
System.out.print("Data bits are : ");
for(int i=0; i<data_bits; i++)
System.out.print(data[i]);
System.out.println();
System.out.print("divisor bits are : ");
for(int i=0; i<divisor_bits; i++)
System.out.print(divisor[i]);
System.out.println();
tot_length=data_bits+divisor_bits-1;
div=new int[tot_length];
rem=new int[tot_length];
crc=new int[tot_length];
for(int i=0;i<data.length;i++)
div[i]=data[i];

System.out.print("Dividend (after appending 0's) are : ");
for(int i=0; i< div.length; i++)
System.out.print(div[i]);
System.out.println();
for(int j=0; j<div.length; j++)
{
rem[j] = div[j];
}
rem=divide(div, divisor, rem);
for(int i=0;i<div.length;i++)
{
crc[i]=(div[i]^rem[i]);
}
System.out.println();
System.out.println("CRC code : ");
for(int i=0;i<crc.length;i++)
 System.out.print(crc[i]);
System.out.println();
System.out.println("Enter CRC code of "+tot_length+" bits : ");
for(int i=0; i<crc.length; i++)
crc[i]=scan.nextInt();
for(int j=0; j<crc.length; j++)
{
rem[j] = crc[j];
}
rem=divide(crc, divisor, rem);
for(int i=0; i< rem.length; i++)
{
if(rem[i]!=0)
{
System.out.println("Error");
break;
}
if(i==rem.length-1)
System.out.println("No Error");
}
System.out.println("THANK YOU.... :)");
}
static int[] divide(int div[],int divisor[], int rem[])
{
int cur=0; while(true)
{
for(int i=0;i<divisor.length;i++)
rem[cur+i]=(rem[cur+i]^divisor[i]);
while(rem[cur]==0 && cur!=rem.length-1) cur++;
if((rem.length-cur)<divisor.length)
break;
}
return rem;
}
}
```

Output:



```
Enter number of data bits :
16
Enter data bits :
1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 0
Enter number of bits in divisor :
6
Enter Divisor bits :
1 1 0 0 1 0
Data bits are : 1111000010101100
divisor bits are : 110010
Dividend (after appending 0's) are : 111100001010110000000

CRC code :
111100001010110001110
Enter CRC code of 21 bits :
1 1 1 1 0 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 0
No Error
THANK YOU.... :)
```



```
Enter number of data bits :
16
Enter data bits :
1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 0
Enter number of bits in divisor :
5
Enter Divisor bits :
1 0 1 0 1
Data bits are : 1111000011001100
divisor bits are : 10101
Dividend (after appending 0's) are : 11110000110011000000

CRC code :
11110000110011000000
Enter CRC code of 20 bits :
1 1 1 1 0 0 0 0 1 1 0 0 1 1 0 1 1 0 0 0
Error
THANK YOU.... :)
```

Lab Program-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 neighbors of its routing table. For each network path, the receiving routers pick the neighbor 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: the Bellman-Ford and the 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 neighboring 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 neighbors 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.

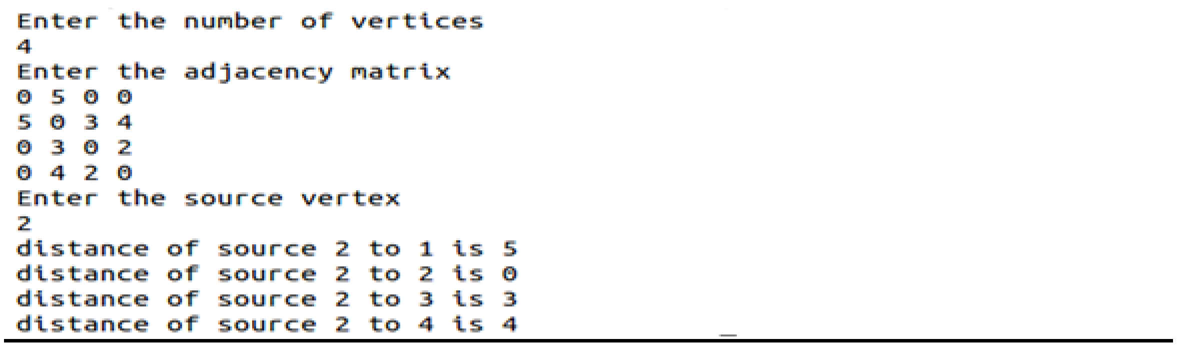
Source code:

```
import java.util.Scanner;
public class BellmanFord
{
private int D[];
private int num_ver;
public static final int MAX_VALUE = 999;
public BellmanFord(int num_ver)
{
this.num_ver = num_ver;
D = new int[num_ver + 1];
}
public void BellmanFordEvaluation(int source, int A[][])
{
for (int node = 1; node <= num_ver; node++)
{
D[node] = MAX_VALUE;
}
D[source] = 0;
for (int node = 1; node <= num_ver - 1; node++)
{
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
if (A[sn][dn] != MAX_VALUE)
{
if (D[dn] > D[sn]+ A[sn][dn])
D[dn] = D[sn] + A[sn][dn];
}
}
```

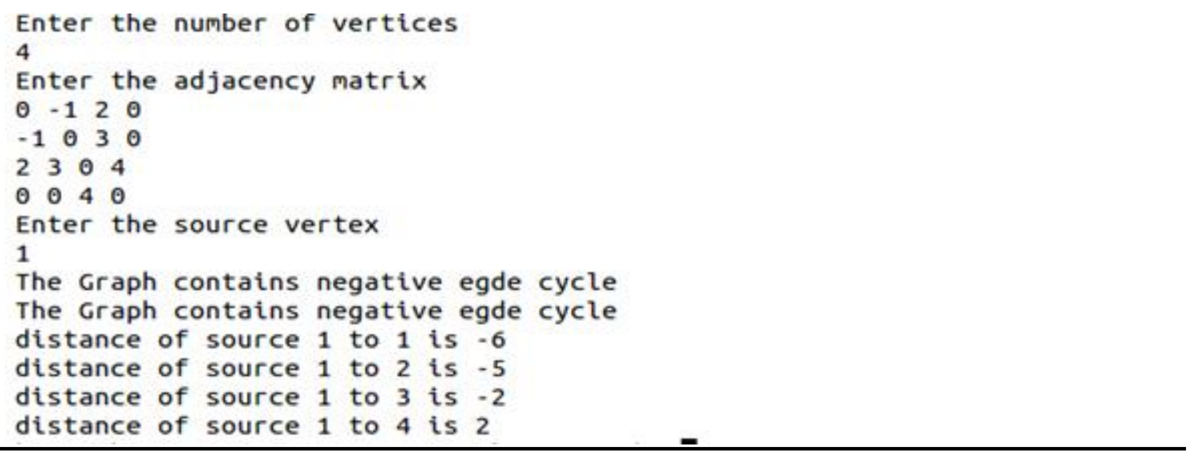
```
}
}
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
if (A[sn][dn] != MAX_VALUE)
{
if (D[dn] > D[sn]+ A[sn][dn])
System.out.println("The Graph contains negative egde cycle");
}
}
}
for (int vertex = 1; vertex <= num_ver; vertex++)
{
System.out.println("distance of source " + source + " to "+ vertex + " is " + D[vertex]);
}
}

public static void main(String[ ] args)
{
int num_ver = 0; int source;
Scanner scanner = new Scanner(System.in);
System.out.println("Enter the number of vertices");
num_ver = scanner.nextInt();
int A[][] = new int[num_ver + 1][num_ver + 1];
System.out.println("Enter the adjacency matrix");
for (int sn = 1; sn <= num_ver; sn++)
{
for (int dn = 1; dn <= num_ver; dn++)
{
A[sn][dn] = scanner.nextInt();
if (sn == dn)
{
```

```
A[sn][dn] = 0;
continue;
}
if (A[sn][dn] == 0)
{
A[sn][dn] = MAX_VALUE;
}
}
}
System.out.println("Enter the source vertex");
source = scanner.nextInt();
BellmanFord b = new BellmanFord (num_ver);
b.BellmanFordEvaluation(source, A);
scanner.close();
}
}
```

OUTPUT:



```
Enter the number of vertices
4
Enter the adjacency matrix
0 5 0 0
5 0 3 4
0 3 0 2
0 4 2 0
Enter the source vertex
2
distance of source 2 to 1 is 5
distance of source 2 to 2 is 0
distance of source 2 to 3 is 3
distance of source 2 to 4 is 4
```



```
Enter the number of vertices
4
Enter the adjacency matrix
0 -1 2 0
-1 0 3 0
2 3 0 4
0 0 4 0
Enter the source vertex
1
The Graph contains negative egde cycle
The Graph contains negative egde cycle
distance of source 1 to 1 is -6
distance of source 1 to 2 is -5
distance of source 1 to 3 is -2
distance of source 1 to 4 is 2
```

Lab Program-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. Implement the above program using as message queues or FIFOs as IPC channels.

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:

TCP Client

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.EOFException;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.Scanner;

class client
{
public static void main(String args[])throws Exception
{
String address = "";
Scanner sc=new Scanner(System.in);
System.out.println("Enter Server Address: ");
address=sc.nextLine();
```

```
Socket s=new Socket(address,5000);

DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
System.out.println("Send Get to start...");
String str="",filename="";
try
{
while(!str.equals("start"))
 str=br.readLine();
dout.writeUTF(str);
dout.flush();

//read the file name
System.out.println("Enter File Name: ");
//Scanner sc=new Scanner(System.in);
filename=sc.nextLine();
sc.close();
dout.writeUTF(filename);



//filename=din.readUTF();
System.out.println("Receving file: "+filename);
filename="client"+filename;
System.out.println("Saving as file: "+filename);
long sz=Long.parseLong(din.readUTF());
System.out.println ("File Size: "+(sz/(1024*1024))+" MB");
byte b[]=new byte [1024];
System.out.println("Receving file..");
FileOutputStream fos=new FileOutputStream(new File(filename),true);
long bytesRead;
do
{
bytesRead = din.read(b, 0, b.length);
fos.write(b,0,b.length);
}while(!(bytesRead<1024));
System.out.println("Comleted");
fos.close();
dout.close();
s.close();
}
catch(EOFException e)
{
}
}
}
```

Tcp Serversocket

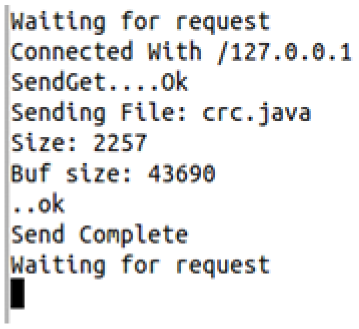
```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;
class server
{
public static void main(String args[])throws Exception
{
/*String filename;
System.out.println("Enter File Name: ");
Scanner sc=new Scanner(System.in);
filename=sc.nextLine();
sc.close(); */

while(true)
```

```
{
//create server socket on port 5000
ServerSocket ss=new ServerSocket(5000);
System.out.println ("Waiting for request");
Socket s=ss.accept();
System.out.println ("Connected With "+s.getInetAddress().toString());
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());

try
{
String str="";
str=din.readUTF();
System.out.println("SendGet....Ok");
if(!str.equals("stop"))
{

//redaing file name from client
String filename="";
filename=din.readUTF();

System.out.println("Sending File: "+filename);
//dout.writeUTF(filename);
//dout.flush();
File f=new File(filename);
FileInputStream fin=new FileInputStream(f);
long sz=(int) f.length();
byte b[]=new byte [1024]; int read;
dout.writeUTF(Long.toString(sz));
dout.flush();
System.out.println ("Size: "+sz);

System.out.println ("Buf size: "+ss.getReceiveBufferSize());
while((read = fin.read(b)) != -1)
{
dout.write(b, 0, read);
dout.flush();
}
fin.close();
System.out.println("..ok");
dout.flush();
}
dout.writeUTF("stop");
System.out.println("Send Complete");
dout.flush();
}
catch(Exception e)
{
e.printStackTrace();
System.out.println("An error occured");
}
din.close();
s.close();
ss.close();
}
}
}
```

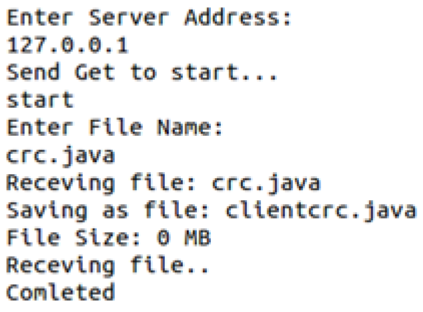
Note:

- Create two different files Client.java and Server.java. Follow the steps given:
- Open a terminal run the server program and provide the filename to send
- Open one more terminal run the client program and provide the IP address of the server. We can give localhost address “127.0.0.1” as it is running on same machine or give the IP address of the machine.
- Send any start bit to start sending file.

Output:



```
Waiting for request
Connected With /127.0.0.1
SendGet....Ok
Sending File: crc.java
Size: 2257
Buf size: 43690
..ok
Send Complete
Waiting for request
```



```
Enter Server Address:
127.0.0.1
Send Get to start...
start
Enter File Name:
crc.java
Receving file: crc.java
Saving as file: clientcrc.java
File Size: 0 MB
Receving file..
Comleted
```

Lab Program-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:

Udpclient.Java

```
import java.net.*;
public class udpclient{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Udpserver.Java

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

public class udpserver{
public static void main(String[] args) throws Exception {
DatagramSocket ds = new DatagramSocket();
String str = "";
Scanner sc=new Scanner(System.in);

    System.out.println("\n Enter the message : ");
```

```
        str=sc.nextLine();

      InetAddress ip = InetAddress.getByName("127.0.0.1");

      DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
      ds.send(dp);
      ds.close();
    }
  }
```

Output:

At Server side:

```
Enter the message:
Hello
```

At Client side:

```
Hello
```

Lab Program-11

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

Algorithm

1. Generate two large random primes, P and Q, of approximately equal size.
2. Compute N = P x Q
3. Compute $Z = (P-1) \times (Q-1)$.
4. Choose an integer E, $1 < E < Z$, such that GCD $(E, Z) = 1$
5. Compute the secret exponent D, $1 < D < Z$, such that $E \times D \equiv 1 \pmod{Z}$
6. The public key is (N, E) and the private key is (N, D).

Note: The values of P, Q, and Z should also be kept secret.

The message is encrypted using public key and decrypted using private key.

An example of RSA encryption

1. Select primes $P=11$, $Q=3$.
2. $N = P \times Q = 11 \times 3 = 33$

$Z = (P-1) \times (Q-1) = 10 \times 2 = 20$

3. Lets choose $E=3$ Check GCD$(E, P-1)$ = GCD(3, 10) = 1 (i.e. 3 and 10 have no common factors except 1), and check GCD$(E, Q-1)$ = GCD(3, 2) = 1

therefore GCD(E, Z) = GCD(3, 20) = 1

4. Compute D such that $E \times D \equiv 1 \pmod{Z}$

compute $D = E^{\wedge -1} \bmod Z = 3^{\wedge -1} \bmod 20$

find a value for D such that Z divides $((E \times D)-1)$

find D such that 20 divides 3D-1. Simple testing (D = 1, 2, ...) gives D = 7

Check: $(E \times D)-1 = 3.7 - 1 = 20$, which is divisible by Z.

5. Public key = (N, E) = (33, 3) Private key = (N, D) = (33, 7).

Now say we want to encrypt the message m = 7,

Cipher code = $M^{\wedge E} \bmod N$

= $7^{\wedge 3} \bmod 33$

= 343 mod 33

= 13.

Hence the ciphertext c = 13.

To check decryption we compute Message' = $C^{\wedge D} \bmod N$

= $13^{\wedge 7} \bmod 33$

= 7.

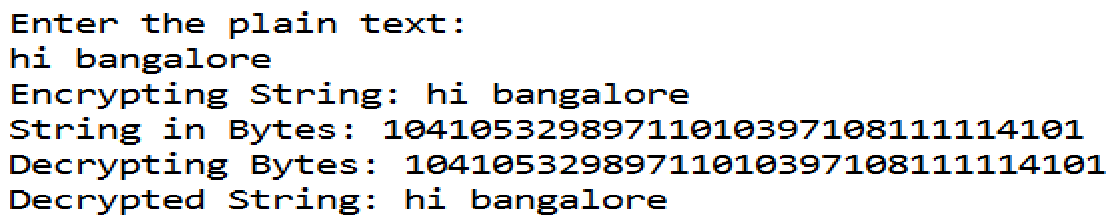
Note that we don't have to calculate the full value of 13 to the power 7 here. We can make use of the fact that a = bc mod n = (b mod n).(c mod n) mod n so we can break down a potentially large number into its components and combine the results of easier, smaller calculations to calculate the final value.

Source Code:

Program:

```
import java.io.DataInputStream;
import java.io.IOException;
import java.math.BigInteger;
import java.util.Random;
public class RSA
{
 private BigInteger p;
private BigInteger q;
private BigInteger N;
private BigInteger phi;
private BigInteger e;
private BigInteger d;
private int bitlength = 1024;
private Random r;
public RSA()
{
r = new Random();
p = BigInteger.probablePrime(bitlength, r);
q = BigInteger.probablePrime(bitlength, r);
N = p.multiply(q);
phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
e = BigInteger.probablePrime(bitlength / 2, r);
while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 && e.compareTo(phi) < 0)
{
e.add(BigInteger.ONE);
}
d = e.modInverse(phi);
}
public RSA(BigInteger e, BigInteger d, BigInteger N)
{
this.e = e;
this.d = d;
this.N = N;
}
@SuppressWarnings("deprecation")
public static void main(String[] args) throws IOException
{
RSA rsa = new RSA();
DataInputStream in = new DataInputStream(System.in);
String teststring;
System.out.println("Enter the plain text:");
teststring = in.readLine();
System.out.println("Encrypting String: " + teststring);
System.out.println("String in Bytes: " + bytesToString(teststring.getBytes()));
// encrypt
byte[] encrypted = rsa.encrypt(teststring.getBytes());
// decrypt
byte[] decrypted = rsa.decrypt(encrypted);
System.out.println("Decrypting Bytes: " + bytesToString(decrypted));
System.out.println("Decrypted String: " + new String(decrypted));
}
private static String bytesToString(byte[] encrypted)
{
String test = "";
for (byte b : encrypted)
{
test += Byte.toString(b);
}
return test;
}
// Encrypt message
public byte[] encrypt(byte[] message)
{
return (new BigInteger(message)).modPow(e, N).toByteArray();
}
// Decrypt message
public byte[] decrypt(byte[] message)
{
return (new BigInteger(message)).modPow(d, N).toByteArray();
}

 }
```

Output:

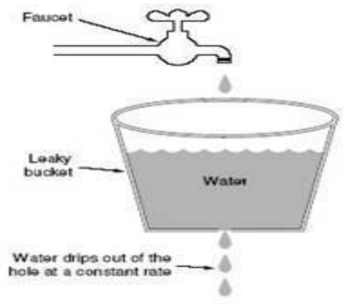


```
Enter the plain text:
hi bangalore
Encrypting String: hi bangalore
String in Bytes: 104105329897110103971081111114101
Decrypting Bytes: 104105329897110103971081111114101
Decrypted String: hi bangalore
```

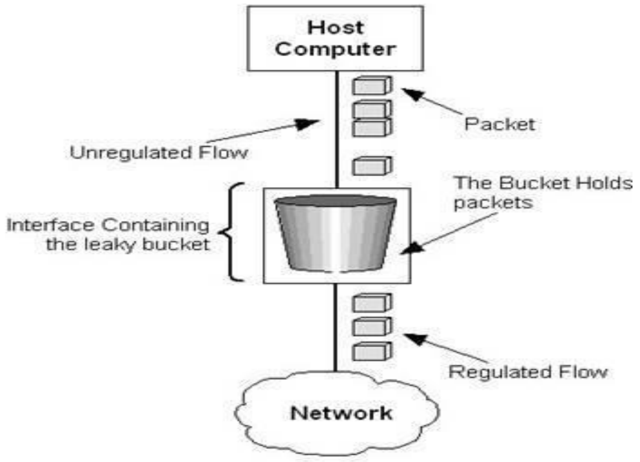
Lab Program-12

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 spillover. 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 and with packets (b)).



(a) A leaky bucket with water



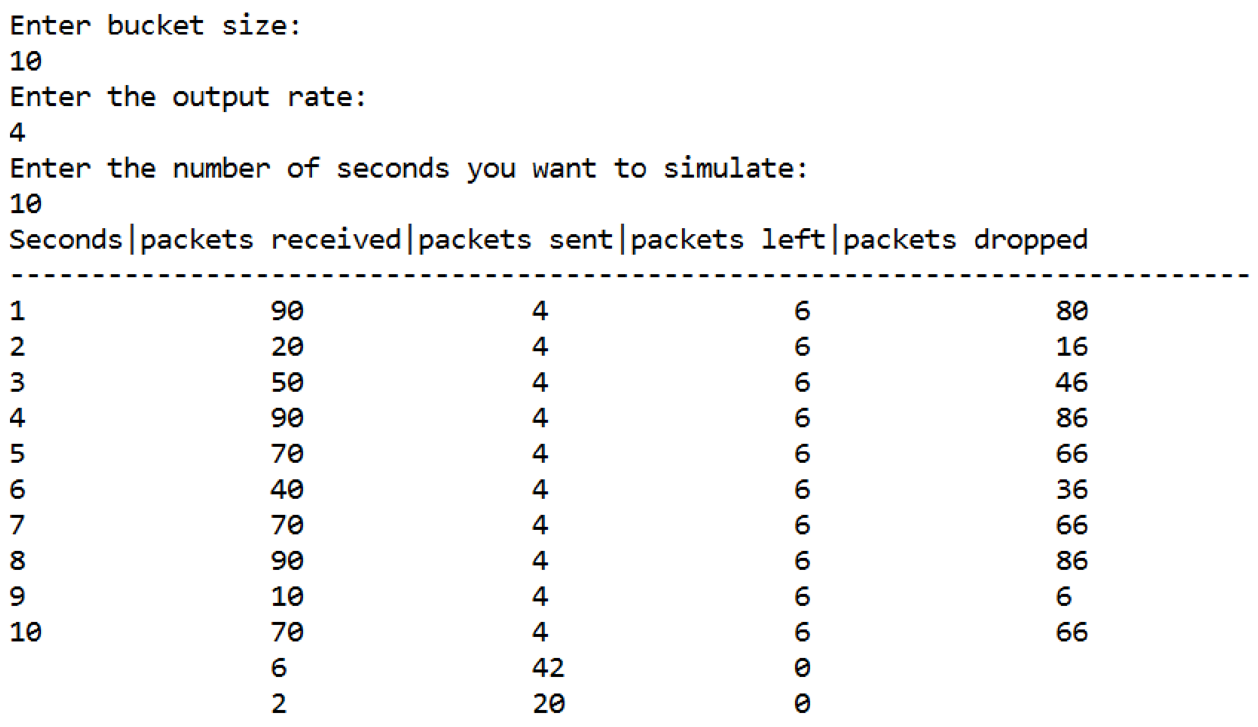
(b) A leaky bucket with packets

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.lang.*;
import java.util.Random;
import java.io.*;
import java.util.Scanner;
class leaky_bucket
{
public static void main(String args[])
{
int drop=0,mini,nsec,p_remain=0;
int o_rate,b_size,i,packet[];
packet = new int[100];
Scanner in = new Scanner(System.in);
System.out.println("Enter bucket size:");
b_size = in.nextInt();
System.out.println("Enter the output rate:");
o_rate = in.nextInt();
System.out.println("Enter the number of seconds you want to simulate:");
nsec = in.nextInt();
Random rand = new Random();
for(i=0;i<nsec;i++)
packet[i]=((rand.nextInt(9)+1)*10);
System.out.println("Seconds|packets received|packets sent|packets left|packets dropped");
System.out.println("--------------------------------------------------------------------------");
for(i=0;i<nsec;i++)
{
p_remain+=packet[i];
if(p_remain>b_size)
{
drop=p_remain-b_size;
p_remain=b_size;
System.out.print(i+1+" ");
System.out.print(packet[i]+" ");
mini=Math.min(p_remain,o_rate);
System.out.print(mini+" ");
p_remain=p_remain-mini;
 System.out.print(p_remain+" ");
System.out.print(drop+" ");
System.out.println();
drop=0;
}
}
while(p_remain!=0)
{
if(p_remain>b_size)
{
drop=p_remain-b_size;
p_remain=b_size;
}
mini=Math.min(p_remain,o_rate);
System.out.print(" "+p_remain+" "+mini);
p_remain=p_remain-mini;
System.out.println(p_remain+" "+drop);
drop=0;
}
}
 }
```

Output



```
Enter bucket size:
10
Enter the output rate:
4
Enter the number of seconds you want to simulate:
10
Seconds|packets received|packets sent|packets left|packets dropped
---------------------------------------------------------------------------
1               90              4               6               80
2               20              4               6               16
3               50              4               6               46
4               90              4               6               86
5               70              4               6               66
6               40              4               6               36
7               70              4               6               66
8               90              4               6               86
9               10              4               6               6
10              70              4               6               66
                6               42              0
                2               20              0
```